ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
UNIVERSITY OF PIRAEUS

Π.Μ.Σ.: "ΠΛΗΡΟΦΟΡΙΚΗ"

ΚΥΚΛΟΣ ΣΠΟΥΔΩΝ: 9ος
ΕΞΑΜΗΝΟ: Δ'

ΔΙΠΛΩΜΑΤΙΚΗ ΔΙΑΤΡΙΒΗ: ΠΛΗΡΟΦΟΡΙΑΚΟ ΣΥΣΤΗΜΑ ΟΦΘΑΛΜΟΛΟΓΙΚΗΣ ΚΛΙΝΙΚΗΣ

ΣΤΡΑΤΗΣ ΙΩΑΝΝΗΣ (ΑΜ: MPPL14083)

Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Πληροφοριακό Σύστημα Οφθαλμολογικής Κλινικής** **Ophthalmological Clinic Intergraded Information System**
Ονοματεπώνυμο Φοιτητή	**Ιωάννης Στρατής**
Πατρώνυμο	**Δημήτριος**
Αριθμός Μητρώου	**ΜΠΠΛ/ 14083**
Επιβλέπων	**Ευθύμης Αλέπης, Καθηγητής**

Ημερομηνία Παράδοσης: **10/2016**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Όνομα Επώνυμο Βαθμίδα	Όνομα Επώνυμο Βαθμίδα	Όνομα Επώνυμο Βαθμίδα

Αφιερώνεται στην οικογένεια μου,
η οποία μου στάθηκε αρωγός στην προσπάθεια μου
να ανταπεξέλθω στις απαιτήσεις
του Προγράμματος Μεταπτυχιακών Σπουδών «Πληροφορική»
του Πανεπιστημίου Πειραιά.
Επιπλέον, αφιερώνεται στους γονείς μου,
στους οποίους οφείλω ότι έχω πετύχει στην ζωή μου μέχρι σήμερα.
Τέλος, ευχαριστώ θερμά τους καθηγητές μου
για την συμβολή τους στο ΠΜΣ και
τις γνώσεις που μου μετέδωσαν,
τις οποίες είναι βέβαιο ότι θα αξιοποιήσω στην περαιτέρω σταδιοδρομία μου.

Περιεχόμενα

Περιεχόμενα Εικόνων

1 Εισαγωγή

1.1 Περίληψη

Στα κεφάλαια της παρούσας Διπλωματικής Εργασίας, που ακολουθούν, αναλύονται τα βήματα που απαιτήθηκαν για την ανάπτυξη ενός Πληροφοριακού Συστήματος μίας Οφθαλμολογικής Κλινικής. Η υλοποίηση του Πληροφοριακού Συστήματος στηρίχτηκε σε καινούριες τεχνολογίες ανάπτυξης λογισμικού, τις οποίες προσφέρει το .Net Framework, όπως WPF,WCF και Entity.

Αρχικά, θα καθοριστούν οι απαιτήσεις και οι παραδοχές για το Πληροφοριακό Σύστημα και στην συνέχεια θα αναλυθεί ο σχεδιασμός του, με την βοήθεια κατάλληλων διαγραμμάτων UML. Έπειτα, θα προχωρήσουμε στην υλοποίηση των απαιτήσεων, όπου θα λάβουν χώρα επιμέρους συζητήσεις σε σημεία του κώδικα, τα οποία χρίζουν μεγαλύτερης προσοχής.

Θα επιχειρηθεί μία λεπτομερής ανάλυση της κατασκευής του σχήματος της Βάσης Δεδομένων, και θα γίνει αναφορά στα οφέλη που επιφέρει η χρήση του κάθε Framework στην ανάπτυξη κώδικα, κατά την υλοποίηση των απαιτήσεων της εφαρμογής.

Τέλος, θα γίνει μία προσπάθεια εξαγωγής συμπερασμάτων, σε ότι αφορά την εμπειρία του προγραμματιστή με την χρήση αυτών των νέων προγραμματιστικών τεχνολογιών ενώ θα ακολουθήσουν προτάσεις για ενδεχόμενες προσθήκες λειτουργιών στο υφιστάμενο σύστημα.

Όπως θα γίνει προφανές, για την ανάπτυξη του ΠΣ απαιτήθηκε ο συνδυασμός γνώσεων από διάφορα γνωστικά πεδία της Επιστήμης της Πληροφορικής, με στόχο την εφαρμογή των απαιτούμενων κανόνων, ώστε στο τέλος να προκύψει ένα αξιόπιστο λογισμικό.

1.2 Abstract

The provided Chapters of this Master Thesis, include all the steps that are required to develop an Intergraded Information System, which will be used by an Ophthalmological Clinic. The implementation of this System has taken place with the support of some new software development technologies, which are provided by the .Net Framework, like WPF, WCF and Entity.

Initially, we will define the requirements and some assumptions, regarding the Information System, followed by the design analysis, which will be accompanied by proper UML diagrams. Then, we will proceed to the implementation of those requirements, while there will be held discussion on the code, where is needed.

We will attempt a thorough analysis, regarding the Database schema creation as well as we will refer to the benefits of each implemented Framework in our code.

Finally, we will arrive to some conclusions, regarding the developer's experience on that new software development technologies usage, followed by suggestions, including any possible function additions to the existing Information System.

Soon, there will be obvious, that the combined knowledge usage coming from various Computer Science fields has been significant, in order to implement the various necessary development rules which those fields define, and finally end up with the development of an Information System, which is reliable.

1.3 Σκοπός της Εργασίας

Στο πλαίσιο δημιουργίας της παρούσας Διπλωματικής Εργασίας γίνεται μία προσπάθεια αξιοποίησης των γνώσεων που αποκτήθηκαν κατά την διάρκεια της φοιτήσεως στο Πρόγραμμα Μεταπτυχιακών Σπουδών «Πληροφορική» του Πανεπιστημίου Πειραιώς. Συγκεκριμένα, η υλοποίηση της Διπλωματικής Εργασίας (από εδώ και στο εξής ΔΕ) απαιτεί την κατανόηση σε ικανοποιητικό βαθμό των παρακάτω μαθημάτων:

- Αρχές Προγραμματισμού – Γλώσσα C
- Δομές Δεδομένων
- Αλληλεπίδραση Ανθρώπου – Υπολογιστή
- Βάσεις Δεδομένων
- Ταχεία Ανάπτυξη Εφαρμογών
- Δίκτυα Υπολογιστών
- Τεχνολογία Λογισμικού
- Ειδικά Θέματα Αντικειμενοστραφούς Προγραμματισμού
- Ιατρική Πληροφορική

Ο συνδυασμός των γνώσεων που προσφέρουν οι παραπάνω επιστημονικοί τομείς οδηγούν δυνητικά στην απόκτηση ενός συγκεκριμένου τρόπου σκέψης σε ότι αφορά τον τομέα ανάπτυξης και σχεδίασης λογισμικού, ο οποίος καταλήγει να είναι ανεξάρτητος της γλώσσας προγραμματισμού καθώς και των εργαλείων που χρησιμοποιούνται.

Στην παρούσα ΔΕ επιχειρείται η σχεδίαση και υλοποίηση ενός Ολοκληρωμένου Πληροφοριακού Συστήματος (από εδώ και στο εξής ΠΣ) στον τομέα της Ιατρικής Επιστήμης. Ωστόσο, εξαιτίας της μεγάλης ποικιλίας εξειδικεύσεων, που παρουσιάζει ο υπόψη επιστημονικός χώρος, επιλέχθηκε η υλοποίηση του εν λόγω ΠΣ να αφορά αποκλειστικά την Οφθαλμολογική Ειδικότητα.

Συγκεκριμένα, το ΠΣ αφορά στην αυτοματοποίηση και μηχανογράφηση των καθημερινών ενεργειών και υποχρεώσεων, που έχει το προσωπικό μίας οφθαλμολογικής κλινικής, με βασικό γνώμονα την επιτυχή διεκπεραίωση τους, στον λιγότερο δυνατό χρόνο, διαφυλάσσοντας πάντα την ακεραιότητα των δεδομένων και την αξιοπιστία του συστήματος.

Συνεπώς, η πληροφορία που αποθηκεύεται στην Βάση Δεδομένων (από εδώ και στο εξής ΒΔ), θα πρέπει να είναι ασφαλής και να προστατεύεται από κακόβουλους εξωτερικούς παράγοντες. Οι χρήστες της ΒΔ, και γενικότερα του ΠΣ, θα ανήκουν είτε στο ιατρικό προσωπικό είτε στην γραμματειακή υποστήριξη της κλινικής. Θα είναι σε θέση να επεξεργάζονται, να εισάγουν καινούρια στοιχεία και τέλος να διαγράφουν ήδη υπάρχοντα δεδομένα. Ωστόσο, λόγω της ιδιαιτερότητας που παρουσιάζει η φύση της ιατρικής πληροφορίας, αναφορικά με την προστασία προσωπικών δεδομένων, δεν δίνονται δικαιώματα πλήρους πρόσβασης στο προσωπικό, το οποίο εκτελεί χρέη γραμματείας.

Επιγραμματικά, οι κύριες λειτουργίες που πρέπει να φέρει εις πέρας το ΠΣ είναι οι εξής:

- Διαχείριση Ιατρικού Φάκελου Ασθενούς
- Διαχείριση και Προγραμματισμός Ραντεβού
- Ψηφιακή Αρχειοθέτηση Φωτογραφιών με Ιατρικό Περιεχόμενο
- Δημιουργία Αναφορών
- Εξαγωγή Στατιστικών Δεδομένων
- Επικοινωνία με εξωτερική ΒΔ, με σκοπό την διασταύρωση στοιχείων ασθενούς
- Επικοινωνία προσωπικού της κλινικής με τους ασθενείς της μέσω ηλεκτρονικής αλληλογραφίας
- Πρόβλεψη backup – restore της ΒΔ

Γίνεται εύκολα αντιληπτό, ότι ο κώδικας, ο οποίος θα υλοποιήσει τις παραπάνω λειτουργίες θα πρέπει να είναι καλά δομημένος, διότι, προσεγγίζοντας το θέμα ρεαλιστικά, οι παραπάνω λειτουργίες θα εμπλουτίζονται διαρκώς με νέες απαιτήσεις, κάτι που προσδίδει μία δυναμικότητα στην ίδια την εφαρμογή. Συνεπώς, ο κώδικας θα πρέπει να φέρει ορισμένα βασικά χαρακτηριστικά, όπως:

- να είναι αξιόπιστος,
- να είναι εύκολα παραμετροποιήσιμος,
- να είναι εύκολα αναβαθμίσιμος,
- να μην φέρει κενά ασφαλείας

Σε ότι αφορά την γλώσσα προγραμματισμού, τις προγραμματιστικές αρχές και τεχνικές που αξιοποιήθηκαν, λήφθηκε σοβαρά υπόψη τόσο η υφιστάμενη τεχνολογική κατάσταση, σε επίπεδο Λειτουργικών Συστημάτων (ΛΣ), όσο και οι διάφορες προγραμματιστικές τάσεις που παρατηρούνται σε παγκόσμιο επίπεδο. Πλέον, ήδη τα Windows 7 και 10 καλύπτουν τις απαιτήσεις του 70% των χρηστών ηλεκτρονικών υπολογιστών και είναι εμφανείς οι διαφορές των εν λόγω εκδόσεων ΛΣ σε σχέση με τα παλαιότερα συστήματα. Ειδικά για τα Windows 10, οι διαφορές αυτές μας επιτρέπουν να μιλάμε όχι απλά για αναβάθμιση εκδόσεων ΛΣ αλλά για υιοθέτηση καινούριων τεχνολογιών και ριζική αλλαγή της προγραμματιστικής προσέγγισης σε αρκετά υποσυστήματα τους. Για αυτό τον λόγο κρίθηκε σκόπιμη η υλοποίηση του ΠΣ υπό το GUI Framework των Windows Presentation Forms (WPF), το οποίο πηγάζει από το .NET Framework.

Εφαρμόζοντας το ίδιο σκεπτικό εκμετάλλευσης προγραμματιστικών τεχνικών νέας γενιάς, λήφθηκαν σοβαρά υπόψη οι τάσεις της κοινότητας των προγραμματιστών σε ότι αφορά την αποθήκευση, επεξεργασία και ανάκτηση πληροφορίας από μία ΒΔ καθώς και την επικοινωνία της με το υπόλοιπο σώμα ενός προγράμματος. Συνεπώς, η υλοποίηση της επικοινωνίας των λειτουργών του ΠΣ με την ΒΔ γίνεται με την βοήθεια του Entity Framework, το οποίο εφαρμόζει μία τεχνική μοντελοποίησης του σχήματος μιας ΒΔ, που έρχεται σε πλήρη αντιστοιχία με τις αρχές του αντικειμενοστραφούς προγραμματισμού, διευκολύνοντας ιδιαίτερα τον προγραμματιστή, όπως θα αναλύσουμε στην συνέχεια.

Όπως ήδη αναφέρθηκε, η σωστή διαχείριση και εκμετάλλευση της ιατρικής πληροφορίας αποτελεί μονόδρομο σε παγκόσμιο επίπεδο. Ωστόσο, ακόμα και στις μέρες μας, ένας ασθενής δεν γνωρίζει με λεπτομέρεια το ιστορικό των διάφορων ασθενειών/παθήσεων του. Συνεπώς έχουμε το φαινόμενο να χάνεται αυτή η τόσο σημαντική πληροφορία. Παρόλα αυτά, εάν μπορούσαμε να εφαρμόσουμε ένα συγκεντρωτικό σύστημα, στο οποίο η εν λόγω πληροφορία να αποθηκεύονταν και να γινόταν προσπελάσιμη μόνο από συγκεκριμένα άτομα, ακολουθώντας τις αρχές προστασίας προσωπικών δεδομένων, θα επέφερε μόνο πλεονεκτήματα τόσο στο ιατρικό σύστημα μίας χώρας όσο και στο ίδιο τον ασθενή. Έτσι λοιπόν, γίνεται μια προσπάθεια στην παρούσα ΔΕ να υλοποιηθεί μία επικοινωνία της ΒΔ του ΠΣ μας, με μία εξωτερική ΒΔ, η οποία υποθέτουμε ότι ανήκει σε έναν κρατικό μηχανισμό και ελέγχεται από αυτόν. Για την Ελλάδα, αυτός ο μηχανισμός θα μπορούσε να είναι ο ΕΟΟΠΥ. Με την βοήθεια του Framework WCF έχουν δημιουργηθεί web services, τα οποία καταναλώνονται από το ΠΣ της κλινικής, και αποδεικνύουν ότι το σενάριο που μόλις περιγράψαμε είναι εφικτό και μόνο οφέλη μπορεί να προσφέρει.

Τέλος, επιχειρώντας να πάρουμε ένα στιγμιότυπο της σχέσης που έχει η Ελλάδα με τις παραπάνω τεχνολογικές τάσεις θα διαπιστώσουμε ότι δεν είναι στο καλύτερο δυνατό επίπεδο. Αυτό είναι εμφανές καθώς οι περισσότεροι Οργανισμοί στην Χώρα μας δεν προβαίνουν στις απαραίτητες αναβαθμίσεις σε υλικό, προκειμένου να είναι δυνατή η εγκατάσταση ΛΣ νέας γενιάς. Εξαίρεση δεν αποτελούν ούτε τα δημόσια νοσοκομεία, γεγονός το οποίο εμποδίζει την δημιουργία γέφυρας ανάμεσα στην Πληροφορική και της Ιατρική στην Χώρα μας. Ωστόσο, λόγω της μεγάλης σημασίας που έχει η διαχείριση της ιατρικής πληροφορίας παγκοσμίως, αποτελεί θέμα χρόνου για την Χώρα μας, να υιοθετήσει τις τεχνολογικές εξελίξεις σε όλα τα επίπεδα (hardware - software) και κατ' επέκταση να προβεί σε αναβάθμισή του ιατρικού λογισμικού, το οποίο να εκμεταλλεύεται τις

δυνατότητες των καινούριων προγραμματιστικών τάσεων. Συνεπώς, η απόκτηση τεχνογνωσίας στις νέες προγραμματιστικές τεχνικές αποτελεί μονόδρομο, και κάτω από αυτή την αρχή, υλοποιήθηκε η παρούσα ΔΕ.

Κλείνοντας, στο πλαίσιο εκπόνησης εργασίας στο μάθημα «Ιατρική Πληροφορική» κατά το Δ' Εξάμηνο φοίτησης, απαιτήθηκε ο σχεδιασμός και η υλοποίηση ενός διαδικτυακού Ιστότοπου, στον οποίο να μπορεί να συνδέεται ο ασθενής και να εκτελεί διάφορες λειτουργίες, όπως να κλείνει ραντεβού, να επεξεργάζεται το προφίλ του κ.α. Η υλοποίηση έγινε σε περιβάλλον ASP.NET και αφορά στην δημιουργία Ιστότοπου, ο οποίος δεν έχει διαφημιστικό ρόλο για την κλινική αλλά ένα παρέχει διευκολύνσεις στους πελάτες-ασθενείς της, σε μερικές ενέργειες που τους επιτρέπεται να εκτελέσουν. Στην παρούσα ΔΕ, θα αναφερθούμε επιγραμματικά σε αυτές τις λειτουργίες, που προσφέρει η υλοποίηση του portal της κλινικής, χωρίς να μπούμε σε λεπτομέρειες.

2 Ανάπτυξη Πληροφοριακού Συστήματος

2.1 Εισαγωγή

Κατά την διάρκεια ανάπτυξης ενός λογισμικού, καθορίζονται τρεις βασικές διαδικασίες, σύμφωνα με τα πρότυπα IEEE 1074-1995. Αυτές είναι οι εξής:

α) Καθορισμός Απαιτήσεων

β) Σχεδιασμός

γ) Υλοποίηση

Στο στάδιο του καθορισμού απαιτήσεων, ορίζεται το τι πρέπει να κάνει ένα σύστημα λογισμικού, και παρέχεται μια μηχανική περιγραφή των αντικειμένων, των λειτουργιών, και των καταστάσεων ενός συστήματος λογισμικού. Κατά τη διάρκεια της διαδικασίας των απαιτήσεων αναπτύσσονται οι προτεραιότητες, η ολοκλήρωση του λογισμικού και οι ανάγκες της διεπαφής, τα μοντέλα ροής δεδομένων πληροφοριών, η λεπτομερής ανάλυση των κινδύνων, και οι δοκιμές και σχέδια εγκατάστασης. Τα επίσημα μοντέλα που εκθέτουν τη δομή των συστημάτων λογισμικού (εισόδους, εξόδους, και συνδέσεις μεταξύ των δραστηριοτήτων) είναι συνήθως ένα αποτέλεσμα της διαδικασίας απαιτήσεων.

Η φάση σχεδιασμού εστιάζει στον τρόπο με τον οποίο οι απαιτήσεις λογισμικού μπορούν να πραγματοποιηθούν. Σε αυτό το στάδιο, οι κύριες ανησυχίες είναι οι λειτουργίες και η δομή του συστήματος που αναπτύσσεται. Αυτό σημαίνει ότι επιλέγονται οι αρχιτεκτονικές λογισμικού, οι συγκεκριμένες διεπαφές, και οι αλγόριθμοι. Οι επιλογές επικυρώνονται ότι ικανοποιούν τους προσδιορισμούς απαιτήσεων. Τα προϊόντα του σχεδιασμού των δραστηριοτήτων πρέπει επίσης να είναι ελεγμένα σχετικά με συγκεκριμένους περιορισμούς που προσδιορίζονται στην αρχή της φάσης σχεδιασμού.

Τέλος, κατά τη διάρκεια της διαδικασίας υλοποίησης, παράγεται ο πηγαίος κώδικας. Ο πηγαίος κώδικας πρέπει να επιβεβαιώνεται ότι ικανοποιεί τους προσδιορισμούς απαιτήσεων, και πρέπει να ελεγχθεί σχετικά με τους περιορισμούς του σχεδιασμού. Τα δεδομένα δοκιμών παράγονται κατά τη διάρκεια της εκτέλεσης του προκύπτοντος πηγαίου κώδικα. Το σύστημα λειτουργίας είναι τεκμηριωμένο (π.χ., αποτελέσματα των δοκιμών, κόστος λογισμικού, μετρήσεις της ποιότητας λογισμικού, εγχειρίδια χρηστών). Στον έλεγχο του κώδικα γίνεται επαλήθευση ότι αναπτύχθηκε σωστά το λογισμικό με βάση αυτά που έχουν καθοριστεί στην ανάλυση και το σχεδιασμό. Επιπλέον γίνεται επικύρωση ότι όντως αναπτύχθηκε το προϊόν σύμφωνα με τις απαιτήσεις του χρήστη.

Στα επόμενα κεφάλαια, θα γίνει εκτενής αναφορά στις παραπάνω φάσεις ανάπτυξης του Πληροφοριακού Συστήματος της Οφθαλμολογικής Κλινικής.

2.2 Καθορισμός Απαιτήσεων - Παραδοχές

Ως πρώτο στάδιο ανάπτυξης ενός λογισμικού, ορίζεται ο καθορισμός απαιτήσεων του, όπως αναφέρθηκε παραπάνω. Συνεπώς, για το Πληροφοριακό Σύστημα της Οφθαλμολογικής Κλινικής, ορίζονται οι παρακάτω απαιτήσεις – παραδοχές:

α) Οι χρήστες του συστήματος θα είναι το προσωπικό της κλινικής. Αυτό θα είναι είτε ιατρικό προσωπικό είτε το προσωπικό, το οποίο θα εκτελεί χρέη γραμματειακής υποστήριξης. Οι χρήστες θα συνδέονται στην εφαρμογή με την χρήση username και password.

β) Το ΠΣ θα περιλαμβάνει καρτέλα προσωπικού για κάθε εργαζόμενο. Η καρτέλα θα περιλαμβάνει τα δημογραφικά στοιχεία του εργαζομένου, στοιχεία επικοινωνίας του, όπως και

φωτογραφία του. Τέλος, το ιατρικό προσωπικό θα έχει την δυνατότητα δημιουργίας νέας καρτέλας εργαζομένου.

γ) Οι χρήστες του συστήματος θα μπορούν να διαχειρίζονται τον ιατρικό φάκελο ενός ασθενούς. Συνεπώς, θα μπορούν να δημιουργούν, να επεξεργάζονται και να διαγράφουν έναν ιατρικό φάκελο. Ο ιατρικός φάκελος θα περιλαμβάνει όλα τα δημογραφικά δεδομένα του ασθενούς, όπως επίσης τις αλλεργίες του, τις φαρμακευτικές αγωγές του και τις ασθένειες του. Τέλος, στον ιατρικό φάκελο ενός ασθενούς θα υπάρχει η δυνατότητα προσθήκης φωτογραφίας του ασθενούς.

δ) Οι εξετάσεις που προσφέρει ως υπηρεσία η κλινική είναι η κλινική εξέταση και ο διαθλαστικός έλεγχος.

ε) Δυνατότητα εμφάνισης, επεξεργασίας και διαγραφής μίας εξέτασης έχει μόνο το ιατρικό προσωπικό. Αντιθέτως, η γραμματεία δεν έχει πρόσβαση στις παραπάνω δυνατότητες.

στ) Από μία εξέταση, ανεξαρτήτου τύπου, μπορεί να προκύψουν από καμία έως πολλές διαγνώσεις.

ζ) Μία διάγνωση χαρακτηρίζεται από τον κωδικό ICD10, σύμφωνα με τα Διεθνή Πρότυπα.

η) Από μία διάγνωση, μπορεί να προκύψουν από καμία έως πολλές θεραπείες.

θ) Μία θεραπεία χαρακτηρίζεται από τον κωδικό φαρμακευτικού σκευάσματος.

ι) Το ΠΣ θα αποθηκεύει όλους τους κωδικούς ICD10, που σχετίζονται με οφθαλμολογικές παθήσεις.

ια) Το ΠΣ θα αποθηκεύει τους πιο συνηθισμένους κωδικούς φαρμακευτικών σκευασμάτων, που διατίθενται στις περιπτώσεις οφθαλμολογικών παθήσεων.

ιβ) Το ΠΣ θα παρέχει λειτουργίες προγραμματισμού ραντεβού. Οι χρήστες του συστήματος θα μπορούν να δημιουργήσουν νέο ραντεβού, να επεξεργαστούν υπάρχον ραντεβού ή και να το διαγράψουν.

ιγ) Κάθε ραντεβού θα χαρακτηρίζεται από τον ασθενή, τον επιβλέπων ιατρό καθώς και την προγραμματισμένη ημερομηνία και ώρα της επίσκεψης.

ιδ) Το σύστημα θα υλοποιεί λειτουργίες back up/restore της Βάσης Δεδομένων του. Θα εμφανίζει τον αριθμό των ημερών που έχουν παρέλθει από την ημέρα που έγινε τελευταία φορά back up.

ιε) Το Σύστημα θα παρέχει στατιστικά δεδομένα για την λειτουργία της κλινικής. Συγκεκριμένα, θα παρέχει στατιστικά στοιχεία για τις εξετάσεις που έχουν πραγματοποιηθεί ανά έτος και ανα μήνα. Επίσης θα παρέχονται στατιστικά δεδομένα για τα ραντεβού-επισκέψεις που έχει δεχτεί η κλινική.

ιστ) Το ΠΣ θα μπορεί να συνδέεται με εξωτερική Βάση Δεδομένων και να αντλεί προσωπικές πληροφορίες των ασθενών, τις οποίες θα εκμεταλλεύεται στην συνέχεια.

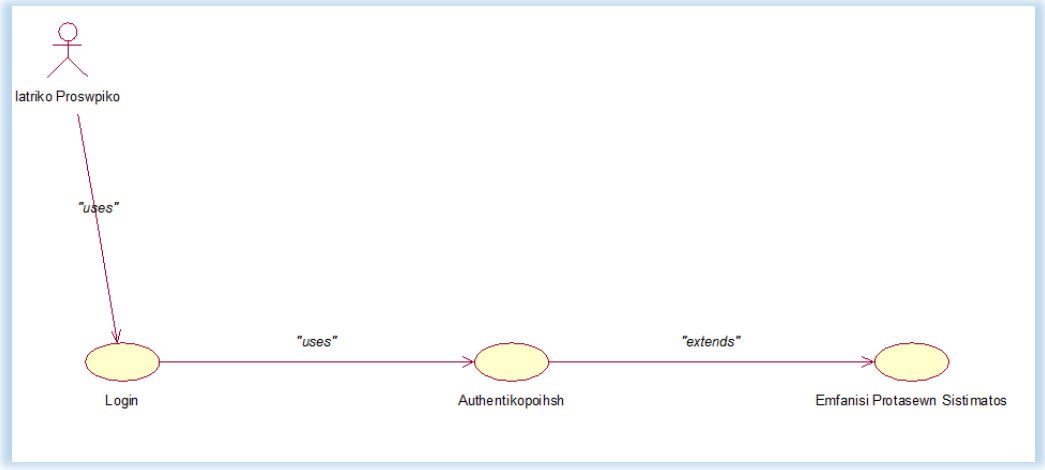
2.3 Σχεδιασμός με χρήση UML

Η Unified Modeling Language (UML) είναι μια γραφική γλώσσα αναπαράστασης αντικειμενοστραφών μοντέλων συστημάτων. Η UML αναπτύχθηκε το 1997 από τους Booch, Rambaugh και Jacobson με σκοπό να λύσει το πρόβλημα των πολλών διαφορετικών αναπαραστάσεων που υπήρχαν μέχρι τότε για τα μοντέλα αντικειμένων. Έκτοτε η UML έχει πράγματι αποτελέσει ένα διεθνές πρότυπο αντικειμενοστραφούς γλώσσας μοντελοποίησης.

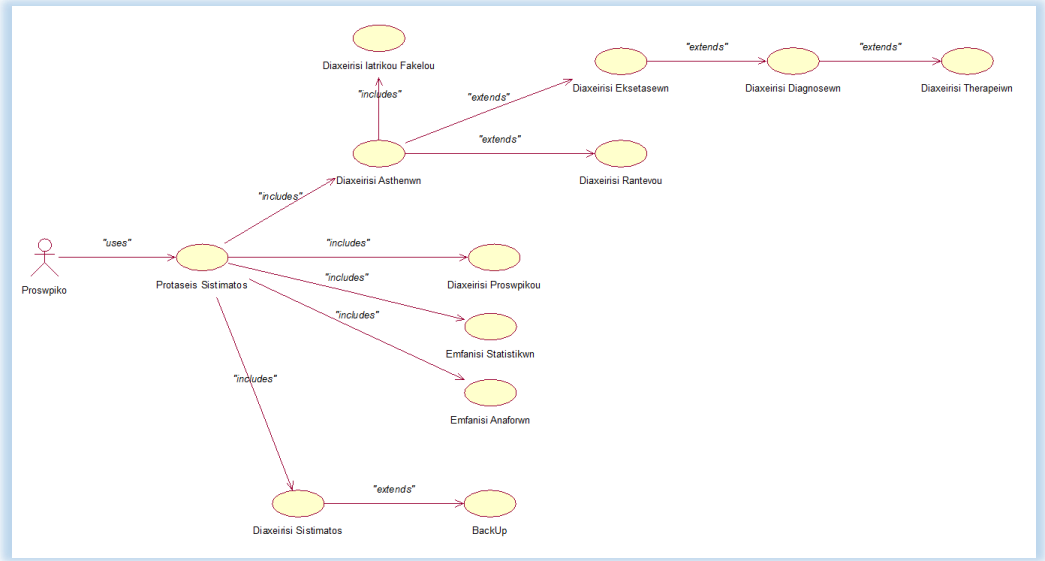
Η UML είναι γλώσσα μοντελοποίησης και όχι μεθοδολογία. Η μεθοδολογία αποτελείται από μια γλώσσα μοντελοποίησης και μια διαδικασία. Η UML ορίζει μόνο τη γλώσσα μοντελοποίησης και

όχι τη διαδικασία. Η γλώσσα μοντελοποίησης βοηθάει στην περιγραφή του σχεδιασμού. Η διαδικασία ορίζει τον τρόπο δημιουργίας του σχεδιασμού.

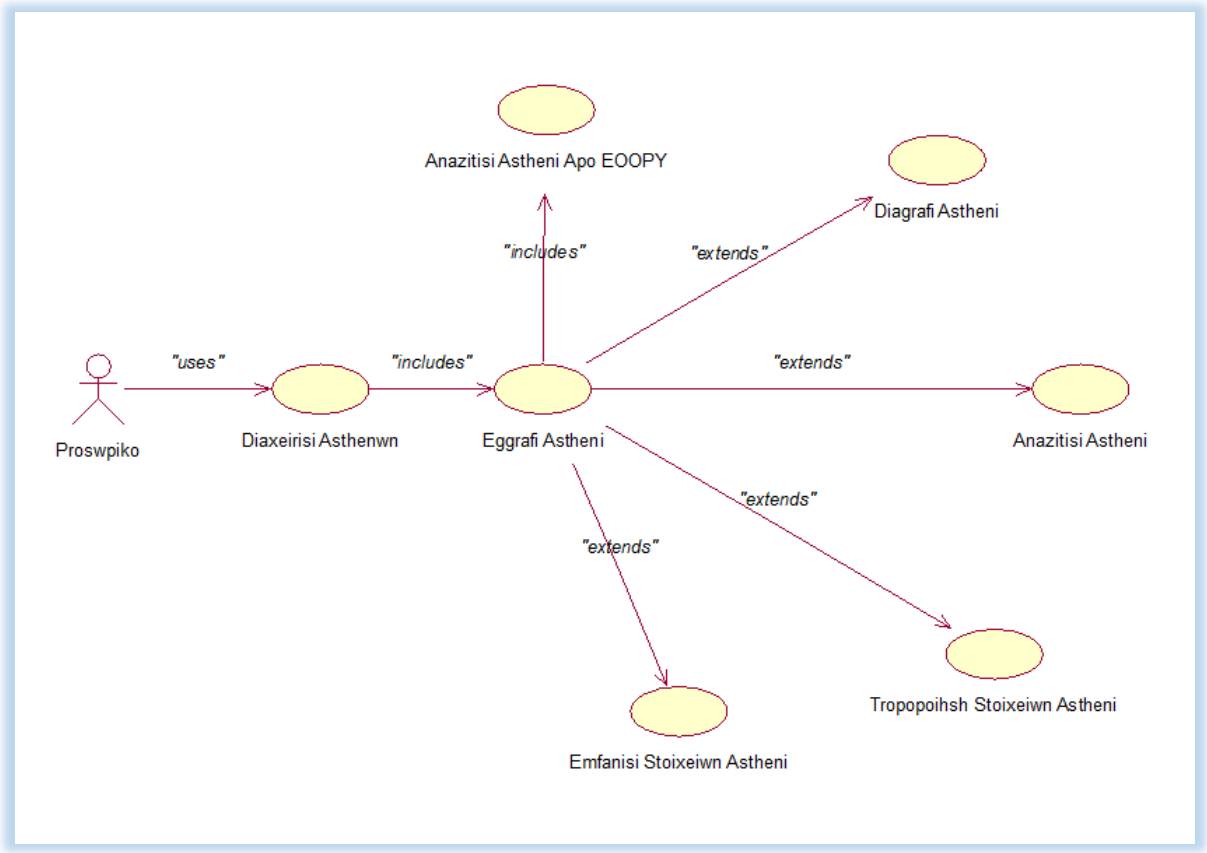
Παρακάτω παρατίθενται τα διαγράμματα περιπτώσεων χρήσης (Use Case Diagrams), τα οποία είναι μεταξύ των 9 διαφορετικών ειδών διαγραμμάτων που ορίζει η UML, και που αναπαριστούν τις λειτουργίες του ΠΣ από την οπτική γωνία του χρήστη. Ενημερωτικά, τα διαγράμματα περιπτώσεων χρήσης αποσκοπούν στην συγκεντρωτική εμφάνιση των απαιτήσεων ενός συστήματος, προσδίδουν μία εποπτική εικόνα του συστήματος, αναγνωρίζουν τους εσωτερικούς και εξωτερικούς παράγοντες που επηρεάζουν ένα σύστημα και τέλος, εμφανίζουν την αλληλεπίδραση με τους χρήστες του συστήματος (actors). Συνεπώς, για την περίπτωση του εξεταζόμενου ΠΣ της παρούσας ΔΕ έχουμε τα εξής διαγράμματα περιπτώσεων χρήσης:



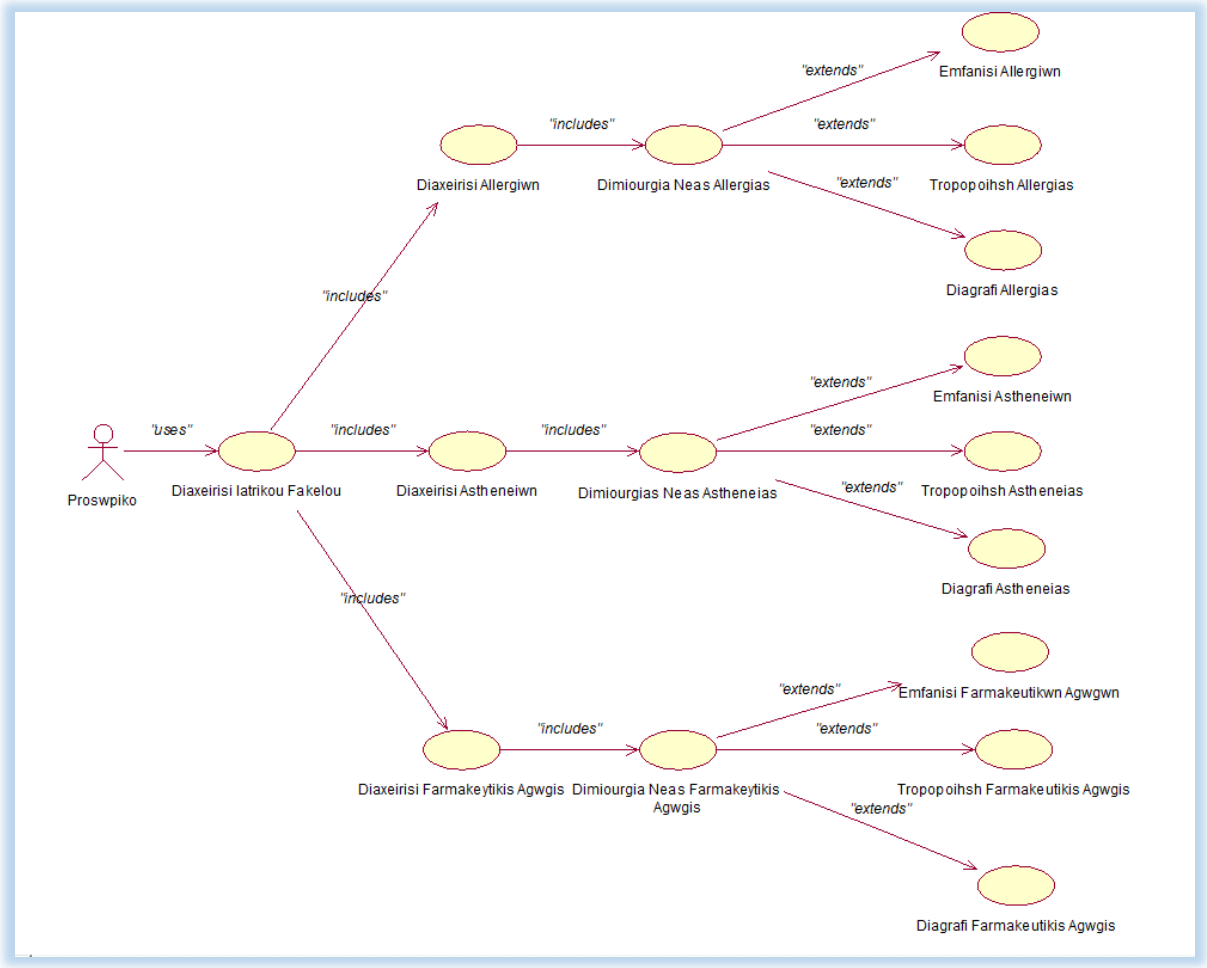
Εικόνα 1. Use Case: Login



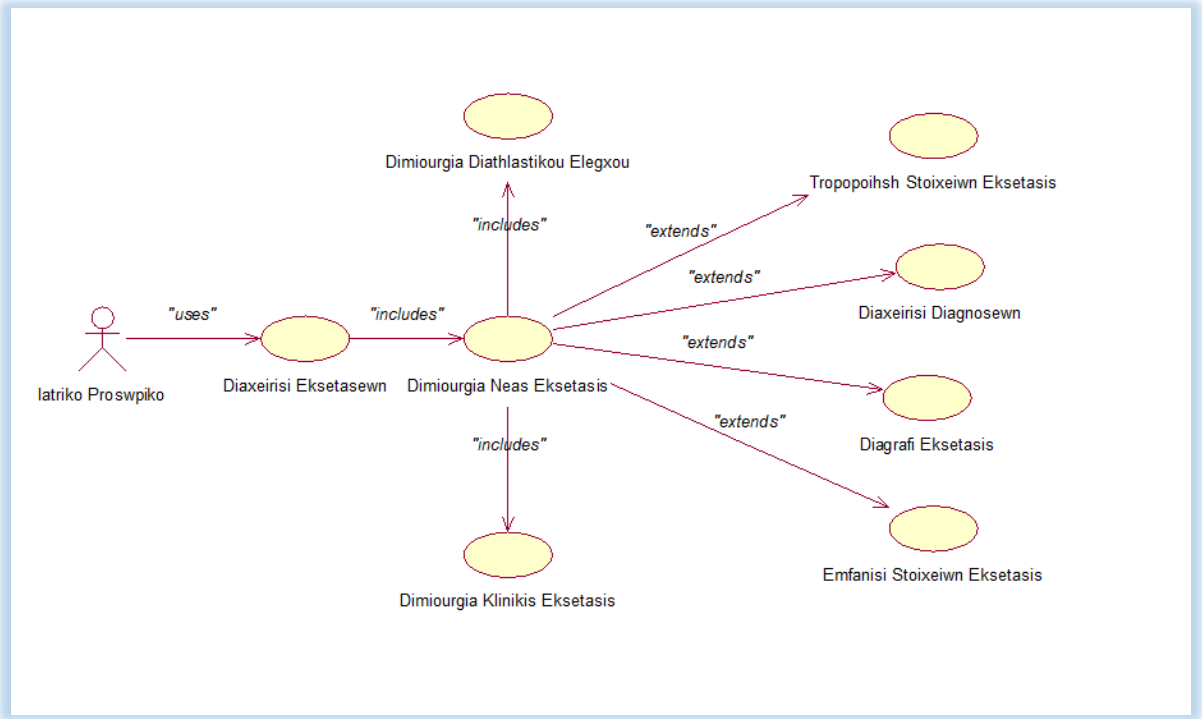
Εικόνα 2. Use Case: Προτάσεις Συστήματος



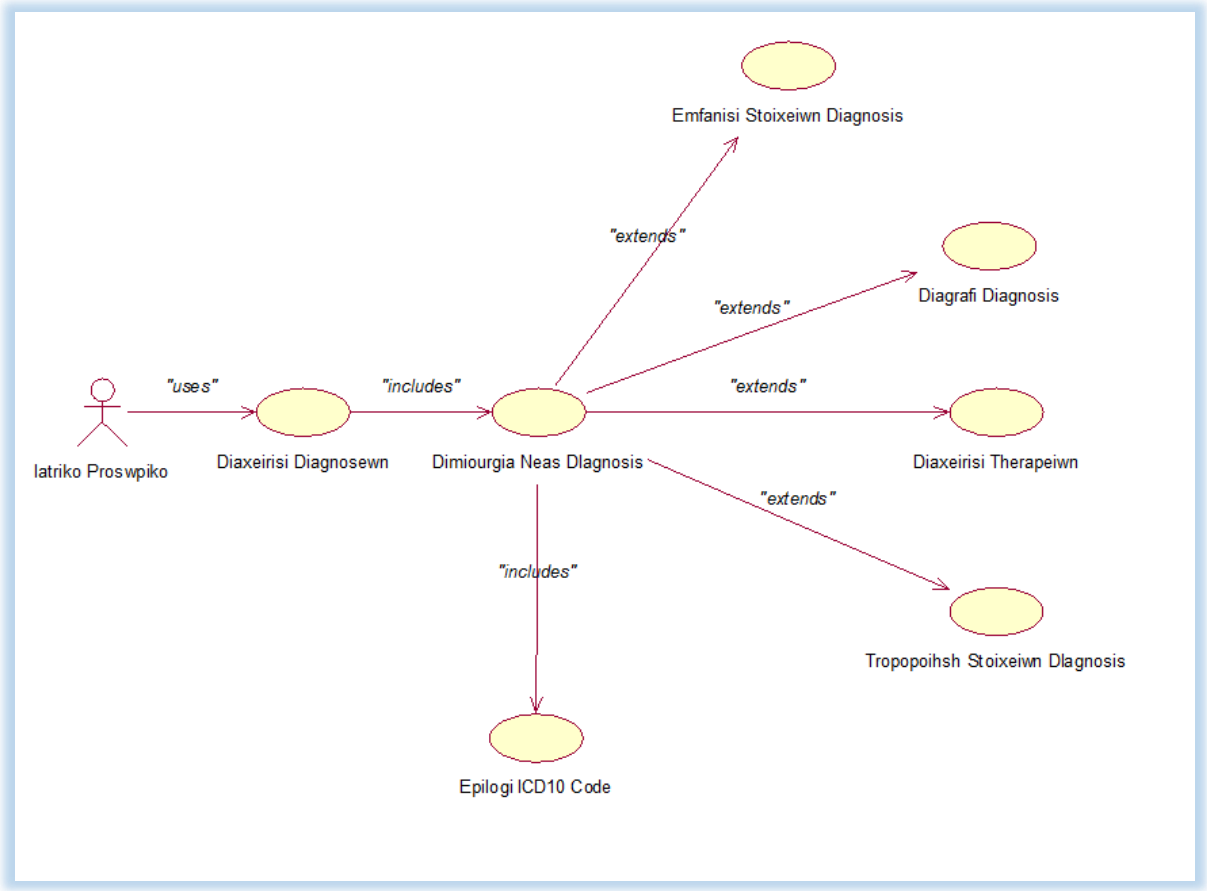
Εικόνα 3. Use Case: Διαχείριση Ασθενών



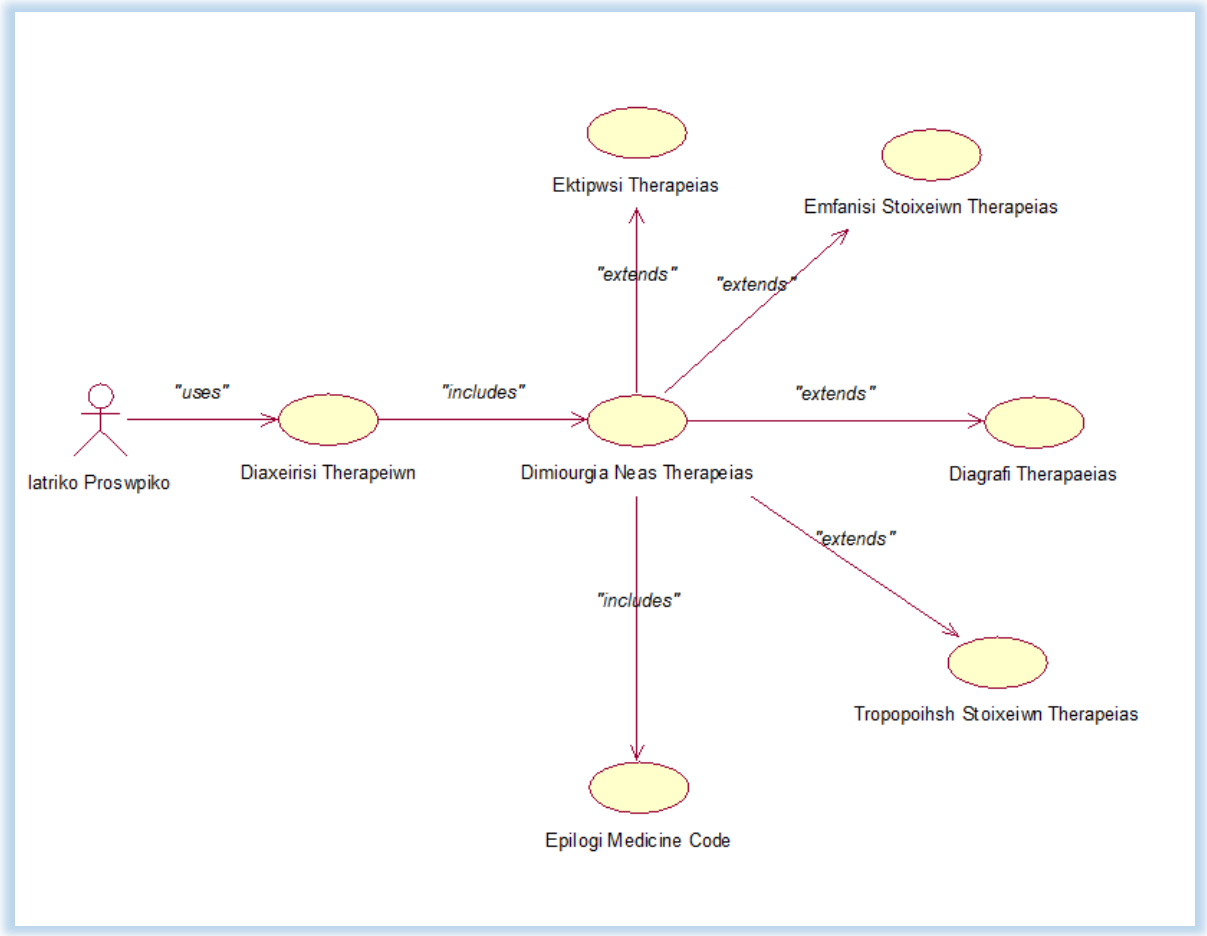
Εικόνα 4. Use Case: Διαχείριση Ιατρικού Φακέλου



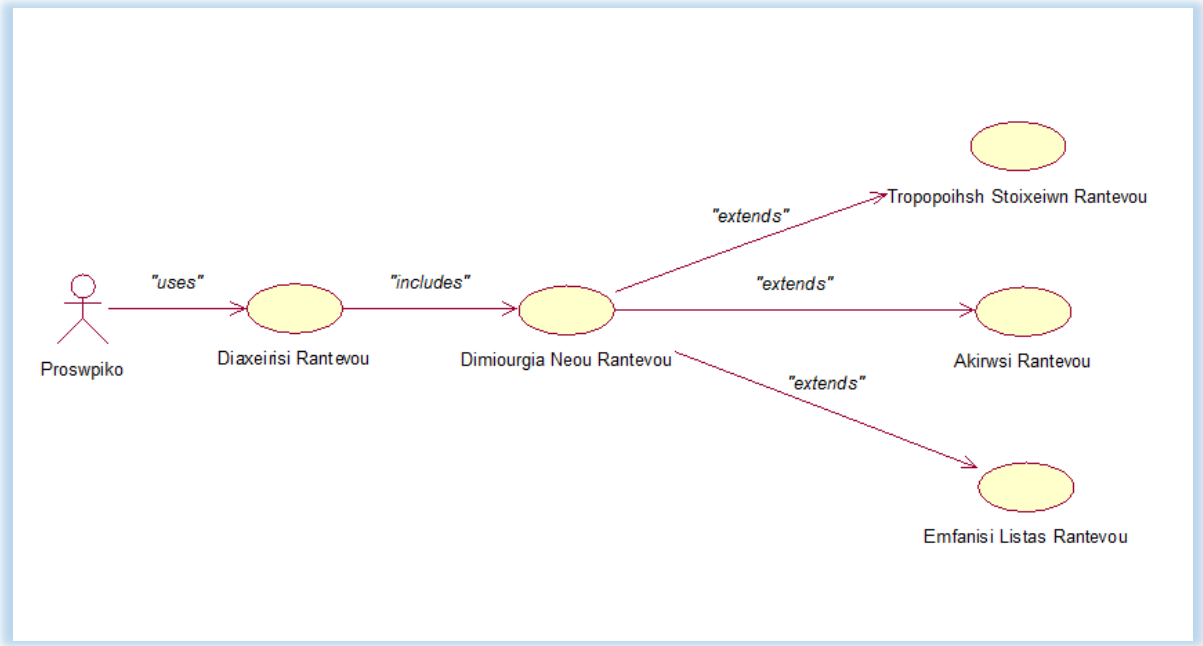
Εικόνα 5. Use Case: Διαχείριση Εξετάσεων



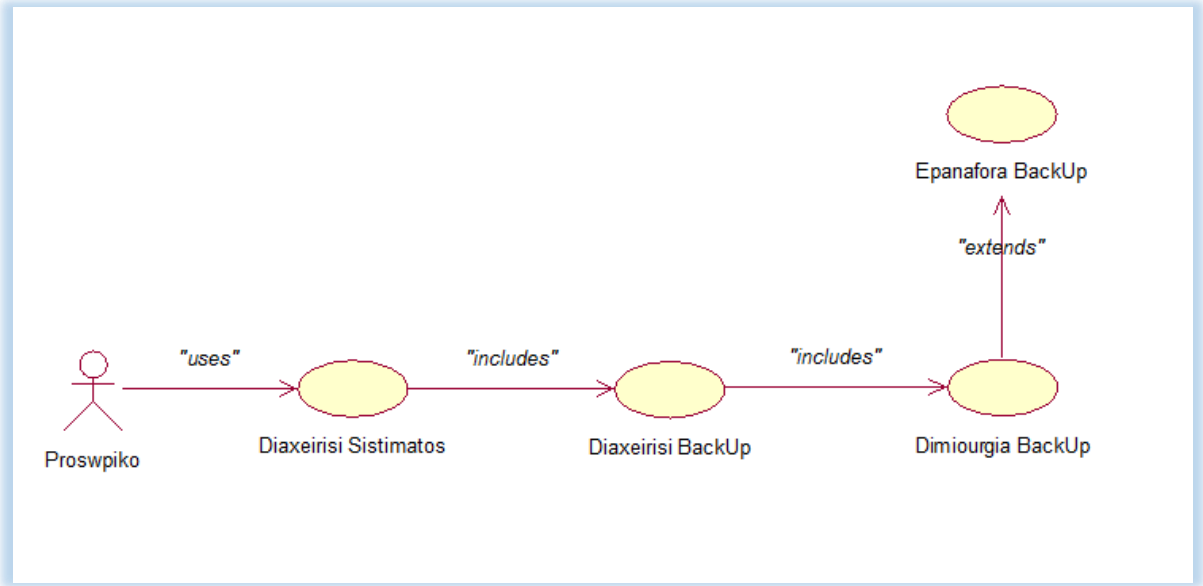
Εικόνα 6. Use Case: Διαχείριση Διαγνώσεων



Εικόνα 7. Use Case: Διαχείριση Θεραπειών



Εικόνα 8. Use Case: Διαχείριση Ραντεβού



Εικόνα 9. Use Case: Διαχείριση Συστήματος

Με τα παραπάνω διαγράμματα απεικονίζεται ένα δυναμικό στιγμιότυπο του συστήματος για κάθε περίπτωση ξεχωριστά. Συνεπώς, έχουμε μία εποπτική εικόνα για τις λειτουργίες του ΠΣ. Στις επόμενες ενότητες θα επιχειρηθεί μία λεπτομερής ανάλυση στην υλοποίηση του υπό εξέταση ΠΣ,

στην διάρκεια της οποίας θα γίνει χρήση και άλλων διαγραμμάτων που προσφέρει η UML, όπου αυτό κρίνεται σκόπιμο.

2.4 Υλοποίηση ΠΣ

2.4.1 Το Visual Studio

Όπως αναφέρθηκε παραπάνω, μετά τον καθορισμό των απαιτήσεων και τον σχεδιασμό του ΠΣ ακολουθεί η υλοποίηση αυτού. Σε αυτή την ενότητα θα γίνει μία αναφορά στις γλώσσες προγραμματισμού που χρησιμοποιήθηκαν για την υλοποίηση του ΠΣ καθώς και στην δομή του κώδικα που εφαρμόστηκε.

Το περιβάλλον δημιουργίας του ΠΣ είναι το Visual Studio 2015, update 3. Η εν λόγω σουίτα αποτελεί ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού, το οποίο δημιουργήθηκε από την Microsoft. Χρησιμοποιείται για την ανάπτυξη προγραμμάτων, ηλεκτρονικών υπολογιστών για τα Microsoft Windows, καθώς και ιστοσελίδων, διαδικτυακών εφαρμογών, υπηρεσιών Web, εφαρμογών κινητών συσκευών κ.α. Το Visual Studio χρησιμοποιεί πλατφόρμες ανάπτυξης λογισμικού της Microsoft, όπως τα Windows API, τα Windows Forms, τα Windows Presentation Foundation, το Windows Store κ.α. Περιλαμβάνει πολλές χρήσιμες λειτουργίες, όπως code editor με υποστήριξη IntelliSense, forms designer για την δημιουργία εφαρμογών GUI, web designer, database schema designer κ.α. Επίσης, δέχεται plugins, τα οποία βελτιώνουν την λειτουργικότητα του σχεδόν σε κάθε επίπεδο και στάδιο παραγωγής κώδικα. Τέλος, το Visual Studio υποστηρίζει διαφορετικές γλώσσες προγραμματισμού τόσο σε επίπεδο code editor όσο και σε επίπεδο debugging. Οι built-in γλώσσες που υποστηρίζονται περιλαμβάνουν την C, C++,VB.NET, C#, F# ενώ η υποστήριξη για τις υπόλοιπες γλώσσες προγραμματισμού, όπως Python, Ruby κ.α., παρέχεται μετά την επιμέρους εγκατάσταση languages services.

Το Visual Studio υιοθετεί μία δικιά του φιλοσοφία και ορολογία σε ότι αφορά την λογική τμηματοποίηση του κώδικα που παράγει. Έτσι για παράδειγμα, χρησιμοποιούνται οι όροι Solution και Project, οι οποίοι είναι τα πιο βασικά δομικά στοιχεία του κώδικα που αναπτύσσεται.

Ένα Project, περιλαμβάνει όλα τα αρχεία πηγαίου κώδικα, εικονίδια, εικόνες, αρχεία δεδομένων και οτιδήποτε άλλο θα γίνεται compile σε ένα εκτελέσιμο πρόγραμμα. Το Project περιλαμβάνει επίσης τις ρυθμίσεις του compiler και των υπολοίπων configuration files, που απαιτούνται για την επικοινωνία του Project με διάφορες υπηρεσίες (services).

Ένα Solution αποτελεί την δομή για την οργάνωση των Project. Ένα Solution διατηρεί τις πληροφορίας κατάστασης (state information) για το σύνολο των Project, που περιλαμβάνει σε αρχεία τύπου .sln.

Στην συνέχεια, ακολουθεί μία αναφορά της δομής του κώδικα που αναπτύχθηκε με σκοπό την υλοποίηση των λειτουργιών του ΠΣ της παρούσας ΔΕ.

2.4.2 Δομή Κώδικα

Ο κώδικας που δημιουργήθηκε για την δημιουργία του ΠΣ και την υλοποίηση των επιμέρους λειτουργιών που αναφέρθηκαν στις προηγούμενες ενότητες, φιλοξενείται σε 3 Solutions, ως εξής:

α) CrystalEye Solution

β) CrystalEyeWebService Solution

γ) CrystalEyeSite Solution

Το CrystalEye Solution, περιλαμβάνει τον κυρίως κώδικα της εφαρμογής που αναπτύχθηκε για την υλοποίηση των λειτουργιών που αναφέρθηκαν στις προηγούμενες ενότητες. Η γλώσσα προγραμματισμού, που χρησιμοποιήθηκε είναι η WPF (Windows Presentation Foundation) και αποτελείται από τα εξής 2 Projects:

α) CrystalEye Project, το οποίο περιλαμβάνει τον κώδικα, ο οποίος υλοποιεί τις λειτουργίες και απαιτήσεις του ΠΣ και το

β) StoreDatabase Project, το οποίο περιλαμβάνει τον κώδικα, ο οποίος σχετίζεται με την σύνδεση του κυρίως προγράμματος με την Βάση Δεδομένων.

Το CrystalEyeWebService Solution περιλαμβάνει τον κώδικα που σχετίζεται με την δημιουργία ενός Web Service σε γλώσσα WCF (Windows Communication Foundation). Το Web Service αφορά στην σύνδεση της κύριας εφαρμογής που υλοποιήθηκε στο CrystalEye Project με μία δεύτερη Βάση Δεδομένων, η οποία υποθετικά ανήκει στον ΕΟΟΠΥ. Η σύνδεση αφορά τον έλεγχο του ΑΜΚΑ ενός ασθενή με την Βάση Δεδομένων του ΕΟΟΠΥ, προκειμένου να διασταυρωθούν τα στοιχεία (προσωπικά δεδομένα) του ασθενή. Το υπόψη Solution περιλαμβάνει τα εξής 4 Projects:

α) StoreDatabase Project, το οποίο αφορά στην σύνδεση με την Βάση Δεδομένων ΕΟΟΠΥ,

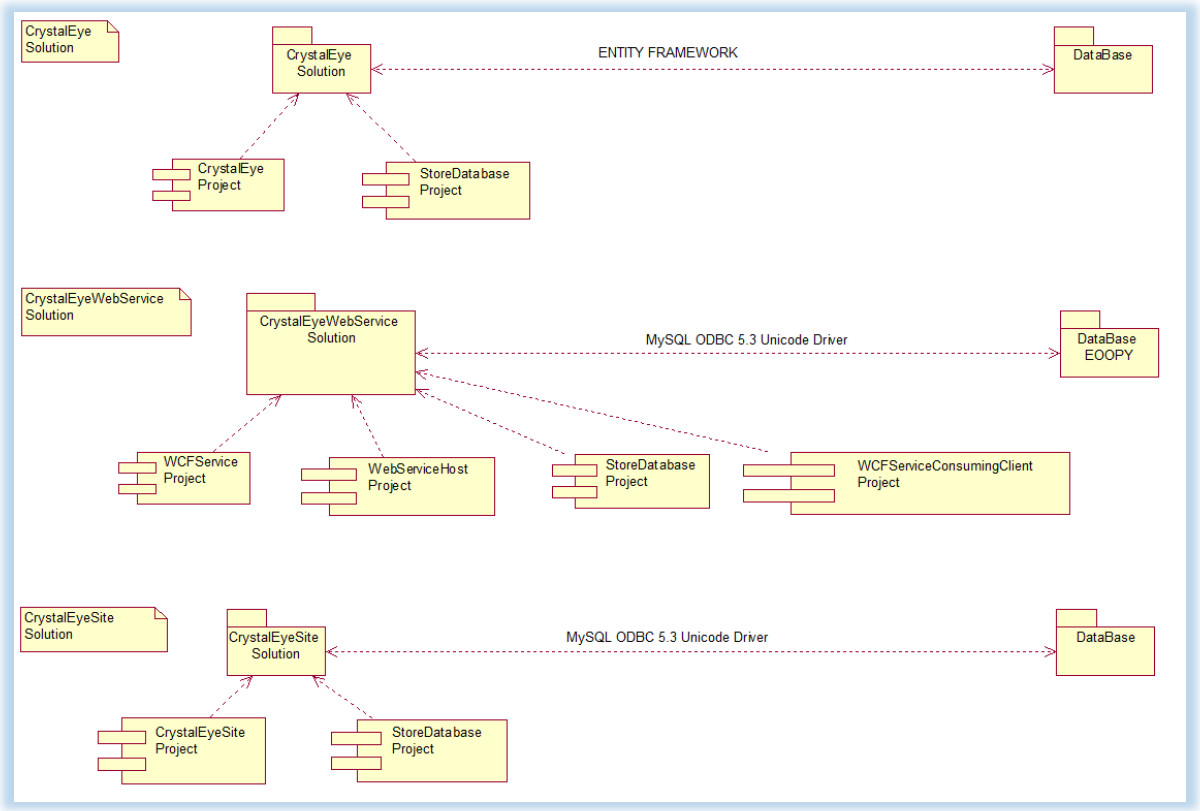
β) WCFService Project, το οποίο περιλαμβάνει τον κύριο κώδικα υλοποίησης της υπηρεσίας, όπως τα contracts κλπ,

γ) WebServiceHost Project, το οποίο περιλαμβάνει τον κώδικα που υλοποιεί το Hosting της Web Service, και τέλος,

δ) WCFServiceConsumingClient Project, το οποίο αφορά στην λειτουργία κατανάλωσης του Web Service από έναν υποθετικό client και ο σκοπός ύπαρξης του είναι καθαρά δοκιμαστικός, προκειμένου να είμαστε σίγουροι ότι η διαδικασία deliver-consume του Web Service πραγματοποιείται με επιτυχία.

Το CrystalEyeSite Solution περιλαμβάνει τον κώδικα υλοποίησης ενός Ιστότοπου, τον οποίο επισκέπτονται οι ασθενείς της οφθαλμολογικής κλινικής και τους δίνεται η δυνατότητα να επεξεργαστούν το προφίλ τους, να κλείσουν ραντεβού με την κλινική κ.α. Η γλώσσα προγραμματισμού που αξιοποιήθηκε είναι η ASP.Net και το Solution αποτελείται από 2 Projects, το CrystalEyeSite Project και το StoreDatabase Project. Το πρώτο αφορά τον κώδικα που υλοποιεί τις λειτουργίες του Ιστότοπου και το δεύτερο αφορά την σύνδεση του Ιστότοπου με την Βάση Δεδομένων του CrystalEye Project. Ωστόσο, δεν θα επεκταθούμε σε λεπτομέρεια στις λειτουργίες και στην ανάπτυξη κώδικα του υπόψη Solution, καθώς η δημιουργία του αποτέλεσε εργασία στο μάθημα της Ιατρικής Πληροφορικής και η λεπτομερής ανάλυση του έχει πραγματοποιηθεί εκεί.

Παρακάτω, φαίνεται μία σχηματική αναπαράσταση των 3 Solutions, με την βοήθεια των διαγραμμάτων Component, που παρέχει η γλώσσα UML:

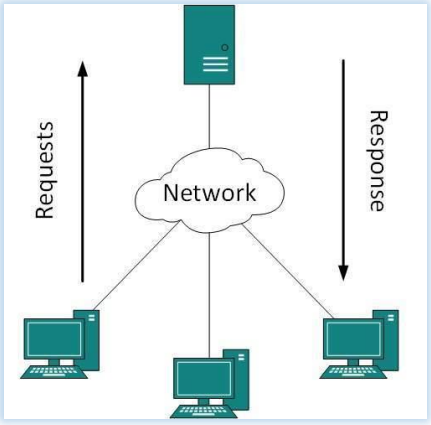


Εικόνα 10. Component View: Δομή Κώδικα ΠΣ

Στις επόμενες ενότητες θα γίνει αναλυτική εξέταση της δομής, των τεχνολογιών και των μεθοδολογιών του κώδικα που χρησιμοποιήθηκαν, για κάθε ένα Solution από τα προαναφερθέντα, πλην αυτό του CrystalEyeSite, για τον λόγο που αναφέραμε παραπάνω. Αρχικά όμως θα επιχειρηθεί μία λεπτομερής ανάλυση της Βάσης Δεδομένων του ΠΣ, καθώς η σωστή δημιουργία της Βάσης Δεδομένων μπορεί να κρίνει την αξιοπιστία της εφαρμογής, ασχέτως της ποιότητας του κώδικα που αφορά το GUI. Για αυτόν τον λόγο δίνεται ιδιαίτερη βαρύτητα στην τήρηση των βασικών κανόνων ανάπτυξης μίας ΒΔ, όπως θα δούμε και στην συνέχεια.

2.4.3 Αρχιτεκτονική Υλοποίησης - GUI

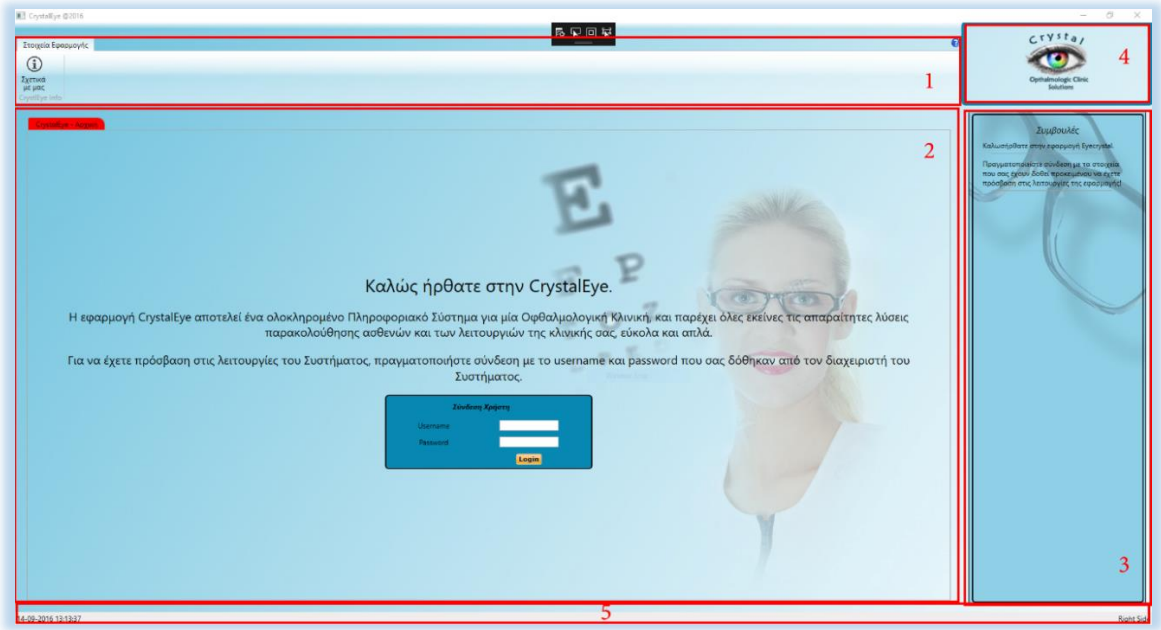
Η αρχιτεκτονική που εφαρμόζεται στο deployment της εφαρμογής είναι αυτή του client-server, όπως φαίνεται στο παρακάτω σχήμα:



Εικόνα 11. Μοντέλο client-server

Συγκεκριμένα, υποτίθεται ότι η ΒΔ είναι εγκατεστημένη σε κάποιο απομακρυσμένο server, και υπάρχει ένα σύνολο από clients, οι οποίοι πραγματοποιούν ερωτήματα στον server. Ωστόσο, για λόγους διευκόλυνσης κατά την υλοποίηση του ΠΣ της παρούσας ΔΕ, το development workstation που έχει εγκατεστημένη την εφαρμογή, έχει τόσο τον ρόλο client όσο και τον ρόλο του server. Εάν ήταν επιθυμητή η εγκατάσταση του συστήματος προς παραγωγή, θα υπήρχε η απαίτηση να τροποποιηθούν τα connection string και κάποιες ιδιότητες των επιμέρους Project.

Η φιλοσοφία στην οποία στηρίζεται η εμφάνιση του UI της εφαρμογής και που κατ' επέκταση επηρεάζει και τον παραγόμενο κώδικα είναι μονοπαραθυρική. Συγκεκριμένα, υπάρχει ένα μοναδικό παράθυρο, το οποίο με την βοήθεια των Usercontrols, που παρέχει το WPF Framework, μπορεί να εναλλάσσει συνεχώς τις φόρμες και, γενικότερα, τα δεδομένα που εμφανίζει. Ενδεικτικά, στην παρακάτω εικόνα έχουμε το αρχικό παράθυρο της εφαρμογής:



Εικόνα 12. Αρχικό Παράθυρο του Πληροφοριακού Συστήματος

Στην παραπάνω εικόνα, φαίνεται το αρχικό παράθυρο της εφαρμογής, το οποίο έχει υποστεί κατάτμηση, σύμφωνα με το είδος των πληροφοριών που φιλοξενεί το κάθε τμήμα. Συγκεκριμένα:

α) Τμήμα 1: Φιλοξενεί το μενού επιλογών της εφαρμογής. Υιοθετεί το νέο στυλ μενού της Microsoft, με την χρήση Ribbon bar. Περιλαμβάνει συγκεκριμένη κατηγοριοποίηση στην απεικόνιση των επιμέρους στοιχείων του μενού, όπως Στοιχεία Εφαρμογής, Ενέργειες, Ασθενής, Χρήστης κλπ.

β) Τμήμα 2: Αποτελεί placeholder για τα Usercontrols. Ουσιαστικά, φιλοξενείται ο μεγαλύτερος όγκος πληροφορίας, όπως φόρμες κλπ. Εφαρμόζεται μία λογική tabs, όπου ο χρήστης ανάλογα με τις επιλογές που κάνει στην εφαρμογή, δημιουργεί καινούρια tabs ή καταργεί τα ήδη υπάρχοντα.

γ) Τμήμα 3: Έχει τον ρόλο Sidebar. Ουσιαστικά, εμφανίζει βοηθητικές πληροφορίες για το tab του τμήματος 2, που έχει το focus. Σε αυτές τις πληροφορίες περιλαμβάνονται τα μηνύματα επικοινωνίας με τον χρήστη, επιπλέον ενέργειες, όπου επιτρέπεται η εκτέλεση τους στο τρέχον tab, μηνύματα σφάλματος καθώς και βοηθητικές πληροφορίες για την είσοδο τιμών του χρήστη στις φόρμες.

δ) Τμήμα 4: Εμφανίζει τον συνδεδεμένο χρήστη (ονοματεπώνυμο) καθώς και ένα κουμπί, το οποίο εκτελεί την αποσύνδεση του. Επίσης εμφανίζει το logo της κλινικής.

ε) Τμήμα 5: Είναι η γνωστή μπάρα κατάστασης. Χωρίζεται σε 3 ίσα τμήματα. Το αριστερό εμφανίζει την ώρα του συστήματος, το μεσαίο εμφανίζει μηνύματα που προέρχονται από εργασίες που εκτελεί η εφαρμογή στο background (threads), και τέλος το τρίτο, το οποίο εμφανίζει τα στοιχεία του συνδεδεμένου χρήστη.

Ένα στοιχείο που αξίζει να σημειωθεί είναι η άρρηκτη σύνδεση των τμημάτων 2 και 3. Συγκεκριμένα, όποτε αλλάζει το ενεργό tab, αυτομάτως αλλάζει και το περιεχόμενο του sidebar, ακόμη κι αν το σύστημα περιμένει την είσοδο από τον χρήστη σε μήνυμα τύπου (OK,Cancel). Σε επόμενα κεφάλαια, θα γίνει ανάλυση του κώδικα που πετυχαίνει την υπόψη δυνατότητα, η οποία παρουσιάζει ενδιαφέρον.

Παρόλο που τονίστηκε ότι η εφαρμογή είναι μονοπαραθυρική, γίνεται επιτρεπτό σε εξαιρετικές περιπτώσεις, να εμφανίζονται επιπλέον παράθυρα. Η περίπτωση στην οποία κάτι τέτοιο προβλέπεται από την σχεδίαση του ΠΣ είναι όταν ο χρήστης επιθυμεί να επεξεργαστεί ένα αρχείο εικόνας, το οποίο αντιστοιχεί σε μία εξέταση ασθενή. Ο λόγος που συμβαίνει αυτό είναι διότι λαμβάνεται ως παραδοχή, η ανάγκη του χρήστη να έχει μεγιστοποιημένη την εικόνα, που περιέχει την ιατρική πληροφορία, σε όλο το μήκος και πλάτος της οθόνης, με σκοπό την καλύτερη ανάλυση της. Συνεπώς, αν και θα μπορούσε η εικόνα να εμφανίζεται ως ένα επιπλέον tab στο τμήμα 2, αυτό αποφεύγεται. Ένας δεύτερος λόγος, που μπορεί να εμφανιστεί επιπλέον παράθυρο στην εφαρμογή είναι στην εμφάνιση των αναφορών. Τέλος, μπορεί να εμφανιστούν επιπλέον παράθυρα, στις περιπτώσεις που εμφανίζονται σφάλματα, όπως κακή σύνδεση με την ΒΔ ή οτιδήποτε προκύπτει γενικότερα ως exception στα try-catch blocks που χρησιμοποιούνται στον κώδικα. Αυτό συμβαίνει για να τονιστεί η σημασία του σφάλματος που εμφανίστηκε, σε αντίθεση με ένα κοινό σφάλμα κακής εισαγωγής στοιχείων από τον χρήστη σε ένα πεδίο, το οποίο θα προκαλέσει μήνυμα σφάλματος στην sidebar.

3 Βάση Δεδομένων

3.1 Εισαγωγή

Είναι φρόνιμο, το σημείο εκκίνησης για την λεπτομερή ανάλυση δημιουργίας του ΠΣ, να είναι η Βάση Δεδομένων. Αυτό γίνεται καθώς η σωστή κατασκευή του σχήματος της ΒΔ ευθύνεται για την αποτελεσματικότητα και λειτουργικότητα της εφαρμογής σε πολύ μεγάλο βαθμό, για όλη την διάρκεια ζωής της.

Με τον όρο Βάση Δεδομένων, αναφερόμαστε σε οργανωμένες και διακριτές συλλογές σχετιζόμενων δεδομένων, τα οποία είναι ψηφιακά και ηλεκτρονικά αποθηκευμένα, και στα οποία είναι δυνατή η ανάκτηση, μέσω αναζήτησης κατ' απαίτηση. Συγκεκριμένα, μία Βάση Δεδομένων εκτός από την ικανότητα της να αποθηκεύει δεδομένα, παρέχει την δυνατότητα γρήγορης άντλησης και ανανέωσης των δεδομένων, μέσω του σχεδιασμού και του τρόπου ιεράρχησης αυτών. Με τον όρο Δεδομένα, εννοούμε γνωστά γεγονότα που μπορούν να καταγραφούν και έχουν κάποια υπονοούμενη σημασία.

Κατ' επέκταση, τα Συστήματα Διαχείρισης Βάσεων Δεδομένων αποτελούν συλλογές προγραμμάτων, τα οποία επιτρέπουν στους χρήστες να δημιουργούν και να συντηρούν μία Βάση Δεδομένων. Αυτή την στιγμή κυκλοφορούν αρκετά RDBMS στην αγορά. Μερικά από αυτά είναι:

α) MS SQL Server
β) SAP SQL Anywhere
γ) Oracle
δ) MySQL κ.α.

Όπως ήδη αναφέρθηκε, το RDBMS που χρησιμοποιήθηκε για την κατασκευή της ΒΔ είναι η MySQL version 6.3. Η επιλογή του συγκεκριμένου RDBMS έγινε διότι η community έκδοση είναι δωρεάν και μπορεί να διαχειριστεί απεριόριστο όγκο δεδομένων. Επίσης, αξιοποιήθηκαν οι γνώσεις που αποκτήθηκαν στο αντίστοιχο μάθημα Β Εξαμήνου «Βάσεις Δεδομένων», στο οποίο είχε γίνει αναλυτική παρουσίαση των πλεονεκτημάτων και του τρόπου λειτουργίας της MySQL.

3.2 Γενική Περιγραφή της ΒΔ

Στο πλαίσιο δημιουργίας του ΠΣ, απαιτείται αρχικά η δημιουργία μίας ΒΔ, η οποία να αποθηκεύει τα δεδομένα του συστήματος. Τα δεδομένα αυτά αφορούν προσωπικά δημογραφικά στοιχεία ασθενή, ιατρού και γραμματείας αλλά και απόρρητα στοιχεία, όπως εξετάσεις ασθενούς. Αποτελείται από πίνακες, views και stored procedures ενώ έχει δοθεί έμφαση στο validation των δεδομένων που εισάγονται.

Η πρόβλεψη σε ότι αφορά το validation των δεδομένων έχει υλοποιηθεί με την βοήθεια των triggers που παρέχει η MySQL και καθορίζει τόσο την αποδεκτή μορφή όσο και τις αποδεκτές τιμές, που θα έχουν τα δεδομένα. Αυτό προσδίδει ομοιογένεια στα δεδομένα, όπου απαιτείται, καθώς και εξασφαλίζει ότι η εισαγωγή δεδομένων από οποιαδήποτε πηγή θα γίνει με τους κανόνες που θέτει η ίδια η ΒΔ, αποκλείοντας την περίπτωση να εισαχθούν εσφαλμένα δεδομένα. Σε μία τέτοια δυσάρεστη περίπτωση, τα αποτελέσματα δεν θα ήταν προβλέψιμα για το ΠΣ καθώς στο GUI της εφαρμογής έχει σχεδιαστεί validation σε δεύτερο επίπεδο, όπως θα δούμε στην συνέχεια. Τέλος, τα triggers ενεργοποιούνται στις καταστάσεις before insert και before update.

Στην συνέχεια, δόθηκε ιδιαίτερη βαρύτητα στην τήρηση όλων των κανόνων Boyce-Codd, προκειμένου να εξασφαλιστεί η ακεραιότητα των δεδομένων, και ο αποκλεισμός της περίπτωσης εμφάνισης επαναλαμβανόμενης πληροφορίας. Συγκεκριμένα εφαρμόστηκαν τα εξής:

α) ο πρώτος κανόνας (1st Normal Form), σύμφωνα με τον οποίο ένα πεδίο πίνακα μπορεί να φιλοξενεί μόνο μία τιμή και όχι συνδυασμό τιμών,

β) ο δεύτερος κανόνας (2nd Normal Form), σύμφωνα με τον οποίο ένα πεδίο, το οποίο δεν είναι υποψήφιο για να έχει τον ρόλο κλειδιού, θα πρέπει να εξαρτάται από ένα υποσύνολο πεδίων στον πίνακα, τα οποία έχουν τον ρόλο υποψήφιου κλειδιού,

γ) ο τρίτος κανόνας (3rd Normal Form), σύμφωνα με τον οποίο ένα πεδίο, το οποίο δεν έχει τον ρόλο υποψήφιου κλειδιού, θα πρέπει να μην εξαρτάται από οποιοδήποτε άλλο πεδίο στον ίδιο πίνακα, το οποίο επίσης να μην έχει τον ρόλο υποψήφιου κλειδιού,

δ) Boyce-Codd Normal Form (BCNF), αποτελεί μία καλύτερη ουσιαστικά έκδοση του 3ου κανόνα και χειρίζεται ανωμαλίες, που δεν μπορεί να φέρει εις πέρας το 3ος κανόνας.

Η τήρηση των παραπάνω κανόνων αποτελεί μονόδρομο στην σχεδίαση οποιασδήποτε σχήματος ΒΔ και η αγνόηση τους συνήθως οδηγεί σε κατασκευή ιδιαίτερα προβληματικών εφαρμογών.

Τέλος, προκειμένου να προβλεφθούν θέματα concurrency στην ΒΔ, έχει εισαχθεί σε ορισμένους πίνακες ένα πεδίο τύπου Timestamp, το οποίο λειτουργεί συνδυαστικά με το Entity Framework, όπως θα αναλύσουμε στην συνέχεια.

3.3 Ανάλυση Απαιτήσεων - Παραδοχών της ΒΔ

Το σχήμα της υπό κατασκευής Βάσης Δεδομένων πρέπει να ανταποκρίνεται πλήρως στις παρακάτω παραδοχές:

α) Το προσωπικό της κλινικής θα μπορεί να εισάγει τα δημογραφικά στοιχεία ενός ασθενούς, στην πρώτη του επίσκεψη στην κλινική. Στην συνέχεια θα μπορεί να τροποποιήσει τα δεδομένα αυτά, σε οποιαδήποτε χρονική στιγμή. Τα δεδομένα αυτά αφορούν:

- ΑΜΚΑ
- ΑΜΑ
- Όνομα
- Επώνυμο
- Φύλο
- Ημ/νία Γέννησης
- Όνομα Πατρός
- Όνομα Μητρός
- Εθνικότητα
- Στοιχεία Διεύθυνσης
- Στοιχεία τρόπου Πληρωμής
- Στοιχεία Επικοινωνίας (email,sms,αλληλογραφία, τηλέφωνο κλπ)
- Φωτογραφία ασθενούς
- Εργασία Ασθενούς
- Στοιχεία σύνδεσης ασθενούς με τον Ιστότοπο της κλινικής

β) Το προσωπικό της κλινικής θα μπορεί να εισάγει και να τροποποιεί ιατρικά δεδομένα ενός ασθενούς, όπως αλλεργίες, ασθενείς και φαρμακευτικές αγωγές που λαμβάνει ο ασθενής την περίοδο της επίσκεψης του στην κλινική.

γ) Ο συνδεδεμένος χρήστης θα είναι είτε ιατρός είτε θα ανήκει στην γραμματειακή υποστήριξη της κλινικής. Και στις δύο περιπτώσεις θα μπορεί να συνδεθεί στην εφαρμογή με χρήση username και password και θα είναι σε θέση να επεξεργάζεται γενικά στοιχεία του λογαριασμού τους, μερικά από τα οποία είναι:

- ΑΜΚΑ
- ΑΜΑ

- Όνομα
- Επώνυμο
- Φύλο
- Ημ/νία Γέννησης
- Όνομα Πατρός
- Όνομα Μητρός
- Εθνικότητα
- Στοιχεία Διεύθυνσης
- Στοιχεία τρόπου Πληρωμής
- Στοιχεία Επικοινωνίας (email,sms,αλληλογραφία, τηλέφωνο κλπ)
- Στοιχεία σύνδεσης του προσωπικού με την εφαρμογή

δ) Το ιατρικό προσωπικό της κλινικής θα μπορεί να εισάγει καινούριο προσωπικό στην ΒΔ.

ε) Η κλινική προσφέρει την εκτέλεση των εξής δύο τύπων εξετάσεων στους ασθενείς της: διαθλαστικό έλεγχο και κλινική εξέταση. Και στις δύο αυτές εξετάσεις αποθηκεύονται δεδομένα, τα οποία προέρχονται είτε από μετρήσεις ειδικών ιατρικών μηχανημάτων είτε από την εξέταση που εκτελεί ο ιατρός. Υπάρχει περιορισμός στις δυνατές τιμές που μπορεί να έχει κάθε μέτρηση. Μία εξέταση εκτελείται από έναν ιατρό αποκλειστικά.

στ) Ειδικά για την περίπτωση της κλινικής εξέτασης δίνεται η δυνατότητα αποθήκευσης του path για φωτογραφικό υλικό που θα αφορά την συγκεκριμένη εξέταση του ασθενούς.

ζ) Από την κάθε εξέταση, ο ιατρός μπορεί να εξάγει από καμία έως πολλές διαγνώσεις. Η κάθε διάγνωση θα συνοδεύεται από τον κωδικό ICD10 καθώς και από βοηθητικές παρατηρήσεις.

η) Από την κάθε διάγνωση, ο ιατρός μπορεί να εξάγει από καμία έως πολλές θεραπείες. Η κάθε θεραπεία θα συνοδεύεται από τον κωδικό του σκευάσματος (φαρμάκου) και από βοηθητικές παρατηρήσεις, όπως διάρκεια χορήγησης φαρμάκου, ποσότητα κ.α.

θ) Η κάθε θεραπεία θα συνοδεύεται από το θέμα της, το οποίο μπορεί να είναι Βεβαίωση, Γνωμάτευση, Ιατρική Γνωμάτευση, Ιατρική Συνταγή, Παραπεμπτικό και Πλάνο.

ι) Το προσωπικό της κλινικής δύναται να εκτελέσει προγραμματισμό ραντεβού για το σύνολο των ασθενών του. Ένας ασθενής δεν μπορεί να έχει προγραμματίσει ραντεβού πάνω από μία φορά για την ίδια ημέρα στην κλινική. Κατά την εισαγωγή του νέου ραντεβού, θα εισάγεται και ο ιατρός, ο οποίος θα εκτελεί την εξέταση του ασθενούς, στην προγραμματισμένη ημερομηνία.

ια) Στον προγραμματισμό ενός ραντεβού θα συμπληρώνεται και το είδος του ραντεβού, όπως Διαθλαστικός έλεγχος, Κλινική εξέταση, Οπτικά πεδία και τέλος Πλήρης Οφθαλμολογικός Έλεγχος.

3.4 Μοντέλο Οντοτήτων-Σχέσεων (E-R)

3.4.1 Γενικά

Ένα μοντέλο Οντοτήτων-Σχέσεων (Ο-Σ) περιγράφει την σχέση μεταξύ των αντικειμένων ενδιαφέροντος σε ένα συγκεκριμένο γνωστικό τομέα και αποτελεί συνήθως το αποτέλεσμα μίας συστηματικής ανάλυσης, με σκοπό να ορίσει και να περιγράψει τι είναι σημαντικό και σε ποιο βαθμό. Συγκεκριμένα στον τομέα του software engineer, χρησιμοποιείται για να παρέχει ένα εννοιολογικό σχήμα κατά την σχεδίαση Βάσεων Δεδομένων, ως μοντέλο δεδομένων ενός συστήματος και των απαιτήσεων του με προσέγγιση top-down.Ένα διάγραμμα που δημιουργείται με αυτή την διαδικασία σχεδίασης καλείται Διάγραμμα Οντοτήτων – Σχέσεων (E-R Diagram).

Το μοντέλο Οντοτήτων-Σχέσεων χρησιμοποιείται στο πρώτο στάδιο σχεδίασης ενός συστήματος πληροφοριών, κατά την ανάλυση των απαιτήσεών του. Σκοπός του είναι να περιγράφει

τις αναγκαίες πληροφορίες οι οποίες πρόκειται να αποθηκευτούν στη βάση δεδομένων ή τον τύπο τους. Η μοντελοποίηση δεδομένων γίνεται για την περιγραφή των χρησιμοποιούμενων όρων και των σχέσεων τους σε έναν ορισμένο τομέα ενδιαφέροντος. Στην περίπτωση σχεδιασμού ενός συστήματος πληροφοριών, που στηρίζεται σε μια βάση δεδομένων, το εννοιολογικό μοντέλο δεδομένων χαρτογραφείται σε προχωρημένο στάδιο σε ένα λογικό μοντέλο δεδομένων, όπως το σχεσιακό μοντέλο δεδομένων. Το στάδιο αυτό ονομάζεται συνήθως στάδιο λογικού σχεδιασμού. Ύστερα, κατά τη διάρκεια του φυσικού σχεδιασμού το λογικό μοντέλο χαρτογραφείται σε κάποιο φυσικό μοντέλο. Ορισμένες φορές και οι δύο φάσεις αναφέρονται ως "φυσικός σχεδιασμός".

Τα βασικά στοιχεία ενός μοντέλου Ο-Σ είναι τα εξής:

α) **Οντότητα**. Αποτελεί ένα αντικείμενο ενδιαφέροντος στον πραγματικό κόσμο, το οποίο ξεχωρίζει από τα υπόλοιπα. Μια οντότητα λειτουργεί αφαιρετικά σε έναν πολύπλοκο τομέα. Στιγμιότυπο μιας οντότητας είναι μία συγκεκριμένη περίπτωση ενός τύπου οντότητας.

β) **Τύπος Οντότητας**. Αποτελεί μία συλλογή χαρακτηριστικών, που περιγράφουν την οντότητα.

γ) **Χαρακτηριστικό**. Κάθε οντότητα έχει διάφορα στοιχεία που την προσδιορίζουν. Ένα τέτοιο στοιχείο ονομάζεται ιδιότητα, χαρακτηριστικό ή πεδίο της οντότητας, τα οποία μπορεί να είναι μονότιμα ή πλειότιμα.

δ) **Συσχέτιση**. Είναι η σύνδεση δύο ή περισσοτέρων τύπων οντοτήτων, η οποία παρουσιάζει ενδιαφέρον για σχεδιασμό. Ένας τύπος συσχέτισης παρουσιάζεται με ρόμβο.

ε) **Πληθικότητα**. Περιγράφει τον αριθμό των στιγμιότυπων ενός τύπου οντοτήτων, που μπορούν αντιστοιχίζονται με μία οντότητα ενός άλλου τύπου σε μία συσχέτιση. Μπορούμε να έχουμε συσχετίσεις με λόγο πληθικότητας 1-1 (ένα προς ένα), 1-N (ένα προς πολλά), N-N (πολλά προς πολλά).

στ) **Ασθενής Τύπος Οντότητας**. Είναι μία οντότητα, η οποία εξαρτάται από την ύπαρξη κάποιας άλλης.

ζ) **Πλειότιμα Χαρακτηριστικά**.

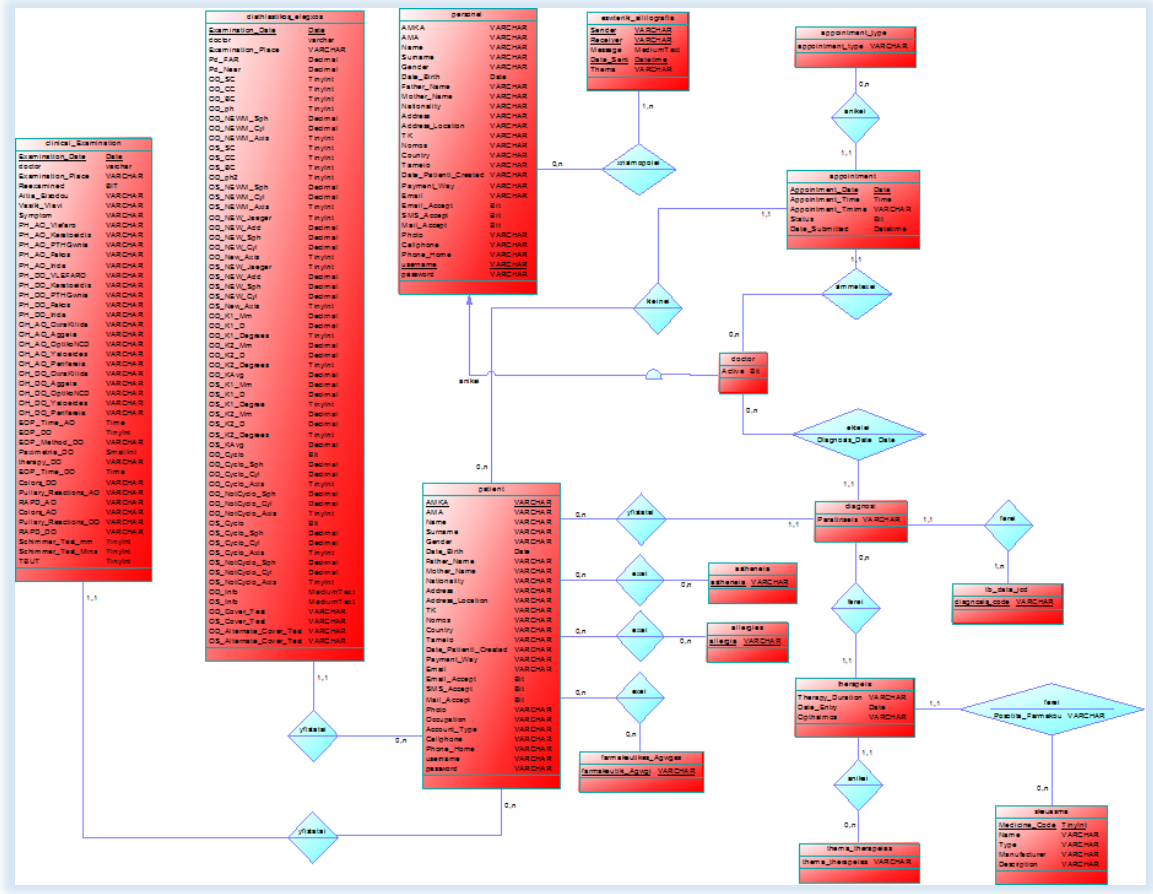
η) **Γενίκευση / Εξειδίκευση**. Με την έννοια γενίκευση (generalization) εννοούμε τον εντοπισμό ενός συνόλου οντοτήτων (κλάση) που έχουν κοινά χαρακτηριστικά με πιο γενικευμένα αντικείμενα (υπέρκλαση). Η εξειδίκευση (specialization) είναι το ακριβώς αντίθετο της γενίκευσης, δηλαδή ο εντοπισμός υποσυνόλων ενός τύπου οντοτήτων με κοινά χαρακτηριστικά, τα οποία τα διαφοροποιούν από τα υπόλοιπα μέλη του.

θ) **Κληρονομικότητα**. Σε κάθε επίπεδο της ιεραρχίας οι τύποι οντοτήτων κληρονομούν τα χαρακτηριστικά των τύπων του αμέσως ανώτερου επιπέδου.

Συνδυάζοντας όλα τα παραπάνω στοιχεία, συνθέτουμε το διάγραμμα Ο-Σ, όπως θα δούμε στην επόμενη ενότητα.

3.4.2 Υλοποίηση Διαγράμματος E-R στην ΒΔ

Το πρώτο βήμα, το οποίο απαιτείται να εκτελεστεί είναι η δημιουργία του διαγράμματος Οντοτήτων – Σχέσεων, σύμφωνα με τις παραπάνω παραδοχές και λεπτομέρειες. Συνεπώς, καταλήγουμε στο παρακάτω διάγραμμα Οντοτήτων – Σχέσεων για την ΒΔ της εφαρμογής.



Εικόνα 13. Διάγραμμα Οντοτήτων –Σχέσεων της ΒΔ

Στο παραπάνω διάγραμμα, με κόκκινο χρώμα φαίνονται 16 οντότητες, ενώ με κυανό χρώμα φαίνονται οι συσχετίσεις μεταξύ τους. Η ανάλυση του διαγράμματος οδηγεί στην δημιουργία ενός σωστού και αξιόπιστου σχήματος ΒΔ.

Με την βοήθεια του παραπάνω διαγράμματος και σε συνδυασμό με τις παραδοχές που έχουν οριστεί, διευκολύνεται κατά πολύ η δημιουργία των πινάκων και των σχέσεων της ΒΔ, όπως θα δούμε στην συνέχεια.

3.5 Σχεσιακό Μοντέλο Δεδομένων

3.5.1 Γενικά

Το Σχεσιακό Μοντέλο Δεδομένων αποτελεί το κύριο μοντέλο σχεδίασης Βάσης Δεδομένων παγκοσμίως. Το συγκεκριμένο μοντέλο χαρακτηρίζεται από την απλότητα του και έχει όλες εκείνες τις ιδιότητες και δυνατότητες, που απαιτούνται για την γρήγορη και αποτελεσματική επεξεργασία δεδομένων.

Τα δεδομένα είναι δυνατόν να αναδιοργανώνονται με πολλούς διαφορετικούς τρόπους, σε νοητούς πίνακες, χωρίς να είναι απαραίτητη η αναδιοργάνωση των φυσικών πινάκων που τα αποθηκεύουν. Οι ερωτήσεις, είτε από το χρήστη είτε από λογισμικό, προς τη βάση δεδομένων, γίνονται συνήθως μέσω της διαδεδομένης διαλογικής γλώσσας SQL (Structured Query Language).

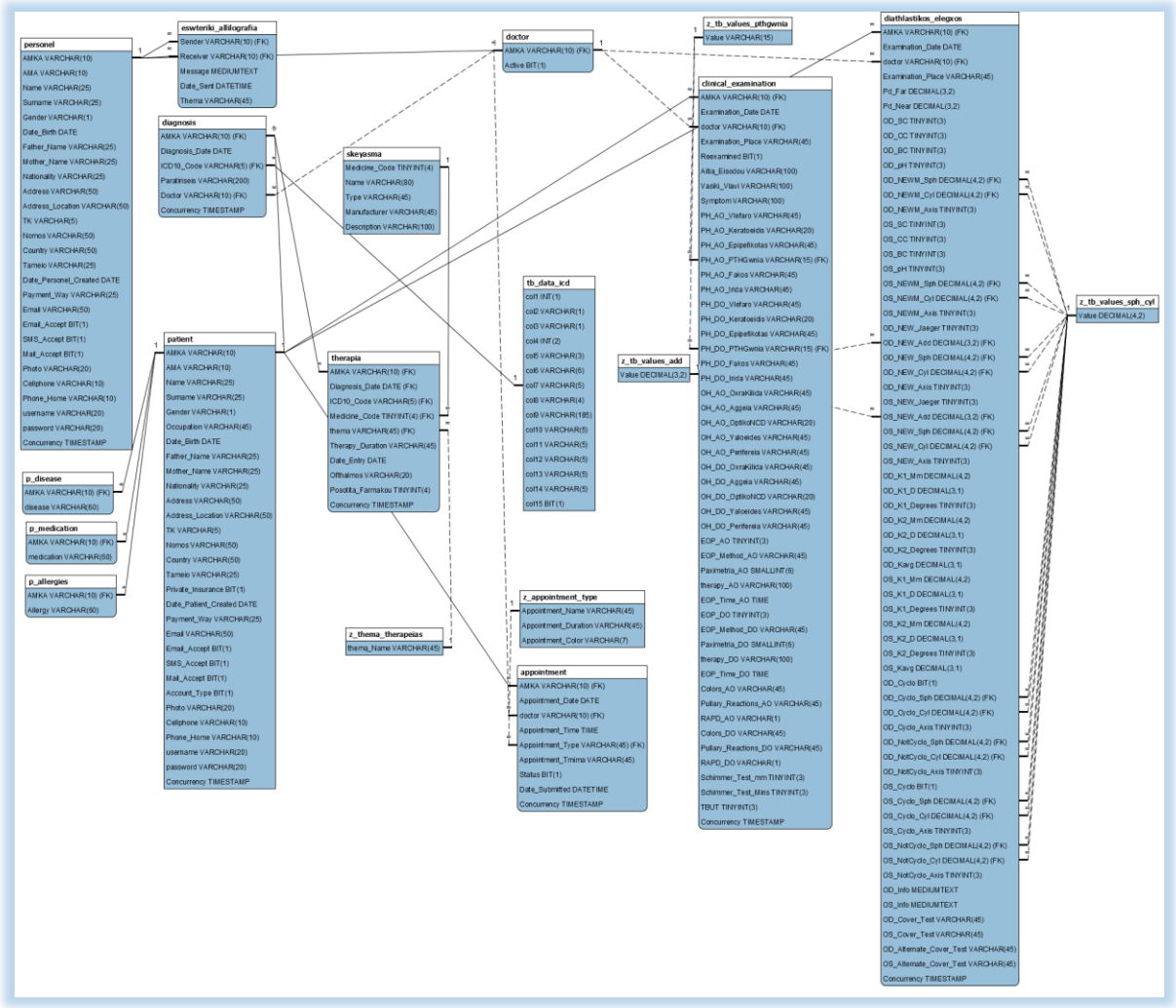
Εκτελώντας ερωτήματα ο χρήστης (ή το λογισμικό που εκπροσωπεί το χρήστη) είναι δυνατόν, ανάλογα με τα δικαιώματά του, να δημιουργήσει, να μεταβάλλει και να διαγράψει δεδομένα στη βάση, ή να ανασύρει πληροφορίες με σύνθετα κριτήρια αναζήτησης.

Σε αυτό το μοντέλο, τα δεδομένα μιας εφαρμογής αναπαρίστανται ως ένα σύνολο από σχέσεις (relations) οι οποίες μπορεί να είναι πίνακες-αρχεία. Στις πιο πολλές περιπτώσεις υιοθετείται η χρήση πινάκων (tables) που περιέχουν ένα πλήθος γραμμών (rows) και στηλών (columns). Η κάθε μια από αυτές τις γραμμές – οι οποίες στην ορολογία του μοντέλου ονομάζονται και πλειάδες (tuples) – περιέχει ένα σύνολο απλών πεδίων (attributes), τα οποία συσχετίζονται μεταξύ τους. Επειδή οι πίνακες χρησιμοποιούνται για την αναπαράσταση των τύπων οντοτήτων καθώς και των τύπων συσχετίσεων που υφίστανται ανάμεσά τους, μπορούμε να θεωρήσουμε κάθε μια από τις γραμμές ενός πίνακα σαν ένα στιγμιότυπο οντότητας ή συσχέτισης ανάλογα με το αντικείμενο στο οποίο αναφέρεται.

Τέλος κάθε πεδίο ή στήλη ενός πίνακα, δέχεται τιμές οι οποίες ανήκουν σε ένα συγκεκριμένο και καθορισμένο, εκ των προτέρων, σύνολο τιμών (domain). Το είδος των τιμών αυτού του συνόλου καθορίζεται από τον τύπο δεδομένων του πεδίου του πίνακα, ο οποίος με τη σειρά του ορίζεται κατά το στάδιο της λογικής σχεδίασης της εφαρμογής.

3.5.2 Υλοποίηση Σχεσιακού Μοντέλου της ΒΔ

Εφαρμόζοντας τα παραπάνω και με την βοήθεια του διαγράμματος Οντοτήτων – Σχέσεων, εξάγονται οι πίνακες και οι σχέσεις μεταξύ τους, καταλήγοντας στο σχήμα της Βάσης Δεδομένων της εφαρμογής, όπως φαίνεται στο ακόλουθο διάγραμμα:



Εικόνα 14. Διάγραμμα Σχεσιακού Μοντέλου της ΒΔ

Στο παραπάνω διάγραμμα παρουσιάζονται οι 19 πίνακες της ΒΔ καθώς και οι σχέσεις που υφίστανται μεταξύ τους. Εκ πρώτης όψεως φαίνεται ότι οι Οντότητες που περιλάμβανε το Μοντέλο Οντοτήτων-Σχέσεων, έχουν μετατραπεί σε πίνακες, καθώς αποτελεί βασικό κανόνα για την δημιουργία του Σχεσιακού Μοντέλου ΒΔ προερχόμενου από το Μοντέλο Οντοτήτων-Σχέσεων. Στην συνέχεια, το είδος των σχέσεων των Οντοτήτων καθώς και η πληθικότητα τους, καθόρισαν την δημιουργία επιπλέον πινάκων ή την προσθήκη επιπλέον πεδίων στους ήδη ορισμένους πίνακες. Για παράδειγμα, ο πίνακας “diagnosis”, εμπλουτίστηκε με επιπλέον πεδία, τα οποία προέκυψαν από την σχέση της Οντότητας “diagnosis” με τις υπόλοιπες Οντότητες στο Μοντέλο Οντοτήτων-Σχέσεων. Τέλος, έχουν δημιουργηθεί επιπλέον πίνακες, των οποίων ο ρόλος είναι βοηθητικός. Για παράδειγμα, ο πίνακας “z_tb_values_sph_cyl”, περιλαμβάνει όλες τις επιτρεπτές τιμές, που μπορούν να έχουν συγκεκριμένα πεδία στον πίνακα “diathlastikos_elegxos”, και που αφορούν τιμές μετρήσεων στην συγκεκριμένη εξέταση. Σε ότι αφορά τις σχέσεις των πινάκων μεταξύ τους, αυτές είναι κυρίως σχέσεις τύπου ένα-προς-πολλά (1-N).

Παρακάτω παρατίθεται μία σύντομη παρουσίαση του κάθε πίνακα για το Σχεσιακό Μοντέλο της ΒΔ:

α) Πίνακας "**personel**": Εδώ αποθηκεύονται τα άτομα που απαρτίζουν το προσωπικό της κλινικής είτε αυτό είναι ιατρικό είτε όχι. Κάθε εγγραφή αφορά έναν εργαζόμενο και περιλαμβάνει κυρίως δημογραφικά δεδομένα και στοιχεία επικοινωνίας.

β) Πίνακας "**doctor**": Περιλαμβάνει το σύνολο των ΑΜΚΑ των εργαζομένων, οι οποίοι είναι ιατροί. Με αυτόν τον τρόπο γίνεται μία διάκριση του ιατρικού προσωπικού της κλινικής, η οποία απαιτείται για την υλοποίηση συγκεκριμένων λειτουργιών της εφαρμογής.

γ) Πίνακας "**patient**": Περιλαμβάνει όλα τα δημογραφικά στοιχεία και δεδομένα επικοινωνίας των ασθενών της κλινικής.

δ) Πίνακας "**p_disease**": Περιλαμβάνει τις ασθένειες που ενδέχεται να έχει ο ασθενής κατά την επίσκεψη του στην κλινική. Αποτελεί πληροφορία, η οποία είναι χρήσιμη στον εξεταστή-ιατρό και που μπορεί να τον επηρεάσει στην διάγνωση του.

ε) Πίνακας "**p_medication**": Περιλαμβάνει τις φαρμακευτικές αγωγές που μπορεί να λαμβάνει ο ασθενής κατά την περίοδο επισκέψεως του στην κλινική και που ενδέχεται να επηρεάσουν την κρίση του ιατρού-εξεταστή στην διάγνωση του.

στ) Πίνακας "**p_allergies**": Περιλαμβάνει τις αλλεργίες που μπορεί να έχει ό ασθενής και που ενδέχεται να επηρεάσουν την κρίση του ιατρού-εξεταστή στην διάγνωση του.

ζ) Πίνακας "**clinical_examination**": Περιλαμβάνει τα δεδομένα εξέτασης τύπου «κλινική εξέταση» για κάθε ασθενή. Κάθε εγγραφή αφορά μία μόνο εξέταση ενός ασθενούς.

η) Πίνακας "**diathalstikos_elegxos**": Περιλαμβάνει τα δεδομένα εξέτασης τύπου «διαθλαστικός έλεγχος» για κάθε ασθενή. Κάθε εγγραφή αφορά μία μόνο εξέταση ενός ασθενούς.

θ) Πίνακας "**diagnosis**": Περιλαμβάνει τις διαγνώσεις που προκύπτουν από μια εξέταση, στην οποία υποβλήθηκε ο ασθενής. Αποθηκεύει δεδομένα, όπως τον κωδικό ICD10, τον ιατρό που εκτελεί την διάγνωση, εξεταζόμενος ασθενής, ημερομηνία διάγνωσης κ.α.

ι) Πίνακας "**tb_data_icd**": Περιλαμβάνει το σύνολο των κωδικών διαγνώσεων για τις οφθαλμολογικές παθήσεις, σύμφωνα με το Διεθνές Πρότυπο ICD10.

ια) Πίνακας "**therapeia**": Περιλαμβάνει τις θεραπείες που προκύπτουν από μία διάγνωση. Αποθηκεύει δεδομένα, όπως κωδικός φαρμάκου, διάρκεια φαρμακευτικής αγωγής κ.α.

ιβ) Πίνακας "**z_thema_therapeias**": Περιλαμβάνει τις επιτρεπτές τιμές ενός Θέματος Θεραπείας.

ιγ) Πίνακας "**skeyasma**": Περιλαμβάνει τα δεδομένα των φαρμάκων που χορηγούνται σε μία θεραπεία και φιλοξενεί δεδομένα, όπως κωδικός φαρμάκου, κατασκευαστής κ.α.

ιδ) Πίνακας "**appointment**": Περιλαμβάνει τα δεδομένα των προγραμματισμένων ραντεβού για το σύνολο των ασθενών.

ιε) Πίνακας "**z_appointment_type**": Περιλαμβάνει τις επιτρεπτές τιμές τύπου ενός ραντεβού, όπως Διαθλαστικός Έλεγχος, Κλινική Εξέταση, Οπτικά Πεδία και Πλήρης Οφθαλμολογικός Έλεγχος.

ιστ) Πίνακας "**z_tb_values_add**": Περιλαμβάνει τις επιτρεπτές τιμές για συγκεκριμένες μετρήσεις στον διαθλαστικό έλεγχο.

ιζ) Πίνακας "**z_tb_values_sph_cyl**": Περιλαμβάνει τις επιτρεπτές τιμές για συγκεκριμένες μετρήσεις στον διαθλαστικό έλεγχο.

ιη) Πίνακας "**z_tb_values_pthgwnia**": Περιλαμβάνει τις επιτρεπτές τιμές για συγκεκριμένες μετρήσεις στην κλινική εξέταση.

ιθ) Πίνακας “**eswteriki_allilografia**”: Περιλαμβάνει δεδομένα σχετικά με την εσωτερική αλληλογραφία του προσωπικού της κλινικής.

Στο Παράρτημα της παρούσας ΔΕ παρατίθενται οι εντολές SQL, οι οποίες απαιτούνται για την δημιουργία των παραπάνω πινάκων. Επίσης, φαίνονται τα primary keys, τα foreign keys και επιπλέον λεπτομέρειες όπως constraints για τον κάθε πίνακα.

4 CrystalEye Solution

4.1 StoreDatabase Project

4.1.1 Entity Framework

Το Entity Framework είναι ένα αντικειμενοστραφές, open-source framework, και object-relational mapping Framework, το οποίο αποτελείται από ένα σύνολο τεχνολογιών στο ADO.NET και υποστηρίζει την ανάπτυξη εφαρμογών, οι οποίες χαρακτηρίζονται ως data-oriented.

Τα δύο βασικά προβλήματα που έπρεπε στο παρελθόν να ξεπεράσουν οι προγραμματιστές είναι αφενός μεν ότι έπρεπε να μοντελοποιήσουν τις οντότητες, τις σχέσεις και την λογική του επιχειρηματικού προβλήματος που ήθελαν να επιλύσουν και αφετέρου δε ότι απαιτούνταν η ενασχόληση με την εκάστοτε μηχανή δεδομένων, η οποία χρησιμοποιούνταν για αποθήκευση και ανάκτηση δεδομένων. Επίσης, είναι γνωστό ότι τα δεδομένα μπορούν να εκτείνονται σε πολλαπλά συστήματα αποθήκευσης, το καθένα από τα οποία χρησιμοποιεί δικά του πρωτόκολλα.

Το Entity Framework επιτρέπει στους προγραμματιστές να εργάζονται με δεδομένα με την μορφή οντοτήτων και των ιδιοτήτων τους, όπως πχ πελάτης και διεύθυνση πελάτη, χωρίς να χρειάζεται να ασχοληθούν με τους υφιστάμενους πίνακες και τα πεδία τους στην Βάσης Δεδομένων, όπου αποθηκεύονται τα δεδομένα. Συνεπώς, η εργασία με τα δεδομένα υφίσταται σε ένα υψηλότερο αφαιρετικό επίπεδο και πλέον γίνεται εφικτή η δημιουργία και η συντήρηση data-oriented εφαρμογών, με λιγότερο κώδικα σε σχέση με τις παραδοσιακές εφαρμογές. Τέλος, επειδή το Entity Framework είναι συστατικό του .NET Framework, μπορεί να εκτελεστεί σε οποιοδήποτε υπολογιστή, στον οποίο το .NET Framework είναι εγκατεστημένο.

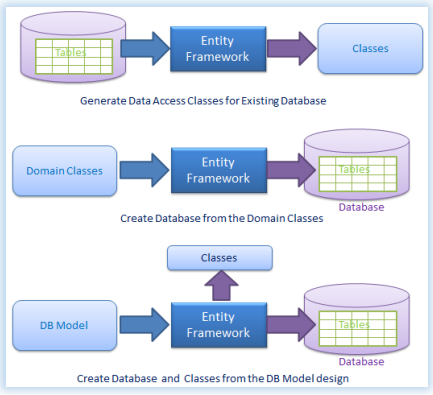
Το Entity Framework είναι χρήσιμο για τις εξής 3 περιπτώσεις:

α) εάν υπάρχει ήδη το σχήμα της Βάσης Δεδομένων ή εάν απαιτείται η δημιουργία της Βάσης να προηγηθεί από την ανάπτυξη διαφόρων λειτουργιών της εφαρμογής,

β) εάν θέλουμε αρχικά να επικεντρωθούμε στην δημιουργία των οντοτήτων και έπειτα να ακολουθήσει η δημιουργία της Βάσης Δεδομένων, και τέλος,

γ) εάν θέλουμε να δημιουργήσουμε το σχήμα της Βάσης Δεδομένων στο visual designer και έπειτα να δημιουργήσουμε τις αντίστοιχες κλάσεις.

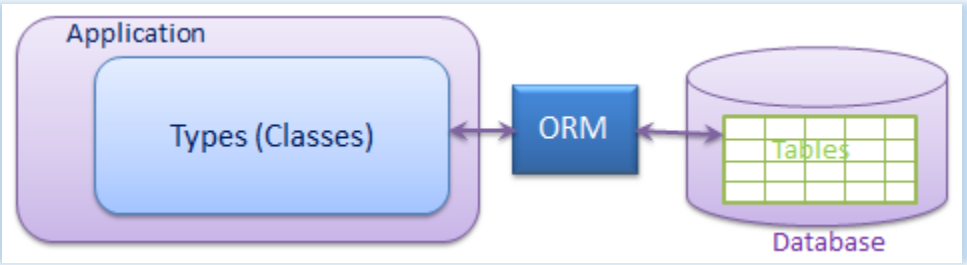
Τα 3 αυτά σενάρια φαίνονται στο παρακάτω σχήμα:



Εικόνα 15. Περιπτώσεις χρήσης του Entity Framework

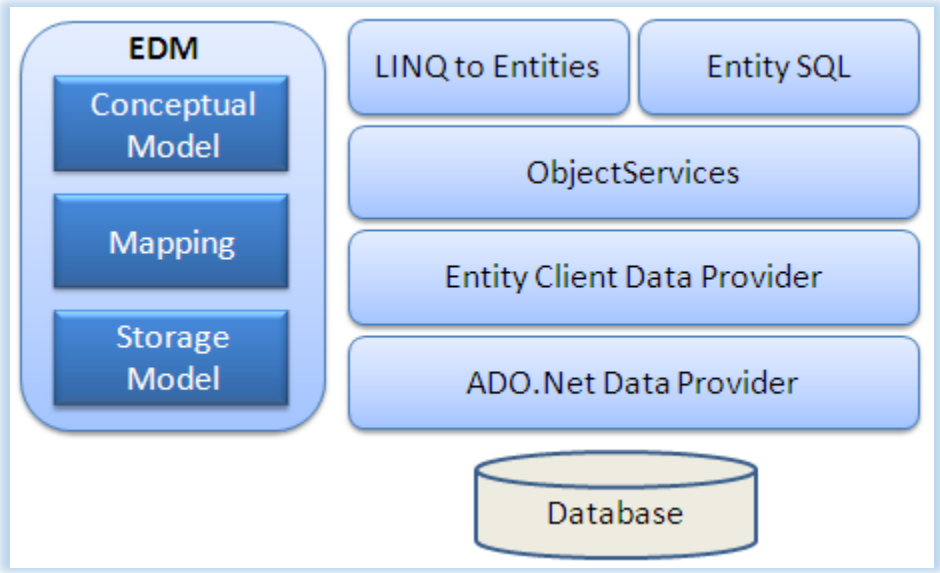
Το Entity Framework, όπως αναφέρθηκε προηγουμένως, είναι ένα object-relational model (ORM) Framework. Αυτό το μοντέλο αποτελεί ένα εργαλείο αποθήκευσης δεδομένων προερχόμενων από οντότητες προς μία σχεσιακή Βάση Δεδομένων, όπως ο MS-SQL Server, με έναν αυτοματοποιημένο τρόπο, που περιορίζει αρκετά τον απαιτούμενο κώδικα για την πραγματοποίηση της αποθήκευσης. Το ORM περιλαμβάνει τρείς βασικές ενότητες: τις Οντότητες, τα αντικείμενα της ΒΔ, και τις Mapping πληροφορίες, οι οποίες αφορούν τον τρόπο με τον οποίο οι Οντότητες απεικονίζονται ορθώς στα αντικείμενα της ΒΔ, όπως πίνακες, Views και Stored Procedures. Επίσης, το ORM επιτρέπει τον σαφή διαχωρισμό της σχεδίασης της Βάσης Δεδομένων από την σχεδίαση των κλάσεων – Οντοτήτων. Αυτό, ουσιαστικά, παρέχει περισσότερες δυνατότητες συντηρισιμότητας και επεκτασιμότητας της εφαρμογής. Επίσης, αυτοματοποιεί τις στάνταρ διαδικασίες (Create, Read, Update, Delete), έτσι ώστε ο προγραμματιστής να μην χρειάζεται να γράψει κώδικα για την επίτευξη τους.

Ένα τυπικό ORM εργαλείο, παράγει κλάσεις για την διασύνδεση με την Βάση Δεδομένων μίας εφαρμογής, όπως φαίνεται στο παρακάτω σχήμα:



Εικόνα 16. Λειτουργία Object-Relational Mapping (ORM)

Παρακάτω, παρατίθεται η γενικότερη αρχιτεκτονική που ακολουθεί το Entity Framework:



Εικόνα 17. Αρχιτεκτονική του Entity Framework

Όπως φαίνεται στο παραπάνω σχήμα, τα κύρια συστατικά, από τα οποία απαρτίζεται το Entity Framework είναι τα εξής:

α) EDM (Entity Data Model): Αποτελείται από 3 μέρη, το Conceptual, Mapping και Storage Model.

β) Conceptual Model: Περιλαμβάνει τις κλάσεις μοντελοποίησης και τις σχέσεις μεταξύ τους. Είναι ανεξάρτητο του μοντέλου σχεδίασης της Βάσης Δεδομένων.

γ) Storage Model: Αποτελεί το μοντέλο σχεδίασης της Βάσης Δεδομένων, το οποίο περιλαμβάνει τους πίνακες, τα Views, τα Stored Procedures καθώς και τις σχέσεις μεταξύ τους όπως και τα κλειδιά (keys).

δ) Mapping: Περιλαμβάνει πληροφορίες σχετικά με τον τρόπο που το Conceptual Model απεικονίζει το Storage Model.

ε) LINQ to Entities: Η LINQ είναι μία γλώσσα παραγωγής ερωτημάτων (query language), η οποία χρησιμοποιείται για την δημιουργία ερωτημάτων προς το μοντέλο των αντικειμένων. Επιστρέφει Οντότητες, οι οποίες ορίζονται στο Conceptual Model.

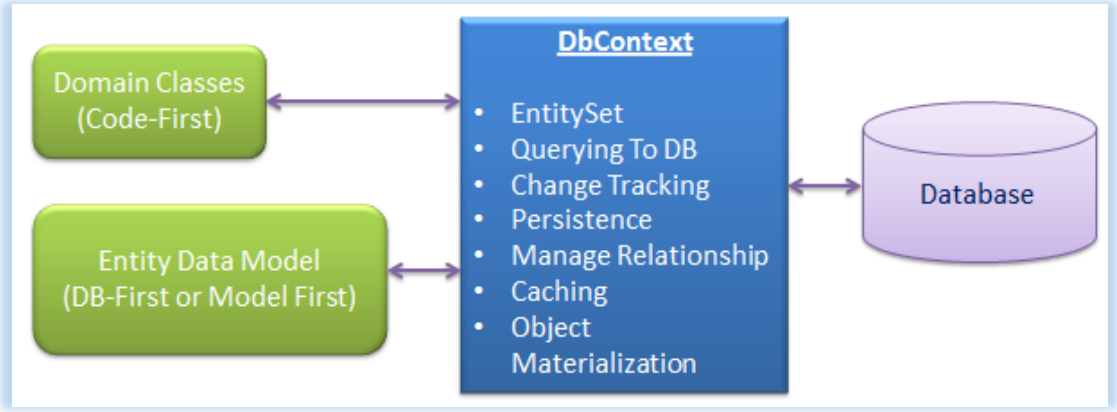
στ) Entity SQL: Αποτελεί άλλη μία γλώσσα παραγωγής ερωτημάτων.

ζ) Object Service: Αποτελεί ένα αρχικό στάδιο προσπέλασης των δεδομένων στην Βάση Δεδομένων. Η υπηρεσία αυτή ευθύνεται για την διαδικασία γνωστή ως materialization, η οποία μετατρέπει τα δεδομένα, τα οποία έρχονται από τον entity object client data provider προς μία δομή entity object.

η) Entity Client Data Provider: Ευθύνεται για την μετατροπή των ερωτημάτων, που δημιουργήθηκαν είτε μέσω LINQ είτε μέσω Entity SQL, σε SQL ερωτήματα, τα οποία είναι κατανοητά από την Βάση Δεδομένων. Επίσης, επικοινωνεί με τον ADO.Net data provider, ο οποίος με την σειρά του αποστέλλει ή επιστρέφει δεδομένα από την Βάση Δεδομένων.

θ) ADO.Net Data Provider: Το επίπεδο αυτό επικοινωνεί με την Βάση Δεδομένων, χρησιμοποιώντας στάνταρ λειτουργίες ADO.Net.

Απαραίτητο στοιχείο στην αρχιτεκτονική του EF, αποτελεί η κλάση DBContext. Η εν λόγω κλάση είναι η γέφυρα μεταξύ των Οντοτήτων και της Βάσης Δεδομένων, όπως φαίνεται στην παρακάτω εικόνα:



Εικόνα 18. DBContext Class

Η DBContext κλάση είναι η κύρια κλάση, η οποία ευθύνεται για την διαχείριση των δεδομένων, ως αντικείμενα. Συγκεκριμένα, είναι υπεύθυνη για τις ακόλουθες διαδικασίες:

α) Entity Set: Η DBContext κλάση περιέχει ένα entity set για το σύνολο των Οντοτήτων που αντιστοιχίζονται σε πίνακες της Βάσης Δεδομένων.

β) Quering: Η DbContext κλάση μετατρέπει τα ερωτήματα, που προέρχονται από την γλώσσα LINQ σε SQL ερωτήματα και τα αποστέλλει στην Βάση Δεδομένων.

γ) Change Tracking: Παρακολουθεί για αλλαγές, που έχουν υποστεί οι Οντότητες, μετά το πέρας εκτέλεσης του αντίστοιχου ερωτήματος στην Βάση Δεδομένων.

δ) Persisting Data: Ευθύνεται για την εκτέλεση των βασικών λειτουργιών Create,Read,Update και Delete, σε σχέση με την κατάσταση στην οποία βρίσκονται οι Οντότητες.

ε) Caching: Αποθηκεύει τις Οντότητες, που έχουν δημιουργηθεί μετά από την εκτέλεση ενός ερωτήματος, κατά την διάρκεια ζωής μίας context κλάσης.

στ) Manage Relationship: Διαχειρίζεται τις σχέσεις μεταξύ των Οντοτήτων με την χρήση ειδικών πρωτοκόλλων, όπως CSDL, MSL κ.α.

ζ) Object Materialization: Η DBContext κλάση μετατρέπει τα δεδομένα, που προέρχονται από τους πίνακες της Βάσης Δεδομένων σε Οντότητες.

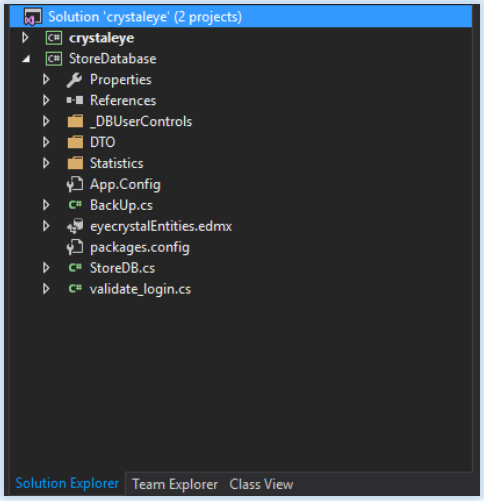
Επισημαίνεται ότι για να επωφεληθούμε από τις δυνατότητες, που παρέχει η DBContext κλάση, απαιτείται προηγουμένως η αρχικοποίηση της, όπως θα δούμε στην συνέχεια.

Σε ότι αφορά το μοντέλο Concurrency, που εφαρμόζει το EF ως προεπιλογή, είναι οπτιμιστικό. Αυτό σημαίνει ότι δεν κλειδώνονται τα δεδομένα στο data source, όταν τα δεδομένα ανακτώνται ή όταν ενημερώνονται. Ωστόσο, επιτρέπει την εκτέλεση concurrency με χρήση των κατάλληλων ρυθμίσεων. Συγκεκριμένα, απαιτείται να τεθεί η ιδιότητα concurrency=fixed για τις οντότητες, που απαιτείται να υφίστανται έλεγχο concurrency. Στην συνέχεια, με την χρήση κατάλληλου πεδίου τύπου Timestamp στην ΒΔ, πραγματοποιείται έλεγχος concurrency στους πίνακες της ΒΔ. Στην ανάλυση του κώδικα, που ακολουθεί σε επόμενο κεφάλαιο, γίνεται επεξήγηση της διαδικασίας που ακολουθείται.

Τέλος, αξίζει να γίνει μια αναφορά στις εκδόσεις του Entity Framework. Το .NET Framework 4.0/4.5 περιλαμβάνει τον πυρήνα του Entity Framework 5.0. Ωστόσο, αυτό έχει αλλάξει με την έκδοση 6.0 του EF και δεν εξαρτάται πλέον από το .Net Framework. Στην παρούσα εργασία αξιοποιήθηκε η έκδοση EF 5.0, καθώς η MySQL δεν είναι ακόμη συμβατή με την έκδοση 6.0.

4.1.2 Ανάλυση StoreDatabase Project

Όπως έχει ήδη αναφερθεί, το εν λόγω Project, περιλαμβάνει όλο τον κώδικα, ο οποίος ευθύνεται για την επικοινωνία της εφαρμογής με την Βάση Δεδομένων καθώς και για την εκτέλεση των στάνταρ λειτουργιών CRUD (Create, Read, Update, Delete). Επιπρόσθετα, περιλαμβάνει αρχεία κλάσεων, τα οποία ευθύνονται για την διεκπεραίωση συγκεκριμένων λειτουργιών, όπως το back up της ΒΔ, την δημιουργία στατιστικών δεδομένων κ.α. Παρακάτω παρατίθεται η δομή του Project, όπως φαίνεται από τον Solution Explorer του Visual Studio:



Εικόνα 19. Δομή του StoreDatabase Project

Όπως φαίνεται παραπάνω, το StoreDatabase Project αποτελείται από αρχεία-κλάσεις με επέκταση .cs, configuration files, φακέλους, η χρήση των οποίων τμηματοποιεί λογικά τον κώδικα, το αρχείο με επέκταση .edmx, που περιλαμβάνει τα επιμέρους αρχεία μοντελοποίησης του EF κ.α. Συγκεκριμένα περιλαμβάνονται τα εξής:

α) Properties, References: Περιλαμβάνουν τις ιδιότητες του Project, και τις απαιτούμενες αναφορές σε άλλα project αντίστοιχα.

β) _DBUserControls: Περιλαμβάνει τα UserControls, τα οποία αποτελούν δομικό στοιχείο του GUI μιας εφαρμογής υλοποιημένη σε WPF, και σχετίζονται με την σύνδεση και αποσύνδεση ενός χρήστη. Η χρησιμότητα των Usercontrols θα αναλυθεί σε επόμενο κεφάλαιο.

γ) DTO: Περιλαμβάνει τα Data Transfer Objects, τα οποία αποτελούν οντότητες, που περιέχουν λιγότερα attributes από τις οντότητες, που δημιουργεί αυτόματα το EF. Ακολουθεί λεπτομερής ανάλυση των DTO στην συνέχεια.

δ) Statistics: Περιλαμβάνει τις κλάσεις, οι οποίες περιέχουν τον κώδικα που ευθύνεται για την παραγωγή των στατιστικών στοιχείων στα δεδομένα της εφαρμογής.

ε) App.config: Αποτελεί αρχείο ρυθμίσεων του Project και δημιουργείται αυτόματα από το Visual Studio. Αποθηκεύονται πληροφορίες, όπως το connectionstring με την Βάση Δεδομένων.

στ) BackUp.cs: Περιλαμβάνει τον κώδικα, που ευθύνεται για τις διαδικασίες back up και restore της ΒΔ. Ακολουθεί εκτενής ανάλυση στην συνέχεια.

ζ) eyecrystalEntities.edmx: Περιλαμβάνει όλα τα αρχεία, που δημιουργεί αυτόματα το EF, κατά την μοντελοποίηση των στοιχείων της ΒΔ σε Οντότητες.

η) packages.config: Αποτελεί βοηθητικό αρχείο ρυθμίσεων του EF.

θ) StoreDB.cs: Αποτελεί ίσως το πιο βασικό αρχείο κλάσης στο εν λόγω Project. Περιλαμβάνει όλες τις μεθόδους, με τις οποίες πραγματοποιούνται οι λειτουργίες της εφαρμογής και αφορούν στην επικοινωνία με την ΒΔ. Στην συνέχεια, θα γίνει εκτενής αναφορά στην δομή της υπ' όψη κλάσης.

ι) validate_login.cs: Κλάση, η οποία περιλαμβάνει τον κώδικα που απαιτείται για το validation της φόρμας Login.

Παρακάτω ακολουθεί η περιγραφή μερικών βασικών αρχείων του Project και η ανάλυση του κώδικα που περιλαμβάνουν:

eyecrystalEntities.edmx

Το αρχείο eyecrystalEntities.edmx, είναι ένα XML αρχείο, το οποίο ορίζει το μοντέλο Οντοτήτων, το μοντέλο της ΒΔ και την απεικόνιση που έχουν τα δύο μοντέλα μεταξύ τους. Περιέχει πληροφορίες, οι οποίες χρησιμοποιούνται από το ADO.Net Entity Data Model Designer, προκειμένου να εξάγει το μοντέλο γραφικά.

Το υπόψη αρχείο, καθώς και τα υπόλοιπα τα οποία ανήκουν ιεραρχικά σε αυτό, δημιουργούνται αυτόματα από το EF. Οι σημαντικότερες κλάσεις που δημιουργεί αυτόματα το EF είναι αυτές των Οντοτήτων. Ενδεικτικά, παρακάτω βλέπουμε την κλάση diagnosis.cs, η οποία αντιστοιχεί στην Οντότητα των διαγνώσεων:

```
namespace StoreDatabase
{
    using System;
    using System.Collections;
    using System.Collections.Generic;
    using System.ComponentModel;
    using System.Linq;
    public partial class diagnosis : INotifyDataErrorInfo,INotifyPropertyChanged
    {
        //dictionary pou tha apothikeuei ta errors gia kathe property
        private Dictionary<string, List<string>> property_errors = new
Dictionary<string, List<string>>();

        public diagnosis()
        {
            this.therapia = new HashSet<therapia>();
        }

        public string AMKA { get; set; }
        public System.DateTime Diagnosis_Date { get; set; }
        private string iCD10_Code;
        public string ICD10_Code
        {
            get { return iCD10_Code; }
            set
            {
                iCD10_Code = value;
                OnPropertyChanged("ICD10_Code");
            }
        }
        private string paratiriseis;
        public string Paratiriseis
        {
            get { return paratiriseis; }
            set
            {
                paratiriseis = value;
                OnPropertyChanged("Paratiriseis");
            }
        }
        private string doctor;
        public string Doctor
        {
```

```
            get { return doctor; }
            set
            {
                doctor = value;
                OnPropertyChanged("Doctor");
            }
        }

        public System.DateTime Concurrency { get; set; }

        public virtual patient patient { get; set; }
        public virtual tb_data_icd tb_data_icd { get; set; }
        public virtual doctor doctor1 { get; set; }
        public virtual ICollection<therapia> therapia { get; set; }

        #region INotifyPropertyChanged
        public event PropertyChangedEventHandler PropertyChanged;

        public void OnPropertyChanged(string name)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(name));
                Validate();
            }
        }

        #endregion

        #region INotifyDataErrorInfo
        public event EventHandler<DataErrorsChangedEventArgs> ErrorsChanged;

        public void RaiseErrorsChanged(string propertyName)
        {

            EventHandler<DataErrorsChangedEventArgs> handler = ErrorsChanged;
            if (handler == null) return;
            var arg = new DataErrorsChangedEventArgs(propertyName);
            handler.Invoke(this, arg);
        }

        public IEnumerable GetErrors(string propertyName)
        {
            List<string> errors = new List<string>();
            if (propertyName != null)
            {
                property_errors.TryGetValue(propertyName, out errors);
                return errors;
            }

            else
                return null;
        }

        public bool HasErrors
        {
            //epistrefei true ean iparxoun errors, vasismeno se count twn errors
sto dictionary mas.
            get
            {
```

```
                try
                {
                    var propErrorsCount = property_errors.Values.FirstOrDefault(r
=> r.Count > 0);
                    if (propErrorsCount != null)
                        return true;
                    else
                        return false;
                }
                catch { }
                return true;
            }
        }

        #endregion

        private void Validate()
        {
            DataValidation();
        }

        private void DataValidation()
        {
            //Validate ICD10_Code
            List<string> listErrorsForICD10_Code;

            if (property_errors.TryGetValue("ICD10_Code", out
listErrorsForICD10_Code) == false)
                listErrorsForICD10_Code = new List<string>();
            else
                listErrorsForICD10_Code.Clear();

            if (ICD10_Code==null || ICD10_Code.Length==0)
            {
                listErrorsForICD10_Code.Add("Δεν μπορεί ο κωδικός ICD10 να είναι
κενός.");         //add stin lista
            }

            property_errors["ICD10_Code"] = listErrorsForICD10_Code;
//register sto dictionary

            if (listErrorsForICD10_Code.Count > 0) {
RaiseErrorsChanged("ICD10_Code"); }

        }
    }
}
```

Αρχικά, γίνεται αντιληπτό ότι η εν λόγω κλάση κληρονομεί τα interfaces INotifyDataErrorInfo και INotifyPropertyChanged. Πρόκειται για δύο σημαντικά interfaces, τα οποία διαδραματίζουν σημαντικό ρόλο στην λειτουργικότητα της εφαρμογής. Επίσης, παρατηρούμε το σύνολο των attributes, τα οποία περιέχει η κλάση και που ουσιαστικά είναι η απεικόνιση των πεδίων που περιλαμβάνει ο πίνακας diagnosis της ΒΔ.

Το πρώτο interface (INotifyDataErrorInfo) σχετίζεται με τον έλεγχο validation, ο οποίος εκτελείται όποτε αλλάζει το value από ένα attribute της κλάσης. Μέσω της setter, που υπάρχει σε κάθε attribute, εκτελείται η μέθοδος OnPropertyChanged(), η οποία ενημερώνει το context ότι η κατάσταση της Οντότητας έχει αλλάξει και είναι διαφοροποιημένη σε σχέση με την Οντότητα που

δημιουργήθηκε αρχικά, ως αποτέλεσμα του ερωτήματος που εκτελέστηκε στην Βάση Δεδομένων. Στην συνέχεια, εκτελείται η μέθοδος Validate(), όπου γίνεται ένας έλεγχος συμφωνίας της καινούριας τιμής του attribute, με τους κανόνες validation, που έχουν οριστεί. Για παράδειγμα, στην συγκεκριμένη περίπτωση ο κωδικός ICD10 δεν μπορεί να είναι ποτέ κενός ή null, διότι τότε δεν υφίσταται διάγνωση. Ωστόσο εάν μία τιμή δεν γίνει αποδεκτή, τότε εισάγεται ένα νέο στοιχείο σφάλματος σε ένα dictionary (property_erros), και ταυτόχρονα ενεργοποιείται ένα event (RaiseErrorsChanged()), το οποίο γίνεται handle στην εφαρμογή σε ανώτερο επίπεδο, με την βοήθεια που παρέχει η γλώσσα WPF, μέσω των λειτουργιών bubbling και tunneling, τις οποίες θα εξηγήσουμε λεπτομερέστατα σε επόμενο κεφάλαιο. Κατά την διαδικασία handle του συγκεκριμένου event, ουσιαστικά ενημερώνεται η ίδια η εφαρμογή για την λάθος τιμή που εισήχθη στο attribute της Οντότητας (που μπορεί να είναι η τιμή ενός TextBox), και κοκκινίζει το συγκεκριμένο πεδίο, ενώ εμφανίζει και τον τύπο του σφάλματος σε κατάλληλο μήνυμα. Το σφάλμα είναι ορισμένο εντός της μεθόδου Validate(), και μεταφέρεται στο dictionary, όταν ανιχνευθεί σφάλμα. Στην προκειμένη περίπτωση, το σφάλμα που επιθυμούμε να εμφανίζεται όταν δεν εισάγεται ο κωδικός ICD10, θα είναι «Δεν μπορεί ο κωδικός ICD10 να είναι κενός».

Το δεύτερο Interface (INotifyPropertyChanged) ενημερώνει ουσιαστικά την εφαρμογή ότι τροποποιήθηκε το value σε ένα attribute της κλάσης. Πρόκειται για ένα εξίσου σημαντικό interface, το οποίο, όταν εφαρμόζεται, εκτελεί την μέθοδο OnPropertyChanged() και ενεργοποιεί το αντίστοιχο event, που αναλύσαμε νωρίτερα.

Συμπερασματικά, η δημιουργία Οντοτήτων για την απεικόνιση των στοιχείων της ΒΔ, αποτελεί μία πολύ καλή πρακτική στην σχεδίαση μίας εφαρμογής σαν αυτής, που εξετάζεται στην παρούσα Διατριβή. Η εφαρμογή γνωρίζει ανα πάσα στιγμή την κατάσταση των δεδομένων που επεξεργάζεται ο χρήστης, από την στιγμή που αυτά ανακτήθηκαν από την ΒΔ. Συνεπώς, υπάρχει η δυνατότητα εκτέλεσης ελέγχου validation σε δεύτερο επίπεδο καθώς ο κύριος έλεγχος γίνεται εντός της ΒΔ, μέσω triggers, όπως είδαμε σε προηγούμενο κεφάλαιο.

StoreDB.cs

Όπως αναφέρθηκε παραπάνω, η κλάση η οποία ευθύνεται για την ικανοποίηση των λειτουργιών της εφαρμογής, που αφορούν στην επικοινωνία με την ΒΔ, είναι η StoreDB.cs. Παρακάτω, παρατίθενται τα fields και ο constructor της υπόψη κλάσης:

```
namespace StoreDatabase
{
    public class StoreDB
    {
        //FIELDS
        private static StoreDB storeDBclinic = new StoreDB();
        public static StoreDB StoreDBclinic
        {
            get { return storeDBclinic; }
        }

        private eyecrystalEntities context;
        public eyecrystalEntities Context
        {
            get
            {
                return context;
            }
            set
            {
                context = value;
            }
```

```
        }

        public StoreDB()
        {
            //context initialisation
            context = new eyecrystalEntities();

        }
. . .
```

Αρχικά παρατηρούμε την δημιουργία της static μεταβλητής storeDBclinic, η οποία αποτελεί attribute εφαρμόζοντας τα απαιτούμενα getter/setter. Ειδικά στην setter μέθοδο, η μεταβλητή δημιουργεί καινούριο αντικείμενο της ίδιας της κλάσης StoreDB. Με αυτό τον τρόπο καταφέρνουμε να έχουμε πάντα ένα μόνο στιγμιότυπο της κλάσης StoreDB, οποτεδήποτε γίνεται αναφορά σε μέθοδο αυτής. Με αυτό τον τρόπο διευκολύνεται το έργο του garbage collector, με δεδομένο ότι η συγκεκριμένη κλάση είναι η περισσότερο προσπελάσιμη από οποιαδήποτε άλλη κλάση, στο δεύτερο και κύριο Project του Solution.

Στην συνέχεια, υπάρχει το attribute context, που είναι τύπου eyecrystalEntites και το οποίο αρχικοποιείται μέσα στον constructor. Συνεπώς, ένας δεύτερος λόγος που εφαρμόζεται η προαναφερθείσα πολιτική δημιουργίας ενός μόνο στιγμιότυπου της κλάσης StoreDB κάθε φορά, είναι για να υπάρχει πάντα ο έλεγχος του αριθμού των context, που δημιουργούνται για την προσπέλαση των Οντοτήτων EF, καθώς δεν είναι επιθυμητό να υπάρχουν context, τα οποία ολοκλήρωσαν την δουλειά τους αλλά παραμένουν στην μνήμη χωρίς να υπάρχει λόγος, ιδιαίτερα αν συνυπολογιστεί και το μεγάλο μέγεθος που καταλαμβάνουν στην RAM.

Έπειτα, ακολουθεί ένα μεγάλος αριθμός μεθόδων, τις οποίες καλεί κυρίως το crystaleye Project. Ενδεικτικά, αναφέρονται οι ακόλουθες:

Μέθοδος getPatients():

```
        ...
 public List<patientDTO> getPatients()
        {
            List<patientDTO> patientList = new List<patientDTO>();

            foreach (patient pat in context.patient)
            {
                patientList.Add(new patientDTO(pat.AMKA, pat.AMA, pat.Name,
pat.Surname, pat.Date_Birth, pat.Date_Patient_Created, pat.Email));
            }


            return patientList;
        }
        ...
```

Η μέθοδος getPatients() επιστρέφει μία λίστα αποτελούμενη από αντικείμενα DTO (Data Transfer Object), που αντιστοιχούν στους ασθενείς της κλινικής. Η εν λόγω μέθοδος, όπως θα δούμε στην συνέχεια, καλείται από την εφαρμογή για να εισάγει τους ασθενείς σε μία dropdown list, προκειμένου ο χρήστης να μπορεί να επιλέξει έναν ασθενή και να εκτελέσει διάφορες ενέργειες. Ο λόγος που δεν επιστρέφονται ολόκληρες οι οντότητες ασθενών, είναι διότι η μνήμη θα γέμιζε με άχρηστη ουσιαστικά πληροφορία. Για παράδειγμα, σε ένα dropdown list, απαιτείται να εμφανίζονται μόνο το ονοματεπώνυμο και το ΑΜΚΑ ενός ασθενή και όχι τα στοιχεία επικοινωνίας και δευτερεύοντα

δεδομένα του ασθενή. Στην περίπτωση που γινόταν φόρτωση ολόκληρων των Οντοτήτων, θα παρατηρούνταν καθυστέρηση στην δημιουργία του dropdown list, παρόλο που θα εμφανίζονταν πάλι μόνο τα βασικά στοιχεία του ασθενή (ονοματεπώνυμο-ΑΜΚΑ). Η χρήση των Data Transfer Objects αποτελεί βασικό δομικό στοιχείο στον κώδικα ανάπτυξης μίας εφαρμογής, ο οποίος εκμεταλλεύεται τα οφέλη του Entity Framework.

Μέθοδος getClinicExamsByYear():

```
        ...
//statistics clinical examination by year
         public Collection<StatsByYear> getClinicExamsByYear()
         {
             Collection<StatsByYear> lst_stats = new Collection<StatsByYear>();

             eyecrystalEntities tempcontext = new eyecrystalEntities();
             using (tempcontext)
             {
                 clinical_examination start =
(clinical_examination)tempcontext.clinical_examination.OrderBy(u =>
u.Examination_Date).FirstOrDefault();
                 clinical_examination end =
(clinical_examination)tempcontext.clinical_examination.OrderByDescending(u =>
u.Examination_Date).FirstOrDefault();

                 int startyear = start.Examination_Date.Year;
                 int endyear = end.Examination_Date.Year;

                 //get examination dates
                 List<DateTime> dates_clinic_Exams = (from a in
context.clinical_examination where a.Examination_Date.Year >= startyear &&
a.Examination_Date.Year <= endyear select a.Examination_Date).ToList();

                 //group examination dates
                 var examspermonth = (from row in dates_clinic_Exams
                                      group row by row.Year into g
                                      select new { Period = g.Key,
TotalPeriodCount = g.Count() }).AsEnumerable();

                 //create object for each record
                 foreach (var record in examspermonth)
                 {
                     StatsByYear examstat = new StatsByYear(record.Period,
record.TotalPeriodCount);
                     lst_stats.Add(examstat);
                 }
             }

             return lst_stats;
         }
        ...
```

Η μέθοδος getClinicExamsByYear() επιστρέφει ένα Collection τύπου StatsByYear. Ουσιαστικά, επιστρέφει το σύνολο των κλινικών εξετάσεων που πραγματοποιήθηκαν στην οφθαλμολογική κλινική ανα έτος. Η κλάση StatsByYear ορίζεται μέσα στον φάκελο Statistics και

αποτελεί μία custom Οντότητα με attributes το έτος και τον αριθμό των εξετάσεων, όπως φαίνεται παρακάτω:

```
namespace StoreDatabase.Statistics
{
    public class StatsByYear
    {
         //FIELDS
        private int _year;
        public int _Year
        {
            get { return _year; }
            set
            {
                _year = value;
            }
        }

        private int clinical_sum;
        public int Clinical_sum
        {
            get { return clinical_sum; }
            set
            {
                clinical_sum = value;
            }
        }

        public StatsByYear(int year, int count)
        {
            //string anaparastasi year
            _Year = year;

            //counts
            clinical_sum = count;

        }
    }
}
```

Με την δημιουργία μίας ξεχωριστής Οντότητας εκμεταλλευόμαστε τα οφέλη της αντικειμενοστραφής προσέγγισης που εφαρμόζει το EF, κυρίως στην διαχείριση των δεδομένων μέσα στον κώδικα μας. Επισημαίνεται ότι, τόσο οι κλάσεις-Οντότητες εντός του φακέλου DTO όσο και εντός του φακέλου Statistics, δεν είναι προϊόν παραγωγής του EF. Συνεπώς, με την χρήση κατάλληλων βρόγχων επανάληψης και την δημιουργία τέτοιων οντοτήτων, η μέθοδος getClinicExamsByYear() εξάγει το Collection με τις επιθυμητές πληροφορίες.

Μέθοδος getDoctors():

```
        ...
  public ObservableCollection<personel> getDoctors()
        {
            eyecrystalEntities temp_context = new eyecrystalEntities();
            personel personel = new personel();
            List<doctor> docs = new List<doctor>();
            ObservableCollection<personel> doctors = new
ObservableCollection<personel>();
```

```
            using (temp_context)
            {
                docs = temp_context.doctor.ToList();
                foreach(doctor doc in docs){

doctors.Add(temp_context.personel.Where(u=>u.AMKA==doc.AMKA).FirstOrDefault());
                }

            }
            return doctors;
        }

        ...
```

Με την μέθοδο getDoctors() επιστρέφεται ένα ObservableCollection με αντικείμενα τύπου personel. Ουσιαστικά, επιστρέφεται το σύνολο των ιατρών της κλινικής, με την μορφή των Οντοτήτων, τις οποίες δημιούργησε το EF. Αρχικά, επιστρέφονται οι Οντότητες τύπου doctor και αποθηκεύονται προσωρινά σε μία λίστα με όνομα μεταβλητής docs. Έπειτα, με την χρήση της γλώσσας παραγωγής ερωτημάτων LINQ, προσθέτουμε σε ένα άλλο ObservableCollection την οντότητα personel, η οποία αντιστοιχεί στον ιατρό. Αυτό γίνεται διότι είναι γνωστή η τιμή του attribute AMKA για κάθε οντότητα doctor και συνεπώς καταλήγουμε στην ανάκτηση της οντότητας personel που αντιστοιχεί στο ίδιο AMKA.

Σε αυτό το σημείο, κρίνεται σκόπιμη η αναφορά στις δυνατότητες που παρέχει η κλάση ObservableCollection σε σχέση με την αντίστοιχη της List. Η ObservableCollection λειτουργεί με τον ίδιο τρόπο με την κλασική Collection με την διαφορά ότι εφαρμόζει τα εξής 2 interfaces:

α) INotifyCollectionChanged

β) INotifyPropertyChanged

Με αυτά τα δύο interfaces, η συγκεκριμένη κλάση είναι χρήσιμη όταν είναι επιθυμητή η γνωστοποίηση στην εφαρμογή, είτε εάν ένα στοιχείο της συλλογής έχει αλλάξει, είτε όταν έχει αλλάξει η ίδια η συλλογή με προσθήκες/αφαιρέσεις στοιχείων. Η συγκεκριμένη κλάση συνεργάζεται άψογα με την γλώσσα προγραμματισμού WPF, όπως θα δούμε στην συνέχεια, και ειδικότερα με την XAML. Στην προκειμένη περίπτωση, το πρόγραμμα εμφανίζει, για παράδειγμα, σε ένα datagrid το σύνολο των ιατρών με τα χαρακτηριστικά τους. Υποθετικά, στην συνέχεια, πραγματοποιείται αλλαγή σε ένα από αυτά, είτε από διαφορετικό χρήστη είτε από το ίδιο το πρόγραμμα. Αυτομάτως, η συλλογή ενεργοποιεί κατάλληλο event, το οποίο μπορεί να το διαχειριστεί η εφαρμογή, όπως για παράδειγμα να εμφανίσει κατάλληλο ενημερωτικό μήνυμα προς τον χρήστη.

Μέθοδος updateItem():

```
//Update Specific entity/context
        public Exception updateItem(eyecrystalEntities specificCOntext)
        {
            //sigxronismos tis vasis dedomenwn me to antikeimeno context
            try
            {
                using (var tran = new TransactionScope())
                {
                    specificCOntext.SaveChanges();

                    tran.Complete();
                    return null;
                }
            }
            catch (Exception ex)
            {
```

```
                if (ex is DbEntityValidationException)
                {
                    DbEntityValidationException e =
(DbEntityValidationException)ex;
                    foreach (var eve in e.EntityValidationErrors)
                    {
                        Console.WriteLine("Entity of type \"{0}\" in state
\"{1}\" has the following validation errors:",
                            eve.Entry.Entity.GetType().Name, eve.Entry.State);
                        foreach (var ve in eve.ValidationErrors)
                        {
                            Console.WriteLine("- Property: \"{0}\", Error:
\"{1}\"",
                                ve.PropertyName, ve.ErrorMessage);
                        }
                    }
                    throw;
                }
                else if (ex is DbUpdateConcurrencyException)
                {

                }


                else if (ex.InnerException != null)
                {
                    MessageBox.Show("Εμφανίστηκε το παρακάτω σφάλμα: " +
ex.InnerException.InnerException.Message);
                }


                else
                {
                    MessageBox.Show("Εμφανίστηκε το παρακάτω σφάλμα: " +
ex.Message);
                }

                return ex;

            }

        }
```

Με την μέθοδο updateItem(), εκτελείται αποθήκευση του context. Ουσιαστικά, λαμβάνει χώρα συγχρονισμός των αλλαγών που έχουν πραγματοποιηθεί στις Οντότητες του context με την Βάση Δεδομένων. Η διαδικασία αυτή συμβαίνει εντός ενός try-catch block, όπου γίνεται επιστροφή του Exception σε περίπτωση που κάτι δεν πήγε όπως προβλεπόταν, ή null εάν η αποθήκευση έγινε επιτυχώς.

Στην συνέχεια, αξίζει να γίνει αναφορά στην κλάση Transactionscope. Η υπόψη κλάση εξασφαλίζει ότι η διαδικασία αποθήκευσης του context είναι συντονισμένη με μια σειρά μηνυμάτων. Εάν υπάρξει σφάλμα, κατά την διάρκεια ενός Update, η διαδικασία αποθήκευσης επαναλαμβάνεται μέχρι δύο φορές, ενώ εάν η διαδικασία αποθήκευσης επιτύχει, τότε οι αλλαγές του context γίνονται αποδεκτές. Επίσης, η μέθοδος saveChanges() είναι εκείνη η εντολή του EF, η οποία αποθηκεύει τις αλλαγές που έχουν υποστεί οι Οντότητες του context μέσα στην Βάση Δεδομένων. Τέλος, με την μέθοδο Complete(), αποδεσμεύεται το Transaction.

BackUP.cs

Η κλάση BackUp.cs περιλαμβάνει τον απαιτούμενο κώδικα για την εκτέλεση των διαδικασιών backup και restore της ΒΔ, όπως φαίνεται παρακάτω:

```
namespace StoreDatabase
{
    public class BackUp
    {
        //create a global StreamWriter
        private StreamWriter OutputStream;
        private List<string> backup_file_names;
        private string backup_directory;
        private string filePath;
        private Process proc;

        public BackUp()
        {

            //initialize the StreamWriter...
            backup_directory = @"backup\";

            //check if directory exists and create it
            if (!Directory.Exists(backup_directory))
            {
                Directory.CreateDirectory(backup_directory);
            }

            //init file_name list for current backup files
            backup_file_names = new List<string>();
        }

        public void CreateBackup(string servername, string username, string
password, string databasename)
        {
            //initialize backup parameters...
            string mysqldumpPath = @"C:\Program Files\MySQL\MySQL Server
5.7\bin\mysqldump.exe";
            string host = servername;  //servername
            string user = username;       //username
            string pswd = password;      //password
            string dbnm = databasename; //database name

            string cmd = String.Format("-h{0} -u{1} -p{2} --opt --databases {3} -
-default_character_set=utf8 --routines", host, user, pswd, dbnm);

            filePath = decide_file_name("crystaleye_");

            OutputStream = new StreamWriter(filePath, false,
Encoding.GetEncoding(1252));

            //create start info...
            ProcessStartInfo startInfo = new ProcessStartInfo();
            startInfo.FileName = mysqldumpPath;
            startInfo.Arguments = cmd;

            startInfo.RedirectStandardError = true;
            startInfo.RedirectStandardInput = false;
            startInfo.RedirectStandardOutput = true; //we need to redirect the
standard input to our StreamWriter
```

```
            startInfo.UseShellExecute = false;
            startInfo.CreateNoWindow = true;
            startInfo.ErrorDialog = false;

            //create the process...
            proc = new Process();
            proc.StartInfo = startInfo;

            //here we set the event handler for the output...
            proc.OutputDataReceived += new
DataReceivedEventHandler(OnDataReceived);

            //start the process...
            proc.Start();

            //tell the process to begin reading asynchronous output...
            proc.BeginOutputReadLine();

            //wait for the process to finish...
            proc.WaitForExit();

            //flush the stream and close, now our dump is created...
            OutputStream.Flush();
            OutputStream.Close();

            //close the process...
            proc.Close();

            //MessageBox.Show("Το αρχείο Back Up της Βάσης Δεδομένων
δημιουργήθηκε επιτυχώς!", "Επιτυχές Back Up", MessageBoxButton.OK,
MessageBoxImage.Information);

        }

        public void RestoreBackup(string servername, string username, string
password, string databasename, string backupfilepath)
        {
            //initialize backup parameters...
            string mysqlPath = @"C:\Program Files\MySQL\MySQL Server
5.7\bin\mysql.exe";
            string host = servername;  //servername
            string user = username;       //username
            string pswd = password;      //password
            string dbnm = databasename; //database name

            string cmd = String.Format(" -h{0} -u{1} -p{2} --default-character-
set=utf8 -D {3}", host, user, pswd, dbnm);
            //start
            var process = Process.Start(new ProcessStartInfo
            {
                FileName = mysqlPath,
                Arguments =
                String.Format(
                    "-C -B --host={0} --user={1} --password={2} --database={3} -e
\"\\. {4}\"",
                    servername, user, password, databasename, backupfilepath),
                ErrorDialog = false,
                CreateNoWindow = true,
                UseShellExecute = false,
```

```
                RedirectStandardError = true,
                RedirectStandardInput = true,
                RedirectStandardOutput = true,
                WorkingDirectory = Environment.CurrentDirectory,
            }
            );

            process.OutputDataReceived += (o, e) =>
Console.Out.WriteLine(e.Data);
            process.ErrorDataReceived += (o, e) =>
Console.Error.WriteLine(e.Data);
            process.Start();
            process.BeginErrorReadLine();
            process.BeginOutputReadLine();
            process.StandardInput.Close();
            process.WaitForExit();

          //end
          //MessageBox.Show("Το αρχείο Back Up της Βάσης Δεδομένων δημιουργήθηκε
επιτυχώς!", "Επιτυχές Back Up", MessageBoxButton.OK,
MessageBoxImage.Information);

        }

        //we need an event handler to fill our StreamWriter...
        private void OnDataReceived(object Sender,
System.Diagnostics.DataReceivedEventArgs e)
        {
            if (e.Data != null)
            {
                OutputStream.WriteLine(e.Data);
            }
        }

        private void load_backup_filenames()
        {
            //search in backup directory and find all the filenames
            string[] file_names = Directory.GetFiles(backup_directory, "*.sql");

            //make a list of those names
            foreach (var item in file_names)
                backup_file_names.Add(item);
        }

        private string decide_file_name(string basename)
        {
            load_backup_filenames();

            string base_name = @"backup\" + basename +
DateTime.Today.ToShortDateString() + "_";
            int file_counter = backup_file_names.Count;

            return base_name + (file_counter + 1).ToString() + ".sql";
        }
    }
}
```

Οι βασικές μεθόδοι που χρησιμοποιεί η υπόψη κλάση είναι η CreateBackup() και η RestoreBackup().

Η CreateBackup(), αξιοποιεί την κλάση ProcessStartinfo, η οποία επιτρέπει την εκτέλεση εφαρμογών του συστήματος και όχι μόνο, με την κατάλληλη εισαγωγή παραμέτρων. Στην προκειμένη περίπτωση, εκτελούμε το utility της MySQL mysqldump, το οποίο επιτρέπει την δημιουργία backup της ΒΔ. Επιπρόσθετα, απαιτείται η χρήση της κλάσης StreamWriter, για να αποθηκευτεί προσωρινά το output του αντικειμένου ProcessStartInfo με την χρήση της μεθόδου Flush() και του event OnDataReceived().

Αντίστοιχα η μέθοδος RestoreBackup(), λειτουργεί με έναν παρόμοιο τρόπο, με την διαφορά ότι εκμεταλλεύεται το utility mysql.

Τέλος, επισημαίνεται ότι τα αρχεία backup αποθηκεύονται σε ειδικό φάκελο (με όνομα backup), ο οποίος δημιουργείται στην εφαρμογή, και ο καθορισμός των ονομάτων των αρχείων περιέχει την ημερομηνία και τον αύξων αριθμό εκτέλεσης του backup. Στον constructor της κλάσης backup, γίνεται έλεγχος εάν υπάρχει ο υπόψη φάκελος και σε περίπτωση που δεν υπάρχει, αυτός δημιουργείται.

Εν κατακλείδι, το Project StoreDatabase έχει ιδιαίτερη βαρύτητα για την αξιόπιστη λειτουργία του ΠΣ. Αν και το Project crystaleye περιλαμβάνει τις περισσότερες λειτουργίες της εφαρμογής, εντούτοις εξαρτάται από το Project StoreDatabase και για αυτό τον λόγο κατά την φόρτωση της εφαρμογής στην μνήμη του συστήματος, πρώτα φορτώνεται το Project StoreDatabase και έπειτα το crystaleye. Για αυτό τον λόγο κρίθηκε σκόπιμο ως σημείο έναρξης της ανάλυσης του κώδικα να είναι το Project StoreDatabase.

4.2 Crystaleye Project

4.2.1 Windows Presentation Foundation

Η Windows Presentation Foundation (WPF), αποτελεί την τελευταία γενιά UI Framework της Microsoft. Αποτελεί μέρος του .Net Framework από την έκδοση 3.0 και πάνω. Συνδυάζει δυνατότητες ανάπτυξης σε UI, γραφικά 2D, γραφικά 3D, έγγραφα και πολυμέσα σε ένα μόνο Framework. Η μηχανή, στην οποία στηρίζεται είναι vector-based rendering και χρησιμοποιεί την επιτάχυνση hardware των καινούριων καρτών γραφικών. Αυτό επιτρέπει στο UI να είναι πιο γρήγορο και ανεξάρτητο από την ανάλυση οθόνης του συστήματος. Τα κυριότερα χαρακτηριστικά της WPF φαίνονται στο παρακάτω σχήμα:

Εικόνα 20. Χαρακτηριστικά του WPF Framework

Στην WPF διαχωρίζεται η εμφάνιση του UI από την συμπεριφορά που θα έχει αυτό. Η εμφάνιση του UI καθορίζεται στην Extensible Application Markup Language (XAML), ενώ η συμπεριφορά του υλοποιείται με την βοήθεια μίας γλώσσας προγραμματισμού, όπως C# ή Visual Basic. Τα δύο αυτά βασικά δομικά συστατικά της WPF, συνδέονται μέσω διαδικασιών databinding, events και commands. Ο διαχωρισμός της εμφάνισης και της συμπεριφοράς επιφέρει τα ακόλουθα οφέλη:

α) Η εμφάνιση και η συμπεριφορά του UI είναι σθεναρά συνδεδεμένες μεταξύ τους,

β) οι Σχεδιαστές και Προγραμματιστές μπορούν να δουλεύουν ταυτόχρονα σε διαφορετικά μοντέλα,

γ) τα εργαλεία σχεδίασης Γραφικών μπορούν να επεξεργαστούν σε απλά XML αρχεία και δεν απαιτούν επιπλέον κώδικα.

Πλεονεκτήματα WPF

Τα κυριότερα πλεονεκτήματα της WPF είναι τα εξής:

α) Device Independent Pixel. Για την εμφάνιση ενός παραθύρου (window), έχει ιδιαίτερη σημασία ο υπολογισμός των Dots per Pixel (DPI) στην οθόνη. Συνήθως, αυτό εξαρτάται από την συσκευή απεικόνισης και το λειτουργικό σύστημα, στο οποίο εκτελείται η εφαρμογή. Μία κλασσική Windows Forms εφαρμογή χρησιμοποιεί Pixel-based προσέγγιση και για αυτό τον λόγο, μία αλλαγή στις ρυθμίσεις DPI του συστήματος, απαιτεί κάθε στοιχείο του παραθύρου να τροποποιήσει το μέγεθος και την εμφάνιση του. Η WPF, ωστόσο, ξεπερνάει αυτό το εμπόδιο και είναι ανεξάρτητη των DPI ρυθμίσεων της οθόνης.

β) Υποστήριξη για Γραφικά και Animations. Οι εφαρμογές WPF απεικονίζονται στην οθόνη με την βοήθεια του DirectX. Συνεπώς, παρέχεται τεράστια built-in υποστήριξη σε θέματα γραφικών και animations. Υπάρχουν ξεχωριστές κλάσεις που ευθύνονται για τα εφέ των animations και γραφικών.

γ) Επαναπροσδιορισμός Styles και Control Templates. Η τεχνική εφαρμογής Styles αποτελείται από ένα σύνολο συστατικών, τα οποία καθορίζουν το πώς θα φαίνονται τα διάφορα controls στην οθόνη. Στις παραδοσιακές Windows Forms εφαρμογές, τα στυλ εμφάνισης των στοιχείων είναι άρρηκτα συνδεδεμένα με τα ίδια τα στοιχεία (πχ χρώμα, μέγεθος κλπ). Στην WPF τα στυλ εμφάνισης είναι πλήρως ανεξάρτητα από το UIElement. Ένα στυλ εμφάνισης μπορεί να χρησιμοποιηθεί σε πολλά στοιχεία του παραθύρου. Επιπλέον, η WPF εισάγει τα Templates, τα οποία έχουν την δυνατότητα να τροποποιήσουν τα controls (UIElemens) ριζικά.

δ) Προσέγγιση με βάση τα Resources. Στις κλασσικές Windows Forms εφαρμογές, εάν απαιτούνταν να εφαρμοστεί ένα συγκεκριμένο χρώμα σε 1000 κουμπιά, θα έπρεπε να δημιουργηθούν 1000 διαφορετικά αντικείμενα τύπου Color. Ωστόσο, στην WPF υπάρχει η δυνατότητα αποθήκευσης των στυλ, controls, animations κ.α. με την μορφή των resources. Με αυτό τον τρόπο, υφίσταται ένα resource, το οποίο μπορεί να γίνει reference σε πολλές φόρμες και να επηρεάζει την λειτουργία των στοιχείων της φόρμας.

ε) Καινούριο Σύστημα Properties και Binding. Κάθε στοιχείο στην WPF καθορίζει ένα μεγάλο αριθμό ιδιοτήτων, γνωστών ως dependency properties, οι οποίες προσθέτουν πολλές δυνατότητες στις κοινές ιδιότητες, στις οποίες θα γίνει αναφορά στην συνέχεια.

XAML

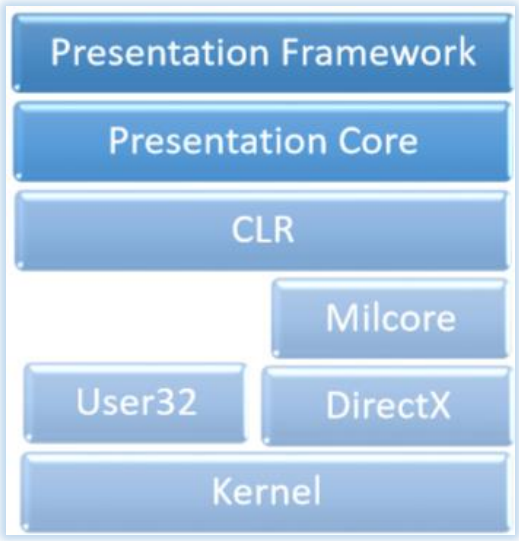
Η XAML είναι μία XML-based γλώσσα, η οποία χρησιμοποιείται για τον ορισμό των χαρακτηριστικών των κλάσεων. Συνεπώς, δίνεται η δυνατότητα ορισμού μεταβλητών και των ιδιοτήτων μίας κλάσης με απευθείας χρήση στην εφαρμογή. Η XAML χρησιμοποιείται γενικά για τον καθορισμό του layout του

UI και των στοιχείων που περιλαμβάνει. Δεν μπορεί να χρησιμοποιηθεί για την τροποποίηση της συμπεριφοράς ενός προγράμματος.

Τα στοιχεία που περιέχει ένα αρχείο XAML απεικονίζονται απευθείας στα αντικείμενα CLR (Common Language Runtime), ενώ τα attributes του απεικονίζονται ως ιδιότητες και events αυτών των αντικειμένων.

Αρχιτεκτονική WPF

Τα κυριότερα δομικά στοιχεία του WPF Framework φαίνονται στην παρακάτω εικόνα:



Εικόνα 21. Αρχιτεκτονική WPF

Από τα παραπάνω στοιχεία, ξεχωρίζουν τα εξής:

α) Presentation Framework,

β) Presentation One και

γ) Milcore.

Τα πρώτα δύο είναι γραμμένα σε διαχειρίσιμο κώδικα (managed code). Το Milcore αποτελεί μέρος μη διαχειρίσμου κώδικα, ο οποίος επιτρέπει την άρρηκτη σύνδεση με το DirectX. Το CLR προσδίδει μεγαλύτερη παραγωγικότητα στην διαδικασία ανάπτυξης, προσφέροντας υπηρεσίες όπως διαχείριση μνήμης, χειρισμός σφαλμάτων κ.α.

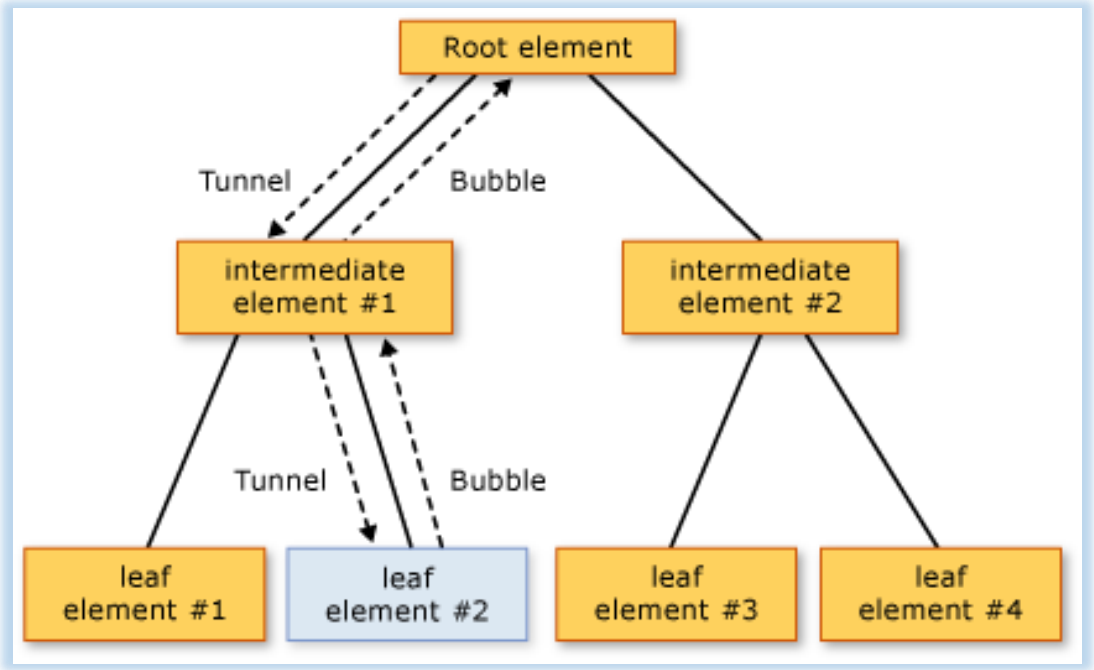
Αξίζει στην συνέχεια να γίνει αναφορά στα routed events, που εφαρμόζει το WPF Framework. Ένα routed event είναι το συμβάν, το οποίο μπορεί να επικαλεστεί πολλούς listeners σε μία ιεραρχική δενδροειδή δομή στοιχείων και όχι απλά το αντικείμενο, που ενεργοποίησε το event. Τα routed events εφαρμόζουν τις εξής 3 στρατηγικές:

α) Direct Event

β) Bubbling Event

γ) Tunnel Event

Στο παρακάτω σχήμα φαίνονται οι περιπτώσεις bubbling και tunnel event:



Εικόνα 22. WPF Routed Events

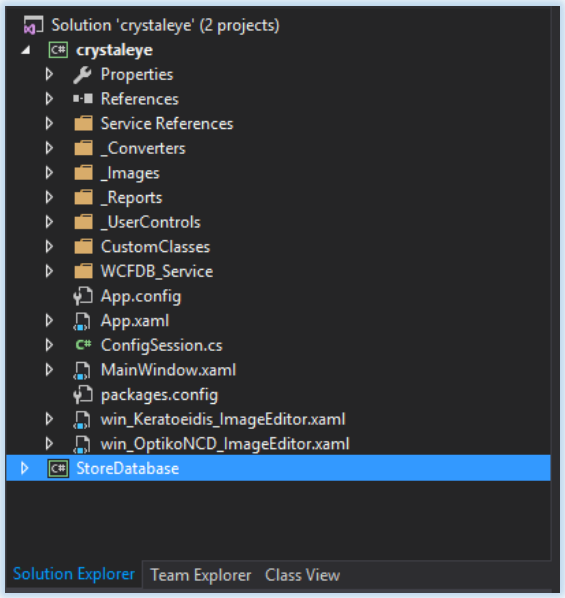
Η πρώτη κατηγορία event είναι η παραδοσιακή φύση των event, που είναι γνωστή από τα Windows Forms. Ενδεικτικό παράδειγμα είναι το MouseEnter event.

Στην δεύτερη περίπτωση, το συμβάν αρχίζει από το στοιχείο που το ενεργοποίησε και ταξιδεύει προς τα πάνω στην δενδροειδή δομή των UIElements μέχρι να φτάσει την κορυφή, η οποία είναι συνήθως το ίδιο το παράθυρο. Αντίθετα, στην τρίτη περίπτωση, το συμβάν ταξιδεύει προς τα κάτω με σημείο έναρξης το πρώτο στοιχείο της δενδροειδούς διάταξης, μέχρι να βρει το στοιχείο (child element), το οποίο προκάλεσε το συμβάν. Στον κώδικα που περιλαμβάνει η παρούσα ΔΕ, έχει χρησιμοποιηθεί αρκετές φορές η δεύτερη περίπτωση event, και πραγματοποιείται εκτενής ανάλυση, σχετικά με την πορεία που ακολουθεί το συμβάν.

Στην συνέχεια, ακολουθεί εκτενής ανάλυση του Project crystaleye, στο οποίο θα συζητηθούν αρκετές από τις προαναφερθείσες, και όχι μόνο, τεχνολογίες που εφαρμόζει το WPF Framework.

4.2.2 Ανάλυση crystaleye Project - Εισαγωγή

Το crystaleye Project, περιλαμβάνει τον κώδικα, ο οποίος ευθύνεται για την υλοποίηση των λειτουργιών, οι οποίες καθοριστήκαν στις απαιτήσεις, όπως και για την εμφάνιση UI της εφαρμογής. Το Framework, που χρησιμοποιήθηκε είναι το WPF, όπως αναφέρθηκε. Περιλαμβάνει αρχεία και φακέλους, τα οποία φαίνονται στην παρακάτω εικόνα, και που θα αναλυθούν στην συνέχεια:



Εικόνα 23. Δομή του Eyecrystal Project

Όπως φαίνεται παραπάνω, το eyecrystal Project αποτελείται από αρχεία-κλάσεις με επέκταση .cs, configuration files, φακέλους, η χρήση των οποίων τμηματοποιεί λογικά τον κώδικα, αρχεία με επέκταση xaml κ.α. Συγκεκριμένα περιλαμβάνονται τα εξής:

α) Properties, References: Περιλαμβάνουν τις ιδιότητες του Project, και τις απαιτούμενες αναφορές σε άλλα project αντίστοιχα.

β) Φάκελος Service References: Περιλαμβάνει τα references, που αφορούν το WebService της σύνδεσης με την ΒΔ του ΕΟΟΠΥ και θα γίνει εκτενής ανάλυση σε επόμενο κεφάλαιο.

γ) _Converters: Περιλαμβάνει τις κλάσεις, οι οποίες λειτουργούν ως converters για τον τρόπο που εμφανίζονται δεδομένα στο UI, και που αποθηκεύονται στον κώδικα αντίστοιχα.

δ) _Images: Περιλαμβάνει αρχεία εικόνων, τα οποία χρησιμοποιούνται στην εμφάνιση UI της εφαρμογής.

ε) _Reports: Περιλαμβάνει αρχεία τύπου rdlc και cs και αφορούν τις αναφορές που μπορεί να εμφανίζει η εφαρμογή για διάφορες κατηγορίες δεδομένων καθώς και την εκτύπωση τους.

στ) _UserControls: Αποτελούν βασικό δομικό στοιχείο στην ανάπτυξη μίας WPF εφαρμογής. Είναι αρχεία τύπου xaml, συνοδευόμενα από το αρχείο cs, το οποίο φιλοξενεί τον κώδικα με τις λειτουργίες του UserControl. Επισημαίνεται, ότι έχει ακολουθηθεί η τυποποίηση, όπου κάθε αρχείο Usercontrol αρχίζει με τους χαρακτήρες «UC_». Συνεπώς, όπου αναφέρεται αρχείο ονόματος, το οποίο να αρχίζει με την παραπάνω αλληλουχία χαρακτήρων, αυτομάτως αναφερόμαστε σε ένα Usercontrol.

ζ) CustomClasses: Περιλαμβάνει όλες τις κλάσεις, οι οποίες περιλαμβάνουν εξειδικευμένες λειτουργίες, όπως αποστολή email.

η) WCFDB_Service: Αφορά το WebService, η ανάλυση του οποίου θα γίνει σε επόμενο κεφάλαιο.

θ) App.config, App.xaml, packages.config: Αρχεία στα οποία αποθηκεύονται πληροφορίες και ιδιότητες του Project. Επίσης το αρχείο app.xaml, χρησιμοποιείται για αποθήκευση resources, styles κ.α. που αξιοποιούνται από πολλά στοιχεία (UIElement) της εφαρμογής.

ι) ConfigSession.cs: Κλάση η οποία χειρίζεται βασικά στοιχεία και διαδικασίες, που απευθύνονται στον συνδεδεμένο χρήστη.

ια) MainWindow.xaml: Το αρχικό και κύριο παράθυρο της εφαρμογής με τον κώδικα XAML και C#.

ιβ) win_Keratoeidis_ImageEditor.xaml: Παράθυρο, το οποίο εμφανίζει μία εικόνα εξέτασης, η οποία μπορεί να υποστεί επεξεργασία από τον ιατρό.

ιγ) win_OptikoNCD_ImageEditor.xaml: Παρόμοια περίπτωση με την παραπάνω, με την διαφορά ότι πρόκειται για διαφορετική εξέταση και συνεπώς χρησιμοποιείται διαφορετική εικόνα.

Στην συνέχεια, θα γίνει λεπτομερής αναφορά σε συγκεκριμένα αρχεία κώδικα, προκειμένου να γίνει κατανοητή η χρήση των τεχνολογιών που χρησιμοποιεί το WPF Framework. Επίσης θα γίνει αναφορά στα αρχεία, τα οποία ικανοποιούν τις βασικές απαιτήσεις που καθορίστηκαν ενώ σε κάθε επιμέρους ανάλυση θα εμφανίζεται και το κομμάτι του κώδικα προς εξέταση και όχι ο συνολικός κώδικας που περιέχει το αρχείο, για λόγους εξοικονόμησης χώρου. Αρχικά θα εξεταστεί το αρχείο XAML, το οποίο ευθύνεται για τον καθορισμό εμφάνισης του παραθύρου της εφαρμογής, προκειμένου να γνωρίσουμε τις ιδιαιτερότητες του κώδικα XAML. Ωστόσο, επειδή η εμφάνιση της εφαρμογής έχει δευτερεύοντα ρόλο στην παρούσα ΔΕ , ακολουθώντας μία απλή και λιτή προσέγγιση, δεν θα επεκταθούμε στην ανάλυση των αρχείων XAML για τα υπόλοιπα παράθυρα και Usercontrols. Αντιθέτως, θα γίνει πλήρης ανάλυση των code-behind αρχείων, για να εξεταστεί ο τρόπος με τον οποίο καθορίστηκε η συμπεριφορά και η λειτουργικότητα της εφαρμογής.

4.2.3 Υλοποίηση Βασικού Παραθύρου του ΠΣ

Το αρχείο MainWindow.xaml ευθύνεται για τον καθορισμό εμφάνισης UI του αρχικού παραθύρου της εφαρμογής. Παρακάτω παρατίθεται ο καθορισμός του attribute Window με τις παραμέτρους, που απαιτούνται:

MainWindows.xaml

```
<Window x:Class="crystaleye.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:r="clr-
namespace:Microsoft.Windows.Controls.Ribbon;assembly=RibbonControlsLibrary"
        xmlns:local="clr-namespace:crystaleye"
        mc:Ignorable="d"
        Title="CrystalEye @2016" MinWidth="1500" MinHeight="768"
ResizeMode="CanResize" WindowState="Maximized"
        Name="mainwindow" Loaded="mainwindow_Loaded">
...
```

Το στοιχείο Window, δημιουργεί ένα αντικείμενο τύπου Window, με τις παραμέτρους που περιλαμβάνει. Με το πρώτο attribute x:Class, ορίζεται το αρχείο-κλάση, το οποίο περιέχει τον κώδικα για τον καθορισμό των λειτουργιών του παραθύρου και που είναι το MainWindow.xaml.cs. Έπειτα, ορίζονται properties, που αποτελούν ουσιαστικά αναφορές σε packages, δίνοντας την δυνατότητα αξιοποίησης των κλάσεων και στοιχείων που περιέχουν. Επίσης, γίνεται αναφορά και στο namespace, στο οποίο περιλαμβάνεται το Window (crystaleye) και ακολουθούν ιδιότητες, όπως Title, MinWidth, WindowState κ.α. Τέλος, γίνεται αναφορά και στα built-in event που χρησιμοποιεί το παράθυρο και που στην προκειμένη περίπτωση είναι το event Loaded.

Στην συνέχεια, ορίζονται τα Resources του παραθύρου. Όπως έχει ήδη αναφερθεί, τα resources περιλαμβάνουν στυλ, templates κτλ,, τα οποία είναι δυνατό να αξιοποιηθούν από πολλά

στοιχεία. Συνεπώς, το πρώτο στοιχείο που περιλαμβάνει το αντικείμενο Window θα είναι το Resources, προκειμένου να χρησιμοποιηθεί από όλα τα υπόλοιπα στοιχεία ελέγχου (Textboxes, Buttons κτλ), που περιέχει το στοιχείο Window. Παρακάτω παρατίθεται ο κώδικας του στοιχείου Resources:

```
  <Window.Resources>
        <SolidColorBrush x:Key="RedBrush" Color="#0788b2"/>
        <SolidColorBrush x:Key="SolidBorderBrush" Color="#888" />
        <SolidColorBrush x:Key="GreenBrush" Color="Red" />
        <SolidColorBrush x:Key="DisabledBackgroundBrush" Color="#EEE" />
        <SolidColorBrush x:Key="DisabledBorderBrush" Color="#AAA" />
        <SolidColorBrush x:Key="DisabledForegroundBrush" Color="#888" />

        <!--TabItem Style-->
        <Style TargetType="{x:Type TabItem}">
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="{x:Type TabItem}">
                        <Grid>
                            <Border
                                Name="Border"
                                Margin="0,0,-4,0"
                                Background="{StaticResource RedBrush}"
                                BorderBrush="{StaticResource  SolidBorderBrush}"
                                BorderThickness="1,1,1,1"
                                CornerRadius="2,12,0,0" >
                                <ContentPresenter x:Name="ContentSite"
                                    VerticalAlignment="Center"
                                    HorizontalAlignment="Center"
                                    ContentSource="Header"
                                    Margin="12,2,12,2"
                                    RecognizesAccessKey="True"/>
                            </Border>
                        </Grid>
                        <ControlTemplate.Triggers>
                            <Trigger Property="IsSelected" Value="True">
                                <Setter Property="Panel.ZIndex" Value="100" />
                                <Setter TargetName="Border" Property="Background"
Value="{StaticResource GreenBrush}" />
                                <Setter TargetName="Border"
Property="BorderThickness" Value="1,1,1,0" />
                            </Trigger>
                            <Trigger Property="IsEnabled" Value="False">
                                <Setter TargetName="Border" Property="Background"
Value="{StaticResource DisabledBackgroundBrush}" />
                                <Setter TargetName="Border"
Property="BorderBrush" Value="{StaticResource DisabledBorderBrush}" />
                                <Setter Property="Foreground"
Value="{StaticResource DisabledForegroundBrush}" />
                            </Trigger>
                        </ControlTemplate.Triggers>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </Window.Resources>
```

Στον παραπάνω κώδικα, εύκολα διακρίνονται κάποια χαρακτηριστικά της γλώσσας XAML. Για παράδειγμα, ο ορισμός του αντικειμένου SolidColorBrush, συνοδεύεται από attributes, όπως x:key, που έχει σχέση με το όνομα που δίνεται στο αντικείμενο, για να μπορεί να γίνει αναφορά σε

αυτό από άλλα αντικείμενα του αρχείου, ενώ ορίζεται και το χρώμα (Color), το οποίο αντιπροσωπεύει το συγκεκριμένο αντικείμενο. Ουσιαστικά, δημιουργήθηκε ένα αντικείμενο τύπου SolidColorbrush, το οποίο αντιπροσωπεύει ένα χρώμα και που μπορεί να τεθεί ως value σε κάποιο άλλο attribute ενός στοιχείου, το οποίο όμως να δέχεται ως τιμή χρώματα.

Στην συνέχεια, ορίζεται ένα στυλ το οποίο θα εφαρμόζεται σε όλα τα αντικείμενα τύπου tabitem, εντός του αρχείου. Σε αντίθεση με την προηγούμενη περίπτωση, στην οποία είχε οριστεί και το όνομα του αντικειμένου μέσω του attribute x:key, εδώ δεν απαιτείται καθώς το WPF είναι αρκετά έξυπνο ώστε να καταλάβει ότι το συγκεκριμένο στυλ θα το εφαρμόσει σε όλα τα αντικείμενα τύπου TabItem, εντός του αρχείου. Το συγκεκριμένο στυλ καθορίζει εξολοκλήρου το Template του στοιχείου Tabitem, μέσω του <Setter Property="Template">, το οποίο αντικαθιστά την default εμφάνιση.

Τέλος, ορίζονται και triggers για το υπόψη στυλ, τα οποία διαφοροποιούν ιδιότητες των στοιχείων που καθορίζουν την εμφάνιση του Tabitem, στις περιπτώσεις όπου ενεργοποιηθεί ένα συμβάν, για παράδειγμα όταν είναι επιλεγμένο το Tabitem (<Trigger Property="IsSelected" Value="True">) ή όταν είναι ενεργοποιημένο (<Trigger Property="IsEnabled" Value="False">). Τα triggers είναι εξαιρετικά χρήσιμα, καθώς δεν χρειάζεται στο code-behind αρχείο να συνταχθεί επιπλέον κώδικας, ο οποίος να χειρίζεται αυτά τα events.

Ο κώδικας XAML ακολουθεί μία δενδροειδή ιεραρχική δομή των στοιχείων που εμφανίζονται. Συνεπώς, τα στοιχεία που περιέχει, έχουν άλλα στοιχεία – γονείς (parent element) και στοιχεία – παιδιά (child elements). Το στοιχείο Window απαιτείται να έχει μόνο ένα child element, που στην προκειμένη περίπτωση είναι το Grid, όπως φαίνεται παρακάτω:

```
...
<Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="auto"></RowDefinition>
            <RowDefinition Height="auto"></RowDefinition>
            <RowDefinition Height="*"></RowDefinition>
            <RowDefinition Height="auto"></RowDefinition>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="auto"></ColumnDefinition>
            <ColumnDefinition Width="*"></ColumnDefinition>
            <ColumnDefinition Width="auto"></ColumnDefinition>
        </Grid.ColumnDefinitions>
        <Grid.Background>
            <ImageBrush ImageSource="_Images\mainBackground.jpg"
Stretch="UniformToFill"></ImageBrush>
         </Grid.Background>

...
```

Αρχικά, ορίζονται ο αριθμός των στηλών και των γραμμών που θα έχει το grid. Επίσης, καθορίζονται attributes, όπως Width και Height για συγκεκριμένες στήλες και γραμμές. Γενικά, στις εφαρμογές WPF αποφεύγεται ο καθορισμός συγκεκριμένης τιμής σε αποστάσεις καθώς έτσι δεν αξιοποιείται η δυνατότητα της WPF εφαρμογής να εμφανίζει μία fluid εμφάνιση. Για αυτό τον λόγο οι τιμές των attributes height και width είναι ίσες με την τιμή auto.

Αφού πλέον έχει καθοριστεί η δομή της εμφάνισης του παραθύρου, είμαστε σε θέση να προσθέτουμε στοιχεία καθορίζοντας την συγκεκριμένη στήλη και γραμμή, που είναι επιθυμητό αυτά να τοποθετηθούν. Πλέον όμως, υπάρχει η δυνατότητα να καρατομηθεί περαιτέρω ο χώρος-κελί που ορίζει ο εκάστοτε συνδυασμός γραμμής-στήλης, είτε με την χρήση επιπλέον Grid είτε με την χρήση άλλων παρεμφερών στοιχείων, που έχουν ως στόχο την τοποθέτηση των στοιχείων στον χώρο (π.χ StackPanel, DockPanel κ.α.)

Για παράδειγμα, με τον παρακάτω κώδικα ορίζουμε την δομή που θα έχει η δεξιά sidebar του παραθύρου:

```
...
<!--RIGHT SIDEBAR-->
        <ScrollViewer Grid.Column="2" Grid.Row="2"
VerticalScrollBarVisibility="Auto" Margin="10">
            <Border  BorderBrush="Black" BorderThickness="1" MaxWidth="300"
CornerRadius="5">
                <DockPanel MinWidth="300" Height="auto">

                    <!--ELEMENT ACTIONS-->
                    <ContentControl DockPanel.Dock="Bottom"
Name="cnt_elementActions">

                    </ContentControl>

                    <!--SYSTEM DIALOGUES-->
                    <ContentControl DockPanel.Dock="Top"
Name="cnt_SystemDialogues">

                    </ContentControl>

                    <!--VALIDATION STATUS-->
                    <ContentControl DockPanel.Dock="Top"
Name="cnt_validationErrors">

                    </ContentControl>

                    <!--HINTS-->
                    <ContentControl DockPanel.Dock="Top" Name="cnt_hints">

                    </ContentControl>
                </DockPanel>
                <Border.Background>
                    <ImageBrush ImageSource="_Images\RigthSidebar.jpg"
Stretch="UniformToFill"></ImageBrush>

                </Border.Background>
            </Border>
        </ScrollViewer>
...
```

Τα σημαντικά στοιχεία XAML του παραπάνω κώδικα είναι τα contentcontrol. Έχει οριστεί η τοποθεσία για το καθένα και λειτουργούν ως placeholder για άλλα στοιχεία XAML, που θα φιλοξενήσουν. Ο καθορισμός του περιεχομένου για κάθε contentcontrol θα γίνεται αυτόματα και κατά περίπτωση στο code-behind αρχείο του παραθύρου, και για αυτό τον λόγο απαιτείται ο ορισμός ονόματος για κάθε contentcontrol, μέσω του attribute Name. Παρόμοια contentcontrols βρίσκονται και σε άλλα σημεία στον κώδικα XAML του αρχείου, όπως το contentcontrol που θα φιλοξενεί το κυρίως περιεχόμενο της εφαρμογής.

Τέλος, ένα στοιχείο το οποίο αξίζει να αναφερθεί είναι το Ribbon. Ο ρόλος του είναι να εμφανίζει μία μπάρα μενού, η οποία φιλοξενεί διάφορα κουμπιά, όπως φαίνεται παρακάτω:

```
...
<RibbonTab Header="Γενικές Καταστάσεις" Visibility="Hidden"
Name="rbn_tab_general_data">
                <RibbonGroup Header="Γενικά">
                    <RibbonButton Name="rbbtn_Appointments_Show"  Label="Εμφάνιση
Ραντεβού" LargeImageSource="_Images/appointment.jpg" ToolTipTitle="Εμφάνιση όλων
των ραντεβού" ToolTipDescription="Μετάβαση στην φόρμα για να επεξεργαστείτε τα
στοιχεία των ραντεβού σας." Click="RibbonButton_Click"></RibbonButton>
```

```
                    <RibbonButton Name="rbbtn_Statistics_Show"  Label="Εμφάνιση
Στατιστικών" LargeImageSource="_Images/stats_icon.png" ToolTipTitle="Εμφάνιση
Στατιστικών" ToolTipDescription="Δείτε στατιστικά δεδομένα σχετικά με την κλινική
σας και την λειτουργία της." Click="rbbtn_Statistics_Show_Click"></RibbonButton>

                </RibbonGroup>
                <RibbonGroup Header="Αναφορές">
                    <RibbonButton Name="rbbtn_Report_Patients_General"
Label="Αναφορά Ασθενών" LargeImageSource="_Images/report_icon.png"
ToolTipTitle="Εμφάνιση της αναφοράς των ασθενών" ToolTipDescription="Μετάβαση
στην αναφορά με την συνοπτική κατάσταση ασθενών."
Click="rbbtn_Report_Patients_General_Click"></RibbonButton>
                    <RibbonButton Name="rbbtn_Report_Personel_General"
Label="Αναφορά Προσωπικού" LargeImageSource="_Images/report_icon.png"
ToolTipTitle="Εμφάνιση της αναφοράς του προσωπικού" ToolTipDescription="Μετάβαση
στην αναφορά με την συνοπτική κατάσταση προσωπικού."
Click="rbbtn_Report_Personel_General_Click"></RibbonButton>

                </RibbonGroup>
</RibbonTab>
...
```

MainWindow.xaml.cs (Code-behind file)

Το αρχείο, που ευθύνεται για τον καθορισμό της συμπεριφοράς και των λειτουργιών του παραθύρου είναι το MainWindow.xaml.cs. Παρακάτω φαίνεται ο ορισμός της κλάσης MainWindow, τα fields και ο constructor:

```
...
public partial class MainWindow : Window
    {
        //FIELDS ***************************************************
        //CURRENT PATIENT
        public patient current_Patient;
        public string filter;

        //CONTRUCTOR*****************************************************************

        public MainWindow()
        {
            InitializeComponent();

            //combobox itemsource
            cmb_patient.ItemsSource = StoreDB.StoreDBclinic.getPatients();

            //Load first tab
            loadFIrstTab();

            this.DataContext = this;
        }
...
```

Αρχικά, είναι εμφανές ότι η κλάση είναι partial, το οποίο σημαίνει ότι κάπου αλλού στο Project υπάρχει εξίσου η ίδια κλάση, η οποία δημιουργήθηκε από το Visual Studio και που ορίζει ουσιαστικά ιδιότητες, που δεν απαιτείται συνήθως να τροποποιήσει ο χρήστης, χωρίς αυτό να είναι δεσμευτικό. Στην συνέχεια, ορίζεται η μεταβλητή τύπου patient, στην οποία θα αποθηκεύεται η οντότητα που αντιπροσωπεύει τον ενεργό ασθενή της εφαρμογής. Ο ενεργός ασθενής είναι ο επιλεγμένος ασθενής από τον χρήστη, για τον οποίο μπορεί να εκτελέσει διάφορες ενέργειες. Στον constructor, αρχικά,

φορτώνεται ένα combobox με τους ασθενείς της κλινικής και στην συνέχεια δημιουργείται το πρώτο tab, το οποίο φιλοξενεί πληροφορίες, που καλωσορίζουν τον χρήστη και τον προτρέπουν να συνδεθεί στην εφαρμογή με τα credentials του. Τέλος με την εντολή this.DataContext = this;, παρέχεται η δυνατότητα στο αρχείο XAML του παραθύρου να μπορεί να προσπελάσει τις μεταβλητές και τα στοιχεία της κλάσης, σε περίπτωση που χρειάζεται να τα αξιοποιήσει.

Στην συνέχεια, ορίζονται διάφορες μέθοδοι, οι οποίες κυρίως καθορίζουν την λειτουργία των κουμπιών του ribbon και που στις περισσότερες περιπτώσεις εισάγουν νέο tab με συγκεκριμένο περιεχόμενο, όπως φαίνεται παρακάτω στην περίπτωση του κουμπιού που αφορά την φόρμα BackUp:

```
...
//BACKUP FORM
        private void rbn_backup_Click(object sender, RoutedEventArgs e)
        {
            if (tab_Main.Items.Cast<TabItem>().Where(i => i.Name ==
"tb_back_up").FirstOrDefault() == null)
            {
                //create new usercontrol patient instance
                UC_BackUp uc_bk = new UC_BackUp();

                //create new tabitem
                TabItem item = new TabItem();
                item.Name = "tb_back_up";
                item.Header = "Back Up";

                //set new tabitem content the usercontrol
                item.Content = uc_bk;

                tab_Main.Items.Insert(tab_Main.Items.Count, item);
//add tabitem in tabcontrol
                tab_Main.SelectedItem = item;
            }
            else
            {
                TabItem item = tab_Main.Items.Cast<TabItem>().Where(i => i.Name
== "tb_back_up").FirstOrDefault();
                tab_Main.SelectedItem = item;
            }
        }
...
```

Αρχικά, ελέγχεται εάν υπάρχει tabitem με όνομα tb_back_up στο Tab control. Εάν υπάρχει σημαίνει ότι ο χρήστης έχει ήδη επισκεφτεί το συγκεκριμένο tabitem και δεν το έχει κλείσει. Συνεπώς, αποκτάει αυτομάτως focus και δεν δημιουργείται καινούριο. Σε διαφορετική περίπτωση, θα πρέπει να δημιουργηθεί καινούριο tabitem με το υπόψη όνομα. Σε ότι αφορά το περιεχόμενο του tabitem, δημιουργείται ένα αντικείμενο τύπου UC_BackUp, το οποίο είναι το UserControl με τις ενέργειες που αφορούν τις λειτουργίες backup-restore της εφαρμογής. Με την εντολή item.Content=uc_bk;, αυτόματα καθορίζεται το περιεχόμενο του tabitem, το οποίο θα είναι το αντικείμενο τύπου UC_BackUp. Στην συνέχεια, με την εντολή tab_Main.SelectedItem = item;, το συγκεκριμένο tabitem αποκτάει focus στο tab control και πλέον ο χρήστης μπορεί και το βλέπει.

Όπως φαίνεται παραπάνω, υπάρχει άρρηκτη σύνδεση μεταξύ του tabitem και του περιεχομένου του. Ένα tabitem με συγκεκριμένο όνομα δεν μπορεί να έχει δύο διαφορετικά περιεχόμενα. Υπάρχουν πολλές στιγμές μέσα στον κώδικα, όπου γίνεται έλεγχος εάν υπάρχει «φορτωμένο» ένα συγκεκριμένο tabitem, δηλαδή μία συγκεκριμένη λειτουργία του συστήματος.

4.2.4 Υλοποίηση Login-Logout

Όπως αναφέρθηκε παραπάνω, το πρώτο tabitem που φορτώνει το παράθυρο είναι αυτό που καλωσορίζει τον χρήστη. Εκτός από αυτό όμως, φιλοξενεί και την φόρμα σύνδεσης του χρήστη με την εφαρμογή και για αυτό τον λόγο έχει ιδιαίτερη σημασία να εξετάσουμε την αλληλουχία των διαδικασιών, προκειμένου η εφαρμογή να δημιουργήσει ένα session για τον χρήστη. Το Usercontrol, που εμπλέκεται για αυτή την λειτουργία είναι το UC_default_main_guest.

Αρχικά θα εξετάσουμε τα fields και τον constructor της υπόψη κλάσης.

UC_default_main_guest.xaml.cs (Code-behind file)

```
...
public partial class UC_default_main_guest : UserControl
    {
        //FIELDS
        private MainWindow parentWindow;
        private UC_Hints hints;
        public UC_Element_Action elementActions;

        public UC_default_main_guest()
        {
            //init Hints list
            hints = new UC_Hints(this);

            //dimiourgia antikeiemenou actions kai energopoihsh montelou gia
patients
            elementActions = new UC_Element_Action(null, null, null);
            elementActions.hide_dialogue();

            InitializeComponent();
            //login form
            UC_Login loginform = new UC_Login();
            cnt_Login.Content = loginform;

            //arxikopoihsh references sto mainwindow
            parentWindow = (MainWindow)Application.Current.MainWindow;

            //listener gia to custom event enimerwsis registered user
            loginform.registered_user_changed += new
UC_Login.InformLoginStatusEventHandler(register);
        }
...
```

Η πρώτη μεταβλητή που ορίζεται είναι τύπου MainWindow. Ο λόγος ύπαρξης της είναι να μπορεί το Usercontrol να αναφερθεί στα στοιχεία του κεντρικού παραθύρου της εφαρμογής, καθώς υπενθυμίζεται ότι στην ιεραρχική δενδροειδή δομή των στοιχείων XAML της εφαρμογής, τα Usercontrols που φιλοξενούνται στα tabitems, θα είναι πάντα παιδιά (child-element) του MainWindow. Στην συνέχεια, οι μεταβλητές τύπου UC_Hints και UC_Element_Action αφορούν Usercontrols, τα οποία θα φιλοξενούνται στην sidebar, για όση ώρα είναι focused το tabitem που περιέχει το Usercontrol UC_default_main_guest. Εάν επιλέξει διαφορετικό tabitem ο χρήστης, τότε αυτόματα αλλάζουν και τα δεδομένα που εμφανίζονται στο sidebar. Η διαδικασία αυτή υλοποιήθηκε με την χρήση του event UserControl_GotFocus().

Σε ότι αφορά τον constructor, αρχικά φορτώνεται το περιεχόμενο των hints και element actions, καθώς το περιεχόμενο των υπόψη Usercontrols προσαρμόζεται αναλόγως της λειτουργικότητας των Usercontrols με τα οποία συνδέονται. Στην συνέχεια, «φορτώνεται» το UC_Login, το οποίο περιέχει την φόρμα συμπλήρωσης των credentials του χρήστη. Έπειτα, υλοποιείται η αναφορά στο parent Window, που όπως αναφέραμε είναι το παράθυρο της εφαρμογής.

Τέλος, ενεργοποιείται ένας listener για το custom event της σύνδεσης του χρήστη. Η UC_Login θα ενεργοποιήσει ένα event, όπως θα δούμε παρακάτω, το οποίο θα πρέπει να γίνει handle. Επειδή το UC_Login αποτελεί child-element του UC_default_main_guest, τοποθετούμε έναν Listener για να γίνει το handle, με την βοήθεια των λειτουργιών bubbling (και tunneling), που προσφέρει το WPF Framework. Στην συνέχεια, παρατίθεται η μέθοδος που εκτελεί το υπόψη handle:

```
...
//methodos pou xrisimopoieitai apo ton event handler gia to login
        private void register(personel user, UC_Registered_User UC_reg, UC_Login
UC_log)
        {
            //REGISTER CODE
            ConfigSession.Current_user = user;  //gnwstopoiw sto programma ton
registered user

            if (StoreDB.StoreDBclinic.isDoctor(ConfigSession.Current_user.AMKA))
                ConfigSession.isdoctor = true;
            else
            {
                ConfigSession.isdoctor = false;
            }

            //dimiourgw listener gia to custom event logout
            UC_reg.registered_user_logout += new
UC_Registered_User.InformLogoutStatusEventHandler(logout);

            //afairw ton listener gia na akouei login
            UC_log.registered_user_changed -= new
UC_Login.InformLoginStatusEventHandler(register);

            //refresh mainwindow combobox itemsource
            ComboBox tab_control =
(ComboBox)parentWindow.FindName("cmb_patient");
            tab_control.ItemsSource = StoreDB.StoreDBclinic.getPatients();

            //klisi methodou gia tropopoihsh tou UI eksaitias tou login
            UIoperations.UI_login(parentWindow.tab_Main,
ConfigSession.Current_user,UC_reg);
        }
...
```

Το συγκεκριμένο event περιλαμβάνει τον συνδεδεμένο χρήστη ως οντότητα τύπου personel, το Usercontrol, από το οποίο ενεργοποιήθηκε το event και τέλος, μία αναφορά στο αντικείμενο UC_Login που είναι ήδη φορτωμένο. Έτσι, αρχικά, αποθηκεύεται η οντότητα του χρήστη σε μία static μεταβλητή στην κλάση ConfigSession. Πλέον, σε οποιοδήποτε σημείο του κώδικα σε όλη την εφαρμογή, δίνεται η δυνατότητα αναφοράς στον συνδεδεμένο χρήστη. Στην συνέχεια, μέσω ενός if statement και με την βοήθεια του StoreDatabase Project, η εφαρμογή γνωρίζει εάν ο χρήστης είναι γιατρός ή όχι. Αμέσως μετά προστίθεται ένας listener για να «ακούσει» την περίπτωση που ο χρήστης εκτελέσει logout. Επίσης αφαιρείται ο listener, ο οποίος «άκουγε» την περίπτωση που ο χρήστης εκτελέσει Login, καθώς υπάρχει ήδη συνδεδεμένος χρήστης στην εφαρμογή. Τέλος, ακολουθούν τροποποιήσεις στην εμφάνιση του παραθύρου της εφαρμογής, οι οποίες προσαρμόζονται πλέον στην νέα κατάσταση στην οποία περιήλθε η εφαρμογή. Συγκεκριμένα, το περιεχόμενο του πρώτου tabitem αλλάζει και εμφανίζει το UC_default_main, το οποίο εμφανίζει πληροφορίες προσαρμοσμένες στον συνδεδεμένο χρήστη.

Στην συνέχεια, ορίζεται ο handler για την περίπτωση logout του χρήστη, στην οποία απενεργοποιείται ο listener για logout και ενεργοποιείται ξανά ο listener για το event login του χρήστη. Επίσης, αφαιρούνται οι τιμές που έχει λάβει η μεταβλητή του ενεργού χρήστη στην κλάση Configsession και αναπροσαρμόζεται το UI, προκειμένου να μπορεί να δεχθεί ξανά Login από τον χρήστη.

Για να ολοκληρωθεί η ανάλυση της διαδικασίας login του χρήση, θα πρέπει να γίνει αναφορά στην UC_Login, η οποία περιλαμβάνει όλους τους ελέγχους των credentials που εισήγαγε ο χρήστης.

UC_Login.xaml.cs (Code-behind file)

Παρακάτω παρατίθεται το τμήμα του κώδικα του υπόψη αρχείου που αφορά τις μεταβλητές που ορίζονται και τον constructor:

```
...
public partial class UC_Login : UserControl
    {
        //FIELDS
        private validate_login login = new validate_login();     //dimiourgia
antikeimenou validate_login kai anathesi ws datacontext tou usercontrol
        private personel registered_user;                        //o xristis tou
programmatos gia to session

        //Custom Event Definition that informs parent Window about the registered
user
        public delegate void InformLoginStatusEventHandler(personel
registereduser,UC_Registered_User UC, UC_Login UC_log);    //delegate gia xrisi
event
        public event InformLoginStatusEventHandler registered_user_changed;
//event gia na enimerwnei ta parent containers gia tin allagi tou login_status

        public void RaiseMyEvent(personel args_registered_user,
UC_Registered_User UC_reg, UC_Login UC_log)
        {
            if (registered_user_changed != null)
            {
                registered_user_changed(args_registered_user, UC_reg, UC_log);
            }
        }

        //CONSTRUCTOR
        public UC_Login()
        {
            InitializeComponent();

            //grid datacontext set
            grd_login.DataContext= login;

        }
...
```

Αρχικά δημιουργούμε ένα αντικείμενο της custom Οντότητας validate_login, η οποία αποθηκεύει τις τιμές username και password του χρήστη. Στην συνέχεια δημιουργούμε ένα custom event, το οποίο θα ενημερώνει την εφαρμογή για την κατάσταση σύνδεσης του χρήστη. Ουσιαστικά, πρόκειται για το event, το οποίο γίνεται handle από το UC_default_main_guest.xaml, που είδαμε προηγουμένως. Η δημιουργία του event υλοποιείται με την χρήση μίας μεταβλητής τύπου delegate. Τα delegates είναι μεταβλητές τύπου reference, που αναφέρονται σε μία μέθοδο. Η ενεργοποίηση

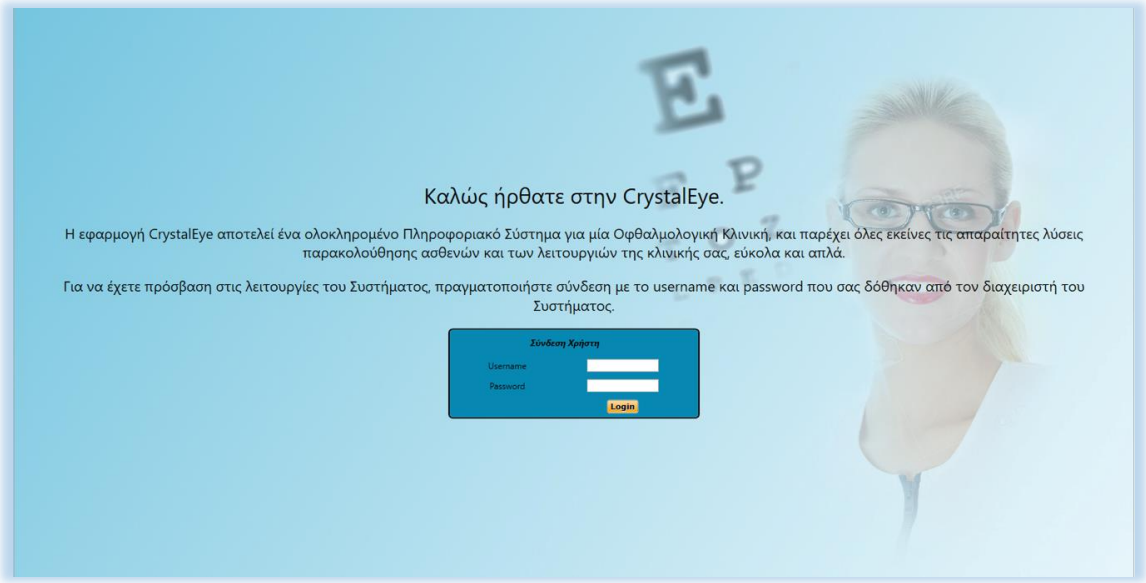
του event πραγματοποιείται με την μέθοδο RaiseEvent() και η οποία μέσω της διαδικασίας bubbling φτάνει στο UC_default_main_guest, προκειμένου να γίνει handle.

Στην συνέχεια, εξετάζουμε το τμήμα του κώδικα που εκτελείται όταν ο χρήστης πατάει το κουμπί login:

```
...
private void btn_Login_Click(object sender, RoutedEventArgs e)
        {
            if(txt_user.Text.Length==0 || txt_pass.SecurePassword.Length == 0)
            {
                txt_Errors.Text = "Εισάγετε τα στοιχεία σας.";
                return;
            }

            registered_user = StoreDB.StoreDBclinic.login_user(txt_user.Text,
txt_pass.SecurePassword);

            if (registered_user == null)
            {
                txt_Errors.Text = "Ο συνδυασμός των στοιχείων που εισάγατε δεν
αντιστοιχεί σε κάποιον χρήστη. Προσπαθήστε ξανά.";
            }
            else
            {
                //dimiourgia antikeimenou usercontrol UC_Registered_User
                UC_Registered_User UC_registered_user = new
UC_Registered_User(registered_user);

                //clear tou parontos usercontrol kai fortwsi tou neou pou tha
emfanizei ton registered_user
                ((ContentControl)this.Parent).Content = UC_registered_user;

                RaiseMyEvent(registered_user, UC_registered_user,this);
            }
        }
...
```

Μεγάλη σημασία στον παραπάνω κώδικα έχει η κλήση της μεθόδου StoreDB.StoreDBclinic.login_user(txt_user.Text, txt_pass.SecurePassword);, που υπάρχει στο StoreDatabase Project. Εάν υπάρξει οντότητα με το username και Password που εισήγαγε ο χρήστης, τότε η μέθοδος επιστρέφει την οντότητα, αλλιώς επιστρέφει null. Εάν το αποτέλεσμα είναι null, τότε εμφανίζεται κατάλληλο μήνυμα. Διαφορετικά, ενεργοποιείται το event για να γνωστοποιήσει στην εφαρμογή την αλλαγή κατάστασης σύνδεσης του χρήστη. Επιπλέον, δημιουργείται ένα αντικείμενο τύπου UC_Registered_User, που περιέχει το ονοματεπώνυμο του συνδεδεμένου χρήστη και τα κουμπιά εμφάνισης προφίλ και αποσύνδεσης. Το υπόψη Usercontrol φιλοξενείται στο χώρο πάνω-αριστερά του παραθύρου και αντικαθιστά το περιεχόμενο του UC_Login.

Παρακάτω παρατίθενται screenshots για την εισαγωγή credentials του χρήστη και του τροποποιημένου UI, εφόσον συνδεθεί:



Εικόνα 24. Εμφάνιση Φόρμας Login

Εικόνα 25. Εμφάνιση τροποποίησης του UI ενός συνδεδεμένου χρήστη

Έπειτα, θα ακολουθήσει ανάλυση του αρχείου UC_patient.xaml.cs, το οποίο ευθύνεται για την επεξεργασία του Ιατρικού Φακέλου του Ασθενούς.

4.2.5 Υλοποίηση Εμφάνισης Εποπτικής Εικόνας Κλινικής

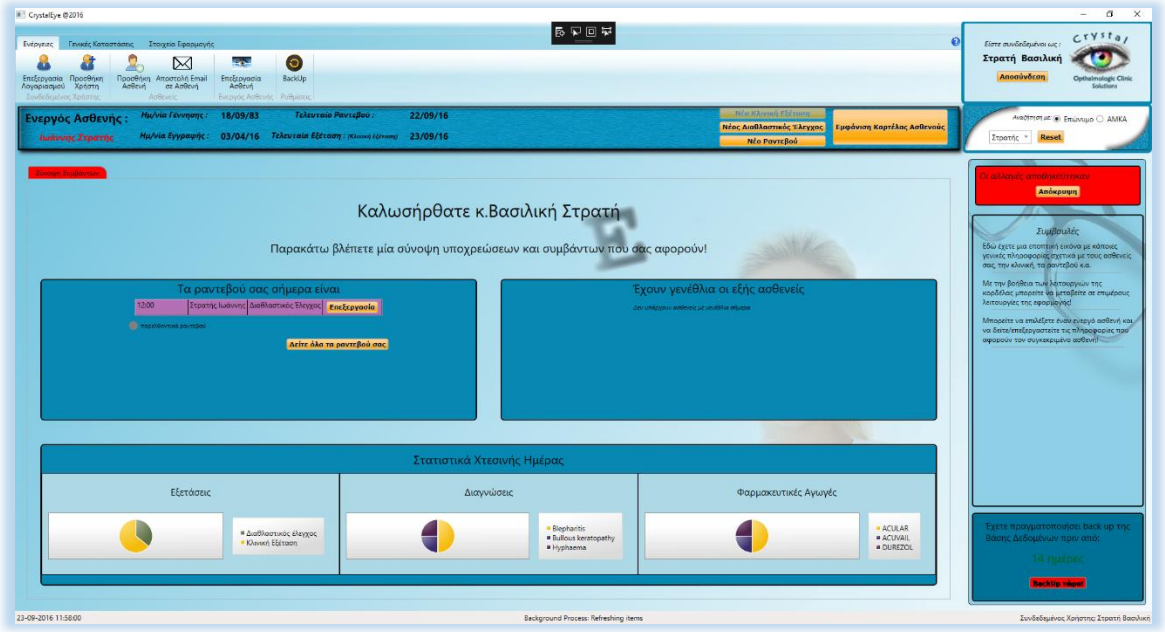
Στην συνέχεια, και εφόσον ο χρήστης έχει επιτυχώς συνδεθεί με την εφαρμογή, δημιουργείται καινούριο tabitem, το οποίο εμφανίζει μια συνολική εικόνα των ραντεβού του ιατρού σε περίπτωση που ο χρήστης είναι ιατρός ή εμφάνιση όλων των ραντεβού της κλινικής σε περίπτωση που ο χρήστης ανήκει στην γραμματεία. Εκτός από τα ραντεβού, εμφανίζονται οι χρήστες, οι οποίοι έχουν γενέθλια εκείνη την ημέρα και τέλος εμφανίζονται στατιστικά στοιχεία, αναφορικά με εξετάσεις, διαγνώσεις και φαρμακευτικές αγωγές, που έλαβαν χώρα την προηγούμενη ημέρα. Το Usercontrol που περιέχει τον κώδικα υλοποίησης όλων των παραπάνω είναι το UC_default_main.xaml.cs.

Επειδή ο κώδικας του υπόψη αρχείου είναι αρκετά μεγάλος, θα αρκεστούμε στην ανάλυση της λειτουργίας, κατά την οποία, μέσω ενός thread, εκτελείται μία διαρκής ανανέωση στο background της εφαρμογής, όπου επικαιροποιείται η λίστα με τα ραντεβού. Συγκεκριμένα, το σενάριο βασίζεται στην εκδήλωση επιθυμίας ενός ασθενή για πραγματοποίηση ραντεβού την συγκεκριμένη ημέρα, μέσω του portal της κλινικής. Συνεπώς, αυτές οι επιθυμίες πρέπει να εμφανίζονται απευθείας στο σύστημα και έπειτα ο χρήστης να τις εγκρίνει ή να μην τις εγκρίνει. Ο κώδικας που υλοποιεί τα παραπάνω είναι ο εξής:

. . .

```
private void Grid_Loaded(object sender, RoutedEventArgs e)
        {

            //  Thread to opoio ananewnei kathe x deuterolepta tin kentriki
selida
            dispatcherTimer = new DispatcherTimer();
            dispatcherTimer.Tick += new EventHandler(dispatcherTimer_Tick);
            dispatcherTimer.Interval = new TimeSpan(0, 1, 0);
            dispatcherTimer.Start();

            //reference sto parent tabcontrol
            TabItem parenttabitem = ((TabItem)this.Parent);
            TabControl parenttab = ((TabControl)parenttabitem.Parent);

            //dimiourgia antikeiemenou actions kai energopoihsh montelou gia
patients
            elementActions = new UC_Element_Action(null, null, parenttabitem);
            elementActions.backup_actions();

        }
//thread gia refresh items
        private void dispatcherTimer_Tick(object sender, EventArgs e)
        {
            get_today_appointments();

            dispatcherTimer2 = new DispatcherTimer();
            dispatcherTimer2.Tick += new EventHandler(show_dispatcher_message);
            dispatcherTimer2.Interval = new TimeSpan(0, 0, 5);
            dispatcherTimer2.Start();
            txt_Message_Statusbar =
(TextBlock)parentwindow.FindName("txt_Status_Bar_Middle");
            txt_Message_Statusbar.Text = "Background Process: Refreshing items";
        }
...
```

Αρχικά, με την φόρτωση του κεντρικού grid στοιχείου του Usercontrol, ενεργοποιείται ένα DispatherTimer αντικείμενο, το οποίο εκτελεί έναν αλγόριθμο κάθε ένα λεπτό. Ο handler λοιπόν του event, μέσω της μεθόδου get_today_appointments() ενημερώνει συνεχώς την εφαρμογή με τα ραντεβού για εκείνη την ημέρα. Πιθανά νέα ραντεβού θα προστεθούν στην ήδη υπάρχουσα λίστα. Στην συνέχεια, δημιουργείται ένα δεύτερο αντικείμενο DispatcherTimer, το οποίο για 5 δευτερόλεπτα εμφανίζει κατάλληλο μήνυμα στο status bar της εφαρμογής, που ενημερώνει τον χρήστη ότι γίνεται ανανέωση των δεδομένων στο background. Συνεπώς, με αυτό τον τρόπο πετυχαίνουμε να έχουμε την λίστα με τα ραντεβού επίκαιρη διαρκώς. Παρακάτω παρατίθεται ένα screenshot, όπου εμφανίζεται το υπόψη μήνυμα:



Εικόνα 26. Παράδειγμα Υλοποίησης Background Task

Στην παραπάνω εικόνα παρατηρούμε το σύνολο των στοιχείων, στα οποία έχει εποπτικό έλεγχο ο χρήστης. Για την συγκεκριμένη ημέρα ο χρήστης «Στρατή Βασιλική», που τυχαίνει και ανήκει στο ιατρικό προσωπικό της κλινικής, έχει μόνο ένα ραντεβού. Παρατηρούμε επίσης το μήνυμα που εμφανίστηκε στο status bar, που ενημερώνει τον χρήστη, ότι υφίσταται ανανέωση των δεδομένων στο background.

4.2.6 Ιατρικός Φάκελος Ασθενή

Στο αρχείο UC_patient.xaml.cs εκτελούνται όλες οι διαδικασίες επεξεργασίας του προφίλ του ασθενή. Το προφίλ περιλαμβάνει τόσο προσωπικά δεδομένα όσο και δεδομένα ιατρικού ενδιαφέροντος, τα οποία συνθέτουν τον ιατρικό του φάκελο. Ο Ιατρικός φάκελος περιλαμβάνει δεδομένα, τα οποία προέρχονται τόσο από τις εξετάσεις του ασθενή στην κλινική όσο και από πληροφορίες, όπως αλλεργίες, φαρμακευτικές αγωγές, ασθένειες κ.α. Επειδή ο κώδικας που περιλαμβάνει το εν λόγω αρχείο είναι αρκετά μεγάλος, θα αρκεστούμε στην ανάλυση των σημαντικότερων λειτουργιών, που εμφανίζονται σε αυτόν. Παρακάτω παρατίθεται ο κώδικας του constructor:

UC_patient.xaml.cs (Code-Behind file)

```
...
public partial class uc_patient : UserControl
    {
        ...

        //CONSTRUCTOR
        public uc_patient(patient patient)
        {
            current_patient = patient;

            //init Hints list
            hints = new UC_Hints(this);

            //dimiourgia antikeimenou dialogue errors (UC)
            dialogueErrors = new UC_ValidationErros();
```

```
            //dimiourgia antikeimenou dialogue (UC)
            dialogue = new UC_SystemDialogue();

            //MyErrors List init
            MyErrors = new ObservableCollection<string>();

            //reference sto parent window
            parentwindow = Application.Current.MainWindow;

            //dimiourgia context gia to sigkekrimeno Usercontrol
            context = StoreDB.StoreDBclinic.createContext();

            InitializeComponent();

            //thetw datacontext to patient instance tou kainouriou context me
vasi to primary key (AMKA)
            current_patient = context.patient.Find(patient.AMKA);
            main_grid.DataContext = current_patient;

            //thetw to itemsource gia ta diafora controls
            dtgrd_quick_history.ItemsSource =
StoreDB.StoreDBclinic.getQuickHistory(context, patient.AMKA);
            lst_agwgi.ItemsSource = StoreDB.StoreDBclinic.getAgwgi(context,
patient.AMKA);
            lst_allergies.ItemsSource =
StoreDB.StoreDBclinic.getAllergies(context, patient.AMKA);
            lst_astheneies.ItemsSource =
StoreDB.StoreDBclinic.getAstheneies(context, patient.AMKA);

            //handler gia to property changed tou current patient
            current_patient.PropertyChanged += Current_patient_PropertyChanged;
            metritis_photo = 0;

            if (current_patient.Photo != null && current_patient.Photo.Length !=
0)
            {
                //tropopoihsh UI
                btn_add_photo.Visibility = Visibility.Collapsed;
                btn_delete_photo.Visibility = Visibility.Visible;
            }
            else
            {
                //tropopoihsh UI
                btn_add_photo.Visibility = Visibility.Visible;
                btn_delete_photo.Visibility = Visibility.Collapsed;
            }

            //ean o sindedemenos xristis den einai iatros den mporei na dei
klinikes eksetaseis kai diathlastikous elegxous
            if (!ConfigSession.isdoctor)
            {
                cmb_eidos_eksetasis.Items.Remove(cmitm_de);
                cmb_eidos_eksetasis.Items.Remove(cbitm_ce);
            }
        }
...
```

Αρχικά, ορίζουμε καινούρια αντικείμενα τα οποία θα συνοδεύουν το UC_patient, στην sidebar. Αυτά είναι τα UC_Hints, UC_ValidationErrors, UC_SystemDIalogue και ElementActions.

Έπειτα, δημιουργούμε καινούριο context, το οποίο χρησιμοποιείται για όλες τις οντότητες και τα queries, τα οποία ενδέχεται να ζητήσει ο χρήστης κατά την επεξεργασία της φόρμας. Στην συνέχεια, τίθεται το Datacontext του maingrid της φόρμας ίσο με την μεταβλητή current_patient. Αυτό επιτρέπει το grid να έχει πρόσβαση στις ιδιότητες της Οντότητας current_patient και να τις εμφανίζει στα διάφορα πεδία, όπως ημερομηνία γέννησης, όνομα κ.λ.π. Ακολούθως, συμπληρώνονται με δεδομένα οι 3 λίστες που υπάρχουν στο UI, και που αφορούν τις αλλεργίες, τις φαρμακευτικές αγωγές και τις ασθένειες του ασθενή. Επίσης, δημιουργείται ένας listener, ο οποίος ενεργεί σε περίπτωση που ανιχνευθεί αλλαγή στις ήδη υπάρχουσες τιμές, στα διάφορα πεδία της φόρμας. Εάν συμβεί κάτι τέτοιο, τότε ενεργοποιείται το κουμπί «Αποθήκευση», διαφορετικά παραμένει απενεργοποιημένο. Στην συνέχεια, τροποποιείται το UI στην περίπτωση που ο χρήστης έχει εισάγει προσωπική φωτογραφία του. Επισημαίνεται, ότι στον ασθενή δίνεται η δυνατότητα να εισάγει φωτογραφία του μέσω του Ιστότοπου. Τέλος, πραγματοποιείται εκ νέου τροποποίηση του UI σε συγκεκριμένα στοιχεία, σε περίπτωση που ο συνδεδεμένος χρήστης δεν είναι ιατρός.

Μία μέθοδος που βρίσκουμε συχνά με διάφορες παραλλαγές στον κώδικα της εφαρμογής είναι η Exit_Click και ενεργοποιείται όταν ο χρήστης κλείνει το ενεργό tabiitem:

```
...
private void Exit_Click(object sender, RoutedEventArgs e)
        {
            //reference sto parent tabcontrol
            TabItem parenttabitem = ((TabItem)this.Parent);
            TabControl parenttab = ((TabControl)parenttabitem.Parent);

            //diagrafi current index tou tabcontrol
            int index = parenttab.SelectedIndex;
            parenttab.Items.RemoveAt(index);

            //sinthikes katastrofis antikeimenou dialogwn sistimatos
            dialogue.hide_dialogue();
            dialogue = null;

            //sinthikes katastrofis antikeimenou dialogwn validation
            dialogueErrors.hide_dialogue();
            dialogueErrors = null;

            //katastrofi context
            StoreDB.StoreDBclinic.destroycontext(context);
        }
...
```

Αρχικά, πραγματοποιείται μία αναφορά στο parent tabitem και tab control που περιέχει το UC_patient. Έπειτα διαγράφεται το tabitem ενώ τα στοιχεία που συνοδεύουν το UC_patient στο sidebar τίθενται με null. Τέλος, καταστρέφουμε το context, που είχε δημιουργηθεί στον constructor διευκολύνοντας την εργασία του garbage collector.

Μία άλλη μέθοδος που έχει ενδιαφέρον είναι η btn_save_click(), όπου πραγματοποιείται αποθήκευση των αλλαγών που έχει εκτελέσει ο χρήστης:

```
...
private void btn_save_Click(object sender, RoutedEventArgs e)
        {
            //retrieve sender object
            Button button = (Button)e.OriginalSource;

            //emfanisi save dialogue sto sidebar
            if (button.Name == "btn_save")
            {
                this.IsEnabled = false;
```

```
                dialogue.ControlRaiseName = button.Name;
                dialogue.saveDB_dialogue("Να αποθηκευτούν οι αλλαγές?");
                return;
            }

            else if (button.Name == "btn_Cancel")
            {
                txt_allergy.Text = "";
                txt_astheneia.Text = "";
                txt_medication.Text = "";
                resetUC();
                this.IsEnabled = true;
            }

            else
            {
                txt_allergy.Text = "";
                txt_astheneia.Text = "";
                txt_medication.Text = "";
                this.IsEnabled = true;

                //kalw sinartisi gia apothikeusi kai enimerwsi dedomenwn stin
vasi.
                Exception ex = StoreDB.StoreDBclinic.updateItem(context);
                if ( ex != null)
                {
                    if(ex is DbUpdateConcurrencyException)
                    {
                        var objectContext =
((IObjectContextAdapter)context).ObjectContext;
                        objectContext.Refresh(RefreshMode.StoreWins,
current_patient);
                        btn_save.IsEnabled = false;
                        dialogue.validated_Data("Το αντικείμενο που επιχειρήτε να
ενημερώσετε, τροποποιήθηκε από κάποιον άλλο χρήστη, όση ώρα εσείς το
επεξεργάζεστε. Η φόρμα ενημερώθηκε με τα τρέχοντα στοιχεία του αντικειμένου.");
                    }
                    else
                    {
                        dialogue.validated_Data("Ελέγξτε ξανά την ορθότητα των
δεδομένων που έχετε εισάγει");
                    }
                }
                else
                {
                    btn_save.IsEnabled = false;

                    dialogue.validated_Data("Οι αλλαγές αποθηκεύτηκαν με
επιτυχία");
                }
            }
        }
...
```

Για την κατανόηση της φιλοσοφίας που εφαρμόζεται στην διαδικασία αποθήκευσης, υπενθυμίζεται ότι τα πλαίσια διαλόγου του συστήματος με τον χρήστη δεν προέρχονται από ξεχωριστό παράθυρο, όπως πιθανόν να έχουμε συνηθίσει, αλλά από το αντικείμενο τύπου SystemDialogue, το εμφανίζει το μήνυμα «Θέλετε να εκτελέσετε αποθήκευση των αλλαγών?». Συνεπώς εάν ο χρήστης πατήσει OK, η διαδικασία της αποθήκευσης συνεχίζεται, διαφορετικά

διακόπτεται. Συνεπώς, όταν εκτελείται το event του κουμπιού «Αποθήκευση», επιβάλλεται να ελεγχθεί εάν το όνομα του κουμπιού, που προκάλεσε την ενεργοποίηση του event, είναι το btn_Save (το κουμπί που ανήκει στην UC_patient) ή btn_Cancel (που προέρχεται από το UC_SystemDialogue). Έτσι αρχικά ο χρήστης πατάει «Αποθήκευση» και μέσω του if statement εμφανίζεται το μήνυμα στο sidebar. Έπειτα πατάει «OK» και εκτελείται ξανά η ίδια μέθοδος-event. Ωστόσο, αυτή την φορά, ικανοποιείται διαφορετική συνθήκη στο if-statement και συγκεκριμένα η else. Πλέον, η εφαρμογή γνωρίζει ότι ο χρήστης όντως θέλει να εκτελέσει αποθήκευση. Έτσι μέσω της μεθόδου updateitem(), που περιλαμβάνεται στην StoreDatabase, γίνεται απόπειρα συγχρονισμού των αλλαγών του context με την ΒΔ. Σε περίπτωση που κάτι δεν πάει καλά επιστρέφεται exception και εμφανίζεται μήνυμα σφάλματος στο sidebar. Ενδιαφέρον έχει η πρόβλεψη concurrency που έχει υλοποιηθεί μέσω του exception DbUpdateConcurrencyException. Η εφαρμογή γνωρίζει την στιγμή της αποθήκευσης εάν τα δεδομένα έχουν αλλάξει από άλλο χρήστη και εμφανίζει κατάλληλο μήνυμα, το οποίο προτρέπει την ανανέωση των δεδομένων της φόρμας και την εκ νέου επεξεργασία τους από τον χρήστη.

Στην συνέχεια, παρατίθεται ο κώδικας που αφορά την προσθήκη εικόνας προφίλ του ασθενή:

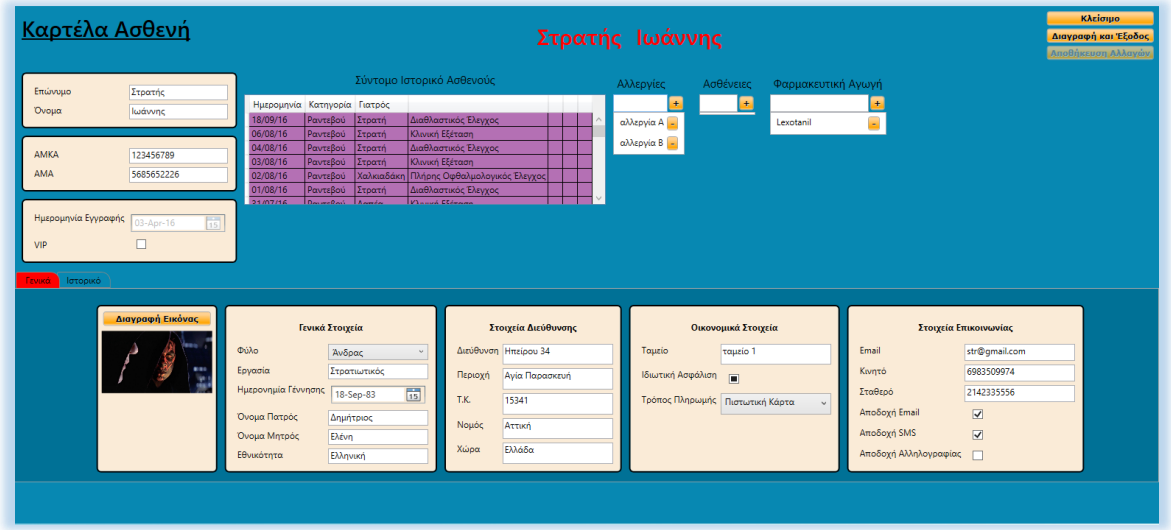
```
...
//PROSTHIKI EIKONAS
        private void btn_add_photo_Click(object sender, RoutedEventArgs e)
        {

            // Create OpenFileDialog
            dlg = new Microsoft.Win32.OpenFileDialog();

            // Set filter for file extension and default file extension
            dlg.DefaultExt = ".jpg";
            dlg.Filter = "JPEG Files (*.jpeg)|*.jpeg|PNG Files (*.png)|*.png|JPG
Files (*.jpg)|*.jpg";

            // Display OpenFileDialog by calling ShowDialog method
            Nullable<bool> result = dlg.ShowDialog();


            // Get the selected file name and display in a TextBox
            if (result == true)
            {
                //elegxos megethous arxeiou (bytes)
                FileInfo fi = new FileInfo(dlg.FileName);
                if (fi.Length > 5000000)
                {
                    MessageBox.Show("Διαλέξτε αρχείο μικρότερο των 5MB", "Μεγάλο
Αρχείο", MessageBoxButton.OK, MessageBoxImage.Error);
                    return;
                }

                //temporary person,context (den akoune ston listener gia changes
tou constructor)
                eyecrystalEntities tempcontext = new eyecrystalEntities();
                patient pers = tempcontext.patient.Find(current_patient.AMKA);
                img_photo.DataContext = pers;   //apagkistrwnw to control apo to
datacontext tou constructor. sinepws den akouei ston listener gia changes

                //copy target
                photopath = AppDomain.CurrentDomain.BaseDirectory +
@"_Images/Patient_Photos/";

                //elegxos ean iparxei to folder _Images
```

```
                if (!Directory.Exists(photopath))
                {
                    Directory.CreateDirectory(photopath);
                }


                using (tempcontext)
                {
                    //check if file already exists
                    if (File.Exists(photopath + Path.GetFileName(pers.Photo)))
                    {
                        File.Delete(photopath + Path.GetFileName(pers.Photo));

                    }

                    //ektelesi antigrafis tou arxeiou sto fakelo tou Application
Images
                    File.Copy(dlg.FileName, photopath + current_patient.AMKA +
"_" + metritis_photo.ToString() + Path.GetExtension(dlg.FileName), true);

                    //set to path sto context
                    pers.Photo = pers.AMKA + "_" + metritis_photo.ToString() +
Path.GetExtension(dlg.FileName);

                    //save to database
                    StoreDB.StoreDBclinic.updateItem(tempcontext);

                    dialogue.validated_Data("Η εικόνα αποθηκεύτηκε με επιτυχία");

                    metritis_photo++;
                }
                //tropopoihsh UI
                btn_add_photo.Visibility = Visibility.Collapsed;
                btn_delete_photo.Visibility = Visibility.Visible;
            }
        }
...
```

Αρχικά, δημιουργείται ένα αντικείμενο τύπου OpenFileDialog, το οποίο δέχεται παραμέτρους, όπως αποδεκτοί τύποι αρχείων, κ.α. Στην συνέχεια, εμφανίζεται το παράθυρο επιλογής αρχείου και γίνεται ένας έλεγχος εάν το αρχείο είναι μικρότερο από 5MB. Σε περίπτωση που είναι, τότε μπορεί να συνεχιστεί η διαδικασία αποθήκευσης της φωτογραφίας. Συγκεκριμένα, πρέπει να εκτελεστούν 2 διαδικασίες: πρώτον, αποθήκευση της φωτογραφίας σε ειδικό φάκελο της εφαρμογής και δεύτερον, ενημέρωση της ΒΔ με το όνομα του αρχείου για τον συγκεκριμένο ασθενή. Σε περίπτωση που δεν υπάρξει κάποιο σφάλμα, εμφανίζεται μήνυμα επιτυχίας στο sidebar. Η αντίστροφη διαδικασία εκτελείται όταν ο χρήστης επιθυμεί την διαγραφή της φωτογραφίας του ασθενή.

Εκτός από τις παραπάνω μεθόδους, το αρχείο UC_patient περιέχει πληθώρα επιπλέον μεθόδων, οι οποίες εκτελούν λειτουργίες κυρίως βοηθητικές. Ωστόσο, δεν γίνεται να γίνει αναφορά σε όλο τον κώδικα, για λόγους εξοικονόμησης χώρου, αν και ο κώδικας περιέχει σχόλια, τα οποία προσδιορίζουν τις λειτουργίες που υλοποιούνται.

Ενδεικτικά, παρατίθεται screenshot του φακέλου ασθενή:



Εικόνα 27. Παράδειγμα Ιατρικού Φακέλου Ασθενή

Με αφορμή την ανάλυση του κώδικα για τις βασικές λειτουργίες της εφαρμογής, που έχουμε δει έως αυτό το σημείο, έχουμε συναντήσει επιπλέον λειτουργίες, οι οποίες έχουν βοηθητικό ρόλο, όπως αποθήκευση context κ.α. Στην ανάλυση των κλάσεων που ακολουθεί, θα επικεντρωθούμε στον κώδικα και στους αλγορίθμους που έχουν υλοποιηθεί, προκειμένου να ικανοποιούνται οι βασικές απαιτήσεις της εφαρμογής. Για αυτό τον λόγο, δεν θα επεκταθούμε στην ανάλυση του κώδικα, ο οποίος ενεργεί βοηθητικά. Συνεπώς, προχωρούμε στην ανάλυση του κώδικα που υλοποιεί την απαίτηση προγραμματισμού των ραντεβού.

4.2.7 Προγραμματισμός Ραντεβού

Στο UC_Appointment.xaml, υλοποιείται η απαίτηση για προγραμματισμό ραντεβού των ασθενών. Παρακάτω παρατίθεται ο αλγόριθμος, ο οποίος αφορά στην εμφάνιση επιτρεπτών ημερομηνιών, στα οποία ο ασθενής μπορεί να προγραμματίσει ραντεβού:

UC_Appointment.xaml.cs (Code-behind file)

```
...
private void DatePicker_Loaded(object sender, RoutedEventArgs e)
        {
            if (datepickerfirstload)
            {
                if (!isEditForm)
                {
                    //picks currentdate when loaded
                    picker = (DatePicker)sender;
                    picker.DisplayDateStart = DateTime.Today;
                    picker.SelectedDate = DateTime.Today;
                }
                else
                {
                    //picks currentdate when loaded if editform
                    picker = (DatePicker)sender;
                    picker.DisplayDateStart = DateTime.Today;
                }

                eyecrystalEntities tempcontext = new eyecrystalEntities();
                using (tempcontext)
```

```
                {
                    List<appointment> future_appoints =
tempcontext.appointment.Where(c => c.AMKA == current_patient.AMKA &&
DateTime.Compare(c.Appointment_Date, DateTime.Today) >= 0).ToList<appointment>();

                    foreach (var item in future_appoints)
                    {
                        //ean sto edit appointment anikei se blackdate eksairese
to gia na mporw na to kanw edit

                        if (isEditForm && item.Appointment_Date == selectedDate)
                        {
                            continue;
                        }
                        else if (!isEditForm)
                        {
                            appointment temp_app = new appointment()
                            {
                                AMKA = item.AMKA,
                                Appointment_Date =
item.Appointment_Date.AddDays(1)
                            };


                            if (!future_appoints.Contains(temp_app))
                            {
                                picker.SelectedDate = temp_app.Appointment_Date;
                            }

                        }
                        CalendarDateRange item_range = new
CalendarDateRange(item.Appointment_Date);
                        picker.BlackoutDates.Add(item_range);


                    }

                    load_reserved_apps();
                }

                datepickerfirstload = false;
            }
        }
...
```

Αρχικά, αποθηκεύουμε σε μία λίστα όλα τα αντικείμενα τύπου appointment, που αφορούν τον ενεργό ασθενή. Στις μελλοντικές ημερομηνίες, στις οποίες έχει ήδη προγραμματίσει ραντεβού ο ασθενής, δεν έχει την δυνατότητα να προγραμματίσει δεύτερο ραντεβού. Αρχικά, το calendar, με το οποίο μπορεί να γίνει επιλογή μιας ημερομηνίας, θέτει την ημερομηνία προγραμματισμού ραντεβού την τρέχουσα ημέρα. Ωστόσο, εάν έχει ήδη προγραμματίσει ραντεβού ο ασθενής εκείνη την ημέρα, ο αλγόριθμος την εισάγει σε μία blacklist και αυτόματα επιλέγει την επόμενη ημέρα. Συνεπώς, το calendar δείχνει την επόμενη μέρα προεπιλεγμένη.

Με τον παραπάνω αλγόριθμο, λαμβάνεται απόφαση για την προεπιλεγμένη ημέρα προγραμματισμού ραντεβού για ένα συγκεκριμένο ασθενή. Το επόμενο βήμα είναι να αποφασιστούν οι προβλεπόμενες ώρες προγραμματισμού ραντεβού, για την επιλεγμένη ημέρα του calendar.

Στην συνέχεια, παρατίθεται ο κώδικας, ο οποίος προσδιορίζει τις επιτρεπτές ώρες, που μπορούν να επιλεχθούν για μία συγκεκριμένη ημερομηνία:

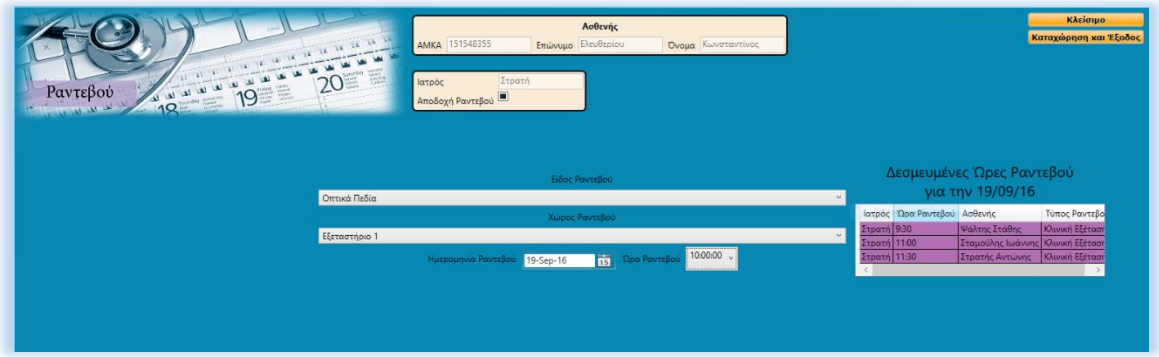
```
...
private ObservableCollection<TimeSpan> availableTimeLineforDoctor()
        {
            //default list gia timeline
            ObservableCollection<TimeSpan> lst_app_time = new
ObservableCollection<TimeSpan>();
            ObservableCollection<TimeSpan> lst_app_time_temp = new
ObservableCollection<TimeSpan>();
            lst_app_time.Add(new TimeSpan(9, 30, 0));
            lst_app_time.Add(new TimeSpan(10, 0, 0));
            lst_app_time.Add(new TimeSpan(10, 30, 0));
            lst_app_time.Add(new TimeSpan(11, 0, 0));
            lst_app_time.Add(new TimeSpan(11, 30, 0));
            lst_app_time.Add(new TimeSpan(12, 0, 0));
            lst_app_time.Add(new TimeSpan(12, 30, 0));
            lst_app_time.Add(new TimeSpan(13, 0, 0));
            lst_app_time.Add(new TimeSpan(13, 30, 0));
            lst_app_time.Add(new TimeSpan(14, 0, 0));
            lst_app_time.Add(new TimeSpan(14, 30, 0));
            lst_app_time.Add(new TimeSpan(15, 0, 0));
            lst_app_time.Add(new TimeSpan(15, 30, 0));
            lst_app_time.Add(new TimeSpan(16, 0, 0));
            lst_app_time.Add(new TimeSpan(16, 30, 0));
            lst_app_time.Add(new TimeSpan(17, 0, 0));
            lst_app_time.Add(new TimeSpan(17, 30, 0));
            lst_app_time.Add(new TimeSpan(18, 0, 0));
            lst_app_time.Add(new TimeSpan(18, 30, 0));
            lst_app_time.Add(new TimeSpan(19, 0, 0));
            lst_app_time.Add(new TimeSpan(19, 30, 0));
            lst_app_time.Add(new TimeSpan(20, 0, 0));
            lst_app_time.Add(new TimeSpan(20, 30, 0));
            lst_app_time.Add(new TimeSpan(21, 0, 0));
            lst_app_time.Add(new TimeSpan(21, 30, 0));
            lst_app_time.Add(new TimeSpan(22, 0, 0));
            lst_app_time.Add(new TimeSpan(22, 30, 0));


            //wres parelthontos
            foreach (var item in lst_app_time)
            {
                if (!isEditForm && item < DateTime.Now.TimeOfDay && selectedDate
== DateTime.Today)
                {
                    lst_app_time_temp.Add(item);
                }
            }


            //afairesi parelthouswn wrwn
            foreach (var item in lst_app_time_temp)
            {
                if (lst_app_time.Contains(item))
                {
                    lst_app_time.Remove(item);
                }

            }

            //afairesi reserverd time
            eyecrystalEntities tempcontext = new eyecrystalEntities();
```

```
            using (tempcontext)
            {
                List<appointment> reserved_apps;
                if (ConfigSession.isdoctor)
                {
                    reserved_apps = tempcontext.appointment.Where(a => a.doctor
== ConfigSession.Current_user.AMKA && a.Appointment_Date ==
selectedDate).ToList<appointment>();
                }
                else if (!ConfigSession.isdoctor && cmb_docs.SelectedIndex != -1)
                {
                    personel doc = (personel)cmb_docs.SelectedItem;
                    reserved_apps = tempcontext.appointment.Where(a => a.doctor
== doc.AMKA && a.Appointment_Date == selectedDate).ToList<appointment>();
                }
                else
                {
                    reserved_apps = tempcontext.appointment.Where(a => a.doctor
== current_appointment.doctor && a.Appointment_Date ==
selectedDate).ToList<appointment>();
                }

                if (reserved_apps != null)
                {
                    foreach (var item in reserved_apps)
                    {

                        if (lst_app_time.Contains(item.Appointment_Time))
                        {
                            //ean prokeitai gia epeksergasia o gia ton
currentpatient na mporei na dei tin arxiki wra tou rantevou
                            if (isEditForm && item.AMKA == current_patient.AMKA
&& item.Appointment_Date == current_appointment.Appointment_Date)
                            {
                                continue;
                            }
                            lst_app_time.Remove(item.Appointment_Time);
                        }
                    }
                }

            }
            return lst_app_time;
        }
...
```

Αρχικά, δημιουργείται μία λίστα, η οποία αποθηκεύει τις προβλεπόμενες ώρες προγραμματισμού ραντεβού, από τις 09:30 έως τις 22:30, για μία επιλεγμένη ημερομηνία. Στην συνέχεια, αφαιρούνται από την λίστα, οι παρελθοντικές ώρες. Δηλαδή, εάν αποφασιστεί να προγραμματιστεί ραντεβού για την ίδια ημέρα, δεν θα πρέπει να υπολογιστούν οι ώρες, που έχουν ήδη παρέλθει, αλλά μόνο οι μελλοντικές. Έπειτα, γίνεται ένας έλεγχος για τον συνδεδεμένο χρήστη – ιατρό, εάν για την επιλεγμένη ημερομηνία έχει ήδη προγραμματισμένα ραντεβού, οπότε και θα πρέπει να μην ληφθούν υπόψη αυτές οι ώρες για την συγκεκριμένη ημερομηνία. Σε περίπτωση που επιθυμεί η γραμματεία να προγραμματίσει ραντεβού, εφαρμόζεται ο ίδιος έλεγχος, ανάλογα με τον ιατρό, τον οποίο επιλέγει για το ραντεβού. Τέλος, επιστρέφεται η λίστα με τις προβλεπόμενες ώρες για την επιλεγμένη ημερομηνία και τον επιλεγμένο ιατρό.

Κάθε εγγραφή ραντεβού περιέχει την ιδιότητα Status. Ο λόγος ύπαρξης της υπόψη ιδιότητας, αφορά στον έλεγχο κατάστασης του ραντεβού, εάν δηλαδή έχει εγκριθεί από την κλινική ή όχι. Όταν ο προγραμματισμός ραντεβού γίνεται από την κλινική, τότε το ραντεβού είναι ήδη εγκεκριμένο. Ωστόσο, μέσω του Ιστότοπου, ένας ασθενής μπορεί να δηλώσει ενδιαφέρον για προγραμματισμό ραντεβού. Σε αυτή την περίπτωση, το ραντεβού δεν είναι εγκεκριμένο, και θα πρέπει να εγκριθεί από την κλινική εκ των υστέρων.

Ενδεικτικά, παρατίθεται screenshot στην οποία φαίνονται τα παραπάνω:



Εικόνα 28. Καταχώρηση Ραντεβού (Screenshot)

Στην παραπάνω εικόνα είναι συνδεδεμένη η ιατρός με επίθετο Στρατή, και επιθυμεί να προγραμματίσει ραντεβού για τον ασθενή Ελευθερίου Κωνσταντίνο. Στην συνέχεια, έχει επιλέξει την 19/09/16 ως επιθυμητή ημερομηνία προγραμματισμού ραντεβού. Ωστόσο, η εφαρμογή υπενθυμίζει στην ιατρό, ότι για την συγκεκριμένη ημερομηνία έχει ήδη προγραμματίσει ραντεβού σε 3 άλλους ασθενείς, όπως φαίνεται στην εικόνα. Επιπρόσθετα, η εφαρμογή δεν του επιτρέπει να επιλέξει τις ώρες που αντιπροσωπεύουν αυτά τα 3 ραντεβού. Για αυτό τον λόγο, παρατηρούμε ότι ενώ κανονικά, η ώρα 09:30 είναι το πρώτο στοιχείο της λίστας Ώρα Ραντεβού, εντούτοις έχει αφαιρεθεί από την λίστα και συνεπώς έχει ως πρώτο στοιχείο την ώρα 10:00 καθώς η ώρα 09:30 είναι δεσμευμένη για τον ασθενή με ονοματεπώνυμο Στάθης Ψάλτης.

4.2.8 Υλοποίηση Κλινικής Εξέτασης Ασθενή

Η εφαρμογή επιτρέπει την ψηφιακή παρακολούθηση δύο εξετάσεων. Η πρώτη είναι η κλινική εξέταση και η δεύτερη είναι ο διαθλαστικός έλεγχος. Στην παρούσα ενότητα θα αναλυθούν τα σημαντικότερα χωρία του κώδικα, που αφορούν την κλινική εξέταση. Παρακάτω παρατίθεται ο κώδικας που αφορά στον ορισμό των μεταβλητών και του constructor δημιουργίας νέας κλινικής εξέτασης:

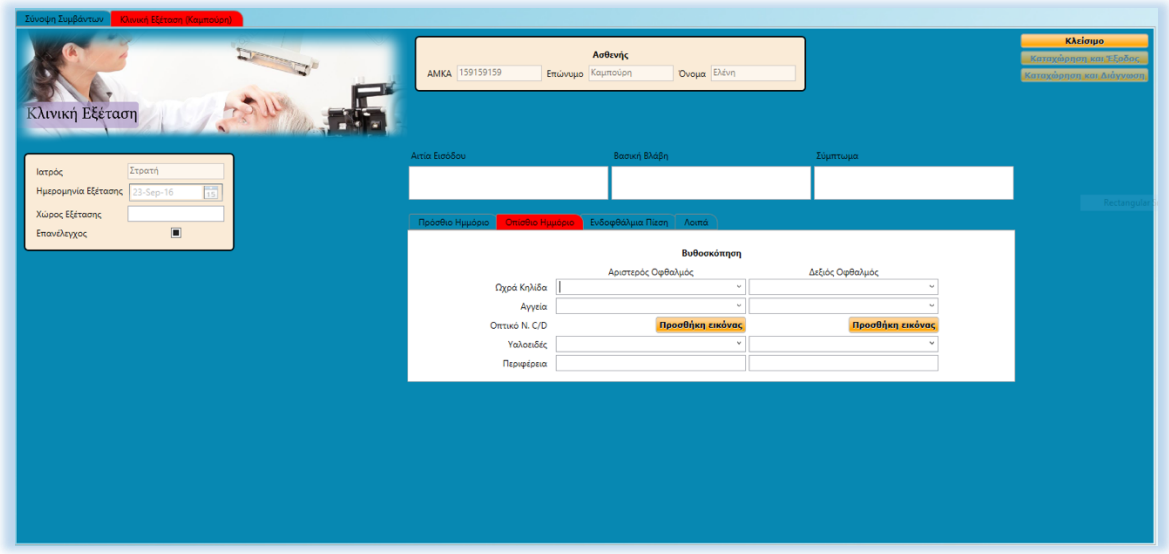
```
...
namespace crystaleye._UserControls
{
    /// <summary>
    /// Interaction logic for UC_clinical_examination.xaml
    /// </summary>
    public partial class UC_clinical_examination : UserControl
    {
        //FIELDS
        private string photopath = AppDomain.CurrentDomain.BaseDirectory +
@"_Images/Clinical_Examinations/";
        public patient current_patient;
        private UC_Hints hints;
        public UC_SystemDialogue dialogue;
        public UC_ValidationErros dialogueErrors;
        public UC_Element_Action elementActions;
```

```
        public eyecrystalEntities context;
        private Window parentwindow;
        public ObservableCollection<String> MyErrors { get; private set; }
//validation errors hosting list
        public clinical_examination current_examination;
        private bool isEditForm = false;
        private bool uploadedphoto = false;
        Dictionary<string, string> photosuploaded = new Dictionary<string,
string>();

        //1os Constructor -- NEW
        public UC_clinical_examination(patient patient)
        {

            //dimiourgia/init neou entity clinicalexamination
            current_examination = new clinical_examination()
            {
                AMKA = patient.AMKA,
                doctor = ConfigSession.Current_user.AMKA,
                Examination_Date=DateTime.Today
            };

            //init Hints list
            hints = new UC_Hints(this);

            //reference ston astheni pou anaferetai i eksetasi
            current_patient = patient;

            //dimiourgia antikeimenou dialogue errors (UC)
            dialogueErrors = new UC_ValidationErros();

            //dimiourgia antikeimenou dialogue (UC)
            dialogue = new UC_SystemDialogue();

            //MyErrors List init
            MyErrors = new ObservableCollection<string>();

            //reference sto parent window
            parentwindow = Application.Current.MainWindow;

            //dimiourgia context gia to sigkekrimeno Usercontrol
            context = StoreDB.StoreDBclinic.createContext();

            //dimiourgia antikeiemenou actions kai energopoihsh montelou gia
patients
            elementActions = new UC_Element_Action(current_patient,
context,null);
            elementActions.patient_actions();

            //load tou context me to current_examination
            context.clinical_examination.Add(current_examination);


            InitializeComponent();

            //thetw datacontext to entity current_examination
            this.DataContext = current_examination;

            //datacontext gia to grid me ta stoixeia tou patient(readonly)
            grd_currentpatient.DataContext = current_patient;
```

```
            load_combobox_items();

            //handler gia to property changed tou current patient
            current_examination.PropertyChanged +=
Current_examination_PropertyChanged;
            ((App)Application.Current).Exit += UC_clinical_examination_Exit;

            photoButtonsUpdate();
        }
...
```

Αρχικά, ορίζεται το path του φακέλου, στον οποίο θα αποθηκεύονται οι συνοδευτικές φωτογραφίες μίας κλινικής εξέτασης, την λειτουργία των οποίων θα αναλύσουμε στην επόμενη ενότητα. Στην συνέχεια, ορίζονται μεταβλητές, που θα αποτελούν reference σε στοιχεία, όπως τον ενεργό ασθενή, στοιχεία που ανήκουν στο sidebar, την κλινική εξέταση κ.α.

Σε ότι αφορά τον πρώτο constructor, αυτός απαιτεί ως argument, τον ασθενή. Έπειτα, δημιουργείται ένα αντικείμενο τύπου clinical examination, και αρχικά αποθηκεύεται το ΑΜΚΑ του ασθενούς, το ΑΜΚΑ του ιατρού-εξεταστή, και η ημέρα εξέτασης. Αυτή η τριάδα δεδομένων αποτελεί και primary key για την οντότητα clinical_examination. Στην συνέχεια, υλοποιούνται τα references, που αναφέρθηκαν στην προηγούμενη παράγραφο και δημιουργείται νέο αντικείμενο context. Η μέθοδος load_combox_items(), ενημερώνει με τα απαραίτητα στοιχεία όλα τα comboboxes που περιέχει η φόρμα δημιουργίας νέας κλινικής εξέτασης. Επειδή, η συγκεκριμένη φόρμα περιέχει πολλά comboboxes, κρίθηκε σκόπιμο η διαχείριση των δεδομένων που θα περιέχουν να πραγματοποιείται μέσα σε μία μόνο μέθοδο, με σκοπό την δημιουργία καθαρότερου κώδικα. Στην συνέχεια, προστίθεται ένας Listener για την περίπτωση που αλλάξει η τιμή στα πεδία της φόρμας, με σκοπό να γνωρίσει η εφαρμογή την αλλαγή και να τροποποιήσει διάφορα χαρακτηριστικά, όπως ενεργοποίηση κουμπιού αποθήκευσης. Έπειτα, δημιουργείται δεύτερος Listener για την περίπτωση εξόδου από το πρόγραμμα. Ο συγκεκριμένος Listener οδηγεί στην λύση του προβλήματος, κατά το οποίο ο χρήστης δημιουργεί νέο αρχείο εικόνας αλλά αποφασίζει να μην αποθηκεύσει την κλινική εξέταση. Σε αυτή την περίπτωση το αρχείο εικόνας θα πρέπει να διαγραφεί, αφού δεν υπάρχει στην ΒΔ εγγραφή της κλινικής εξέτασης με την οποία να υπάρχει αντιστοίχιση. Τέλος, με την μέθοδο photoButtonsUpdate(), γίνεται έλεγχος ύπαρξης ή όχι υφιστάμενου φωτογραφικού υλικού για μία κλινική εξέταση και αναλόγως προσαρμόζεται η εμφάνιση των κουμπιών «Προσθήκη», «Εμφάνιση» και «Διαγραφή» φωτογραφίας. Παρακάτω παρατίθεται screenshot με την φόρμα εισαγωγής καινούριας κλινικής εξέτασης:



Εικόνα 29. Παράδειγμα Εκτέλεσης νέας Κλινικής Εξέτασης

4.2.9 Ψηφιακή Αρχειοθέτηση Φωτογραφιών με Ιατρικό Περιεχόμενο

Η καταχώρηση και επεξεργασία μίας κλινικής εξέτασης για έναν ασθενή επιτρέπει τόσο την εισαγωγή διαφόρων τιμών σε επιμέρους πεδία που αφορούν την εξέταση όσο και την εισαγωγή φωτογραφιών για τα πεδία «Κερατοειδής» και «Οπτικό Ν. C/D». Έχει ληφθεί η παραδοχή για την απλοποίηση των διαδικασιών, ότι οι εικόνες αυτές θα είναι συμβατού τύπου (π.χ. jpg). Ωστόσο, προσεγγίζοντας το θέμα ρεαλιστικά, οι τύποι εικόνων που περιέχουν ιατρική πληροφορία είναι συγκεκριμένοι και η χρήση τους αποτελεί μονόδρομο σε παρόμοιες εφαρμογές, που έχουν τεθεί σε παραγωγή.

Όπως αναφέρθηκε προηγουμένως, η επεξεργασία της εικόνας λαμβάνει χώρα σε ένα ξεχωριστό παράθυρο. Υπάρχουν δύο κλάσεις στο crystaleye Project, οι οποίες ευθύνονται για την εμφάνιση αυτών των παραθύρων: η win_Keratoeidis_ImageEditor.xaml και η win_OptikoNCD_ImageEditor.xaml. Επειδή και τα δύο αρχεία εμφανίζουν παρόμοιο κώδικα, θα αναλυθεί ο κώδικας του αρχείου win_Keratoeidis_ImageEditor.xaml.cs:

Win_Keratoeidis_ImageEditor.xaml.cs (Code-behind file)

```
...
public partial class win_Keratoeidis_ImageEditor : Window
    {
        private string photopath;
        private string photofile;
        private TextBox txtBoxwrite;
        private TextBlock txtBlockwrite;
        private clinical_examination clexam, clex;
        private string attribute;
        private UC_clinical_examination UC_sender;
        private bool uploaded = false;


        //CONSTRUCTOR
        public win_Keratoeidis_ImageEditor(UC_clinical_examination uc,
clinical_examination exam, string file, string mati)
        {
            UC_sender = uc;
            photofile = file;
```

```
            attribute = mati;
            //init photopath

            clexam = exam;
            photopath = AppDomain.CurrentDomain.BaseDirectory +
@"_Images/Clinical_Examinations";
            check_Directory_Existance(photopath);

            InitializeComponent();

            //iparxei eggrafi / arxeio kai tha to tropopoihsw
            if (file != null && file.Length!=0)
            {
                BitmapImage bmp = new BitmapImage();
                bmp.BeginInit();
                bmp.CacheOption = BitmapCacheOption.OnLoad;
                bmp.UriSource = new Uri(photopath +"/"+ photofile);
                bmp.EndInit();

                img_keratoeidis.Source = bmp;
            }
        }
...
```

Αρχικά ορίζονται οι global μεταβλητές της κλάσης. Αυτές περιλαμβάνουν το path του φακέλου, που θα αποθηκεύονται οι εικόνες, το όνομα των αρχείων των εικόνων κ.α. Στην συνέχεια, ο constructor θέτει το source για την εικόνα που θα φιλοξενεί στο παράθυρο και μέσω του κώδικα XAML που αφορά το συγκεκριμένο αρχείο, έχουν τοποθετηθεί στοιχεία που επιτρέπουν την επεξεργασία της εικόνας, όπως pen, eraser, Textbox κ.α. Τα arguments που έχει ο constructor, αφορούν στον να υπάρχει σύνδεση της εικόνας που δημιουργείται με τα στοιχεία της εξέτασης που αφορά.

Στην συνέχεια έχει ενδιαφέρον η ανάλυση του κώδικα, που αφορά στην αποθήκευση της εικόνας:

```
...
 //SAVE IMAGE
        private void btn_Save_Click(object sender, RoutedEventArgs e)
        {
            MessageBoxResult result = MessageBox.Show("Είστε σίγουροι ότι θέλετε
να αποθηκεύσετε το αρχείο εικόνας?", "Επιβεβαίωση", MessageBoxButton.OKCancel,
MessageBoxImage.Question);
            if (result == MessageBoxResult.OK)
            {
                //exam reference
                eyecrystalEntities tempcontext = new eyecrystalEntities();
                if (clexam != null)
                {
                    //load context
                    //edit form
                    clex = tempcontext.clinical_examination.Find(clexam.AMKA,
clexam.Examination_Date);

                    if (attribute == "PH_AO_Keratoeidis")
                    {
                        photofile = "AK" +clex.AMKA +
(clex.Examination_Date.Date.ToString("ddMM")) + ".jpg";
                    }
                    else if (attribute == "PH_DO_Keratoeidis")
                    {
```

```
                        photofile = "DK" + clex.AMKA +
(clex.Examination_Date.Date.ToString("ddMM")) + ".jpg";
                    }
                }
                else
                {
                    //new form
                    if (attribute == "PH_AO_Keratoeidis")
                    {
                        photofile = "AK" + UC_sender.current_examination.AMKA +
(DateTime.Now.ToString("ddMM")) + ".jpg";
                    }
                    else if (attribute == "PH_DO_Keratoeidis")
                    {
                        photofile = "DK" + UC_sender.current_examination.AMKA +
(DateTime.Now.ToString("ddMM")) + ".jpg";
                    }

                }


                var ms = new MemoryStream();

                using (FileStream fs = new FileStream(photopath+ "/" + photofile,
FileMode.Create))
                {
                    RenderTargetBitmap rtb = new
RenderTargetBitmap((int)inkcanvas_box.ActualWidth,
(int)inkcanvas_box.ActualHeight, 96d, 96d, PixelFormats.Default);
                    rtb.Render(inkcanvas_box);
                    JpegBitmapEncoder encoder = new JpegBitmapEncoder();
                    encoder.Frames.Add(BitmapFrame.Create(rtb));
                    encoder.Save(fs);

                    //save context
                    using (tempcontext)
                    {
                        //load context
                        //edit form
                        if (clexam!=null && attribute == "PH_AO_Keratoeidis")
                        {
                            clex.PH_AO_Keratoeidis = photofile;
                        }
                        else if (clexam != null && attribute ==
"PH_DO_Keratoeidis")
                        {
                            clex.PH_DO_Keratoeidis = photofile;
                        }
                        //new form
                        else if(clexam == null && attribute ==
"PH_AO_Keratoeidis")
                        {
                            UC_sender.current_examination.PH_AO_Keratoeidis =
photofile;
                        }
                        else if (clexam == null && attribute ==
"PH_DO_Keratoeidis")
                        {
                            UC_sender.current_examination.PH_DO_Keratoeidis =
photofile;
```

```
                        }

                        //save to db the path
                        if (StoreDB.StoreDBclinic.updateItem(tempcontext)!=null)
                        {
                            MessageBox.Show("Κάτι πήγε στραβά. Προσπαθήστε
Ξανά!", "Σφάλμα", MessageBoxButton.OK, MessageBoxImage.Error);
                        }
                        else
                        {
                            uploaded = true;
                            MessageBox.Show("Η εικόνα αποθηκεύτηκε", "Επιτυχής
Αποθήκευση", MessageBoxButton.OK, MessageBoxImage.Information);
                            Window win =
Application.Current.Windows.OfType<Window>().SingleOrDefault(w => w.Name ==
"window_keratoeidis");
                            win.Close();
                        }

                    }
                }

            }
}
...
```

Αρχικά, εμφανίζεται έναν Messagebox, με το οποίο γίνεται ερώτηση στον χρήστη εάν επιθυμεί να προχωρήσει η διαδικασία αποθήκευσης της εικόνας. Στην συνέχεια και με δεδομένο ότι ο χρήστης έχει απαντήσει θετικά, δημιουργείται ένα προσωρινό context, το οποίο φορτώνει την συγκεκριμένη εξέταση, για την οποία δημιουργείται η εικόνα. Έπειτα, προσδιορίζεται το όνομα του αρχείου της εικόνας και ενημερώνεται το context, ενώ μέσω της μεθόδου Save() της κλάσης JpegBitmapEncoder, η εφαρμογή προβαίνει σε αποθήκευση της εικόνας στον φάκελο που έχει καθοριστεί από τον constructor. Τέλος, εμφανίζεται κατάλληλο μήνυμα, το οποίο ενημερώνει τον χρήστη για την επιτυχία ή όχι της διαδικασίας.

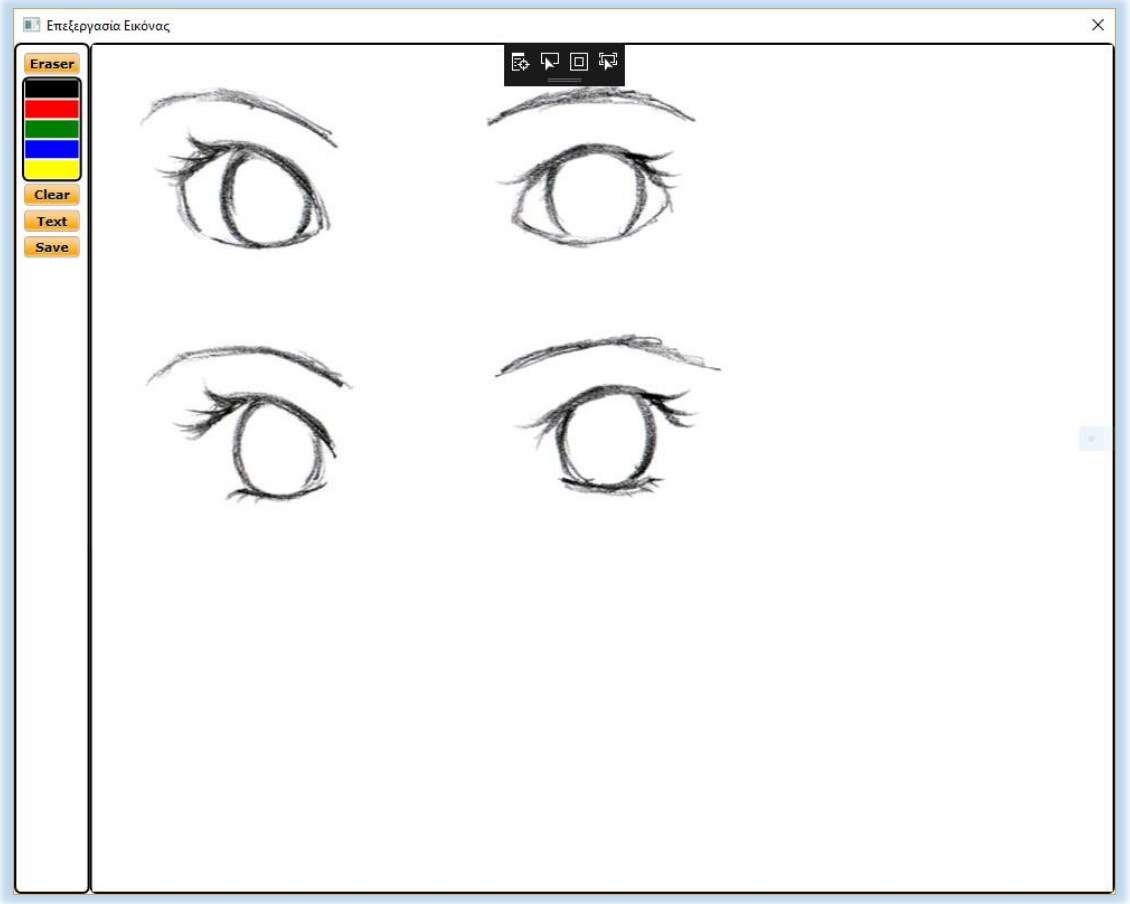
Ο υπόλοιπος κώδικας που φιλοξενείται στην υπό εξέταση κλάση αφορά στις λειτουργίες επεξεργασίας της φωτογραφίας. Για παράδειγμα, παρακάτω παρατίθεται το κομμάτι του κώδικα που αφορά την προσθήκη Textbox στην εικόνα:

```
...
private void Inkcanvas_box_MouseDown(object sender, MouseButtonEventArgs e)
        {
            //get mouse position
            ConfigSession.Mouse_Postiion = Mouse.GetPosition(inkcanvas_box);

            //dimiourgia textbox
            txtBoxwrite = new TextBox();
            txtBoxwrite.Text = "Write something here";
            InkCanvas.SetTop(txtBoxwrite, ConfigSession.Mouse_Postiion.Y);
            InkCanvas.SetLeft(txtBoxwrite, ConfigSession.Mouse_Postiion.X);
            inkcanvas_box.Children.Add(txtBoxwrite);

            txtBoxwrite.KeyDown += Tex_KeyDown;

            //apenergopoiw ton listener gia to mouse event
            inkcanvas_box.MouseDown -= Inkcanvas_box_MouseDown;
        }
...
```

Αρχικά, αποθηκεύεται η θέση του κέρσορα του ποντικιού στην οθόνη. Έπειτα, δημιουργείται ένα αντικείμενο τύπου Textbox και στην συνέχεια με κατάλληλες παραμέτρους που εισάγονται στην κλάση InkCanvas, τοποθετείται το Textbox στο σημείο που είναι ο κέρσορας. Τέλος, εισάγεται ένας listener για την περίπτωση που ο χρήστης πατήσει οποιοδήποτε πλήκτρο στο πληκτρολόγιο. Όταν όμως το πλήκτρο είναι το Enter, τότε εκτελείται ο παρακάτω αλγόριθμος:

```
...
private void Tex_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.Enter)
            {
                txtBlockwrite = new TextBlock();

                txtBlockwrite.Text = txtBoxwrite.Text;
                txtBlockwrite.Background = Brushes.White;

                InkCanvas.SetTop(txtBlockwrite, ConfigSession.Mouse_Postiion.Y);
                InkCanvas.SetLeft(txtBlockwrite, ConfigSession.Mouse_Postiion.X);
                inkcanvas_box.Children.Add(txtBlockwrite);

                inkcanvas_box.Children.Remove(txtBoxwrite);

                txtBoxwrite.KeyDown -= Tex_KeyDown;

               //reset ink
                btn_EditingMode.Content = "Pen";
                inkcanvas_box.EditingMode = InkCanvasEditingMode.Ink;
                brd_ink_options.Visibility = Visibility.Visible;
            }
        }
...
```

Ουσιαστικά, δημιουργείται ένα αντικείμενο τύπου Textblock, το οποίο έχει ως Text, την σημείωση που έγραψε ο χρήστης στο TextBox. Έπειτα, αντικαθίσταται το Textbox με το Textblock και, αφαιρείται και ο listener που «άκουγε» την πληκτρολόγηση του χρήστη. Τέλος, τίθεται εκ νέου το editing mode του αντικειμένου inkcanvas_box σε mode ink. Συμπερασματικά, δίνεται η δυνατότητα στον χρήστη να προσθέσει μία σημείωση, παρέχοντας ιατρική πληροφορία σε οποιοδήποτε σημείο της εικόνας. Παρακάτω παρατίθεται screenshot της λειτουργίας, που περιεγράφηκε παραπάνω:



Εικόνα 30. Παράδειγμα Επεξεργασίας Εικόνας με Ιατρικό Περιεχόμενο

Στην συνέχεια, θα γίνει ανάλυση του κώδικα που υλοποιεί την ανάπτυξη των αναφορών που χρησιμοποιούνται στην εφαρμογή. Οι αναφορές φιλοξενούνται σε ξεχωριστά παράθυρα και όχι σε Usercontrols. Ενδεικτικά, θα εξεταστεί η περίπτωση της αναφοράς, η οποία εμφανίζει το σύνολο του προσωπικού της κλινικής.

4.2.10 Δημιουργία Αναφορών

Η δημιουργία αναφορών στην εφαρμογή πραγματοποιήθηκε με την χρήση του εργαλείου ReportViewer, το οποίο παρέχεται στο Visual Studio. Δημιουργεί ένα αρχείο με επέκταση .rdlc, με το οποίο είναι δυνατή η σχεδίαση μιας αναφοράς.

Αρχικά, δημιουργείται το παράθυρο, το οποίο θα φιλοξενήσει το control ReportViewer, στο οποίο όμως πρέπει να προστεθούν παράμετροι, προκειμένου να πραγματοποιηθεί επικοινωνία με το αρχείο αναφοράς rdlc και να τεθεί ένα datasource με τα δεδομένα που επιθυμούμε να εμφανίζονται στην αναφορά. Παρακάτω παρατίθεται ο κώδικας που αφορά στην υλοποίηση των προαναφερθέντων:

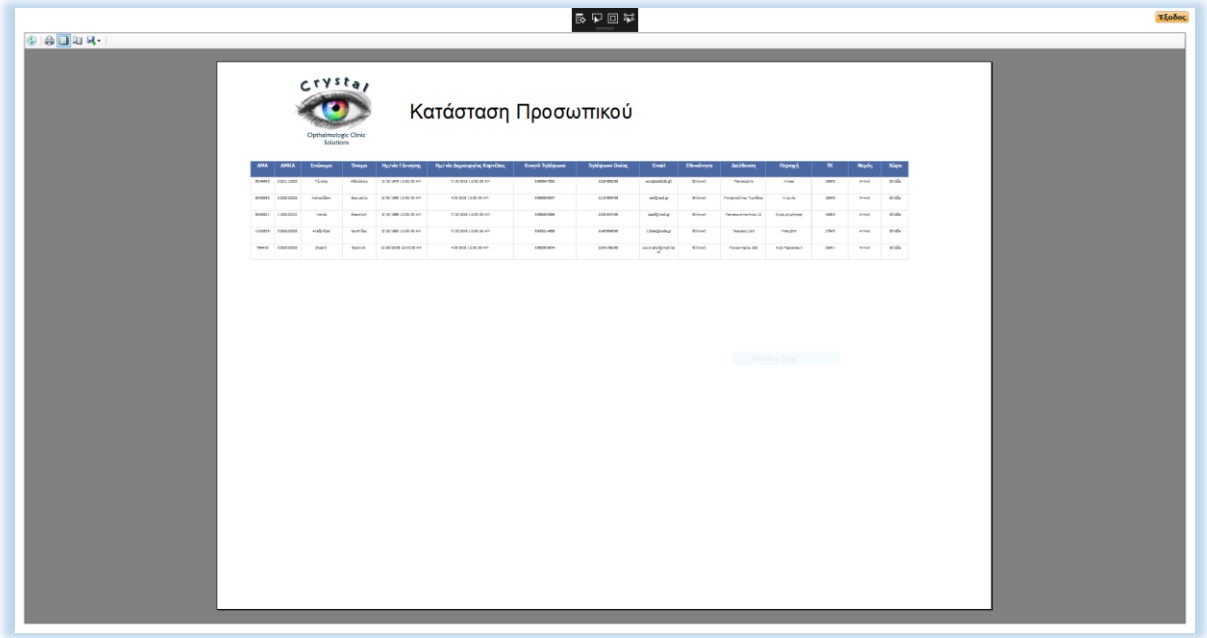
Report_Personel_General.xaml..cs(Code-behind file)

```
...
public partial class Report_Personel_General : Window
    {
        //FIELDS
```

```
        private bool _isReportViewerLoaded;
        private eyecrystalEntities tempcontext;
        private string report_path;

        public Report_Personel_General()
        {
            report_path = @"..\..\_Reports\Report_Personel_General.rdlc";

            InitializeComponent();
        }

        private void _reportViewer_Load(object sender, EventArgs e)
        {
            if (!_isReportViewerLoaded)
            {
                //init dbcontext
                tempcontext = new eyecrystalEntities();

                //init report datasource object
                ReportDataSource reportDataSource = new ReportDataSource();

                //assign datasource to the report (Name,Value)
                reportDataSource.Name = "DataSet1"; // Name of the DataSet we set
in .rdlc
                reportDataSource.Value = tempcontext.personel;

                //assign the datasource to the reportviewer element
                _reportViewer.LocalReport.DataSources.Add(reportDataSource);

                //path to the report.rdlc file
                _reportViewer.LocalReport.ReportPath = report_path;

                //causes the report to be refreshed and rendered
                set_page_settings();

                _isReportViewerLoaded = true;
            }
        }
...
```

Στον constructor δημιουργείται ένα reference στο αρχείο rdlc, που περιέχει την σχεδίαση της αναφοράς. Στην συνέχεια, με την μέθοδο _reportViewer_Load(), παρέχονται παράμετροι, όπως ReportDatasource, που απαιτούνται από το αρχείο rdlc για να λειτουργήσει σωστά. Έτσι καθορίζεται το dataset, το οποίο θα αποτελεί το datasource για την αναφορά, ενώ προσδιορίζεται και το Path του αρχείου rdlc. Τέλος, μέσω της μεθόδου set_page_settings(), καθορίζονται διάφορες ιδιότητες που επηρεάζουν την εμφάνιση της αναφοράς, όπως τα κουμπιά που επιθυμούμε να εμφανίζονται στο μενού κλπ. Ουσιαστικά, το control ReportViewer λειτουργεί ως placeholder για το αρχείο rdlc, που στην προκειμένη περίπτωση είναι το Report_Personel_General.rdlc.

Παρακάτω παρατίθεται ένα screenshot της υπόψη αναφοράς:



Εικόνα 31. Παράδειγμα Αναφοράς Κατάστασης Προσωπικού της κλινικής

Οι αναφορές δεν εξυπηρετούν μόνο στην εμφάνιση συγκεντρωτικών καταστάσεων, όπως στο παραπάνω παράδειγμα. Για αυτό τον λόγο, δημιουργήθηκε και μια αναφορά, η οποία απεικονίζει τα δεδομένα μίας θεραπείας και που μπορεί να προσομοιώσει μία συνταγή ιατρού. Τέλος, επισημαίνεται, ότι υπάρχουν πολλές περιπτώσεις λειτουργιών στην εφαρμογή, οι οποίες χρίζουν συνοδείας αναφοράς. Ωστόσο, επιλέχθηκαν μόνο μερικές περιπτώσεις από αυτές, καθώς ο σκοπός είναι η ανάδειξη της λειτουργίας και ο τρόπος υλοποίησης αυτής.

Στην συνέχεια, ακολουθεί η ανάλυση του κώδικα, ο οποίος αφορά την δημιουργία στατιστικών δεδομένων για την κλινική.

4.2.11 Εμφάνιση Στατιστικών

Η κλάση UC_Statistics.xaml.cs περιλαμβάνει όλο των κώδικα, ο οποίος υλοποιεί την απαίτηση εμφάνισης στατιστικών δεδομένων στην εφαρμογή. Θα πρέπει να σημειωθεί ότι τα στατιστικά δεδομένα εμφανίζονται στο UserControl με την χρήστη των XAML στοιχείων Chart, στα οποία γίνεται συχνά αναφορά στον κώδικα, θέτοντας τιμές στις παραμέτρους που δέχονται. Η φιλοσοφία που έχει αναπτυχθεί είναι η εξής:

α) Λήψη δεδομένων από την ΒΔ

β) Δημιουργία Οντοτήτων, που απεικονίζουν τα παραπάνω δεδομένα

γ) Παραμετροποίηση των Charts, προκειμένου να εμφανίζουν τα στατιστικά δεδομένα.

δ) Δυνατότητα φιλτραρίσματος των στατιστικών δεδομένων, σε σχέση με το είδος των δεδομένων.

Η πρώτη και δεύτερη φάση εξελίσσονται στο StoreDatabase Project, όπως είναι φυσιολογικό. Αρχικά, η StoreDB παρέχει τις απαιτούμενες μεθόδους, προκειμένου να γίνει ανάκτηση των δεδομένων από την ΒΔ και να δημιουργηθούν τα κατάλληλα αντικείμενα, που θα απεικονίζουν τα δεδομένα αυτά. Ενδεικτικά, παρατίθεται ο παρακάτω κώδικας, που αφορά τα προαναφερθέντα, σε σχέση με την εμφάνιση στατιστικών στοιχείων των ραντεβού ανά μήνα:

UC_Statistics.xaml.cs (Code-behind file)

```
...
//statistics appointments by month
        public Collection<StatsByMonth> getAppointmentsByMonth(int year)
        {
            Collection<StatsByMonth> lst_stats = new Collection<StatsByMonth>();

            eyecrystalEntities tempcontext = new eyecrystalEntities();
            using (tempcontext)
            {
                //get examination dates
                List<DateTime> dates_appointments = (from a in
tempcontext.appointment where a.Appointment_Date.Year == year && a.Status==true
orderby a.Appointment_Date select a.Appointment_Date).ToList();

                //group examination dates
                var examspermonth = (from row in dates_appointments
                                     group row by row.Month into g
                                     select new { Period = g.Key,
TotalPeriodCount = g.Count() }).AsEnumerable();

                //create object for each record
                foreach (var record in examspermonth)
                {
                    StatsByMonth examstat = new StatsByMonth(record.Period,
record.TotalPeriodCount);
                    lst_stats.Add(examstat);
                }
            }

            return lst_stats;

               }
...
```

Αρχικά δημιουργείται μία λίστα με αντικείμενα τύπου StatsByMonth. Στην συνέχεια, συμπληρώνεται μία δεύτερη λίστα με στοιχεία τύπου Datetime και που αφορούν τις ημερομηνίες των παρελθοντικών ραντεβού. Έπειτα, με την χρήση κατάλληλων query σε LINQ, ομαδοποιούνται οι ημερομηνίες αυτές, που στην προκειμένη περίπτωση είναι ανα μήνα. Τέλος, κάθε ομάδα τύπου StatsByMonth εισάγεται ως στοιχείο στην πρώτη λίστα.

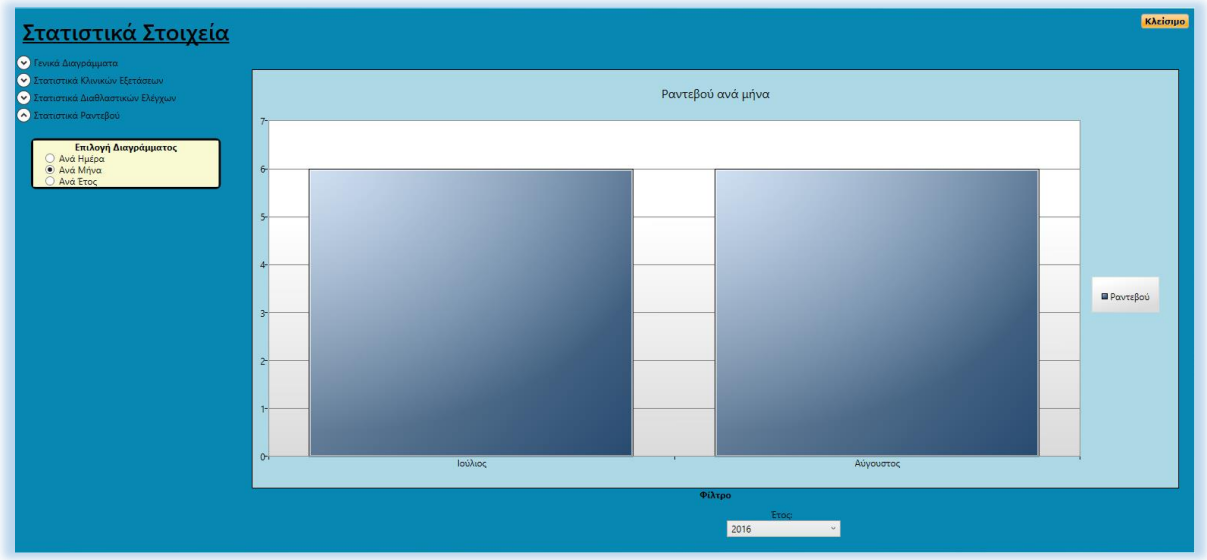
Στην τρίτη φάση, συνδέονται το στοιχείο Chart με τα δεδομένα της παραπάνω λίστας. Η υλοποίηση αυτού γίνεται με τον παρακάτω κώδικα, ο οποίος βρίσκεται στο αρχείο UC_Statistics.xaml.cs:

```
...
private void month_filter_app_Checked(object sender, RoutedEventArgs e)
        {
            int selected_year = (int)lst_years.SelectedItem;

            if (month_filter_app.IsChecked == true)
            {
                stats_month =
StoreDB.StoreDBclinic.getAppointmentsByMonth(selected_year);
                AppMonthStats.ItemsSource = stats_month;

                make_chart_notvisible();
                chartApp_month.Visibility = Visibility.Visible;
            }
        }
```

...

Με την εντολή AppMonthStats.ItemsSource = stats_month, ενημερώνεται το στοιχείο Chart για την πηγή των δεδομένων και στην συνέχεια δημιουργεί τα charts, σύμφωνα με τις παραμέτρους που έχουν δοθεί στον κώδικα XAML.

Ενδεικτικά, παρατίθεται ένα screenshot της συγκεκριμένης λειτουργίας της εφαρμογής:



Εικόνα 32. Παράδειγμα Εμφάνισης Στατιστικών Στοιχείων Ραντεβού

Είναι εμφανές, ότι δίνεται η δυνατότητα στον χρήστη να παρατηρήσει τα στατιστικά των ραντεβού ανα έτος, ανα μήνα και ανα ημέρα. Για κάθε κατηγορία δεδομένων, υπάρχει η δυνατότητα εφαρμογής φίλτρου.

Στην συνέχεια, ακολουθεί η ανάλυση του κώδικα που αφορά την δυνατότητα αποστολής email του χρήστη προς τον ασθενή. Ο υπόψη κώδικας φιλοξενείται στην κλάση MailOperation.cs.

4.2.12 Αποστολή Email

Στο αρχείο MailOperation.cs υλοποιείται η απαίτηση επικοινωνίας του προσωπικού της κλινικής με τον ασθενή. Η εφαρμογή περιλαμβάνει το UC_contact.xaml, το οποίο περιέχει μία φόρμα επικοινωνίας. Ωστόσο, την υλοποίηση της διαδικασίας αποστολής email της πραγματοποιεί η custom κλάση MailOperation.cs. Επισημαίνεται, ότι στο στάδιο υλοποίησης της εν λόγω λειτουργίας, απαιτήθηκε η εγκατάσταση του προγράμματος hmailserver, το οποίο δημιουργεί έναν email Server προκειμένου να γίνει ο έλεγχος λειτουργικότητας της αποστολής/λήψης email. Παρακάτω παρατίθεται ο υπόψη κώδικας:

MailOperation.cs

```
...
namespace crystaleye.CustomClasses
{
    public class MailOperation
    {
        //FIELDS
        private MailAddress sender, receiver;
        private string subject;
        private string body;
        private string server;
```

```
        public MailOperation()
        {
        }

        public MailOperation(string messageSender, string messageReceiver, string
messageSubject, string messageBody)
        {
            //set server
            server = "mail.local";

            //set message metadata-data
            sender = new MailAddress(messageSender);
            receiver = new MailAddress(messageReceiver);
            subject = messageSubject;
            body = messageBody;
        }

        public bool send_mail()
        {
            // Create a message and set up the recipients.
            MailMessage message = new MailMessage();
            message.From = sender;
            message.To.Add(receiver);
            message.Subject = subject;
            message.Body = body;

            //Send the message.
            SmtpClient client = new SmtpClient(server);
            // Add credentials if the SMTP server requires them.
            client.Credentials = CredentialCache.DefaultNetworkCredentials;

            try
            {
                client.Send(message);
                return true;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Σφάλμα κατα την διαδικασία αποστολής Email():
{0}",
                            ex.ToString());
                return false;
            }
        }

        public static bool send_birth_mail(string receiver)
        {
            MailAddress admin = new MailAddress("eyecrystal@mail.local");
            string serv = "mail.local";
            // Create a message and set up the recipients.
            MailMessage message = new MailMessage();
            message.From = admin;
            message.To.Add(receiver);
            message.Subject = "Χρόνια Πολλά για τα Γενέθλια σας";
            message.Body = "Η CrystalEye σας εύχετε χρόνια πολλά για τα γενέθλια
σας με υγεία και ευτυχία!";

            //Send the message.
            SmtpClient client = new SmtpClient(serv);
```

```
            // Add credentials if the SMTP server requires them.
            client.Credentials = CredentialCache.DefaultNetworkCredentials;

            try
            {
                client.Send(message);
                return true;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Σφάλμα κατα την διαδικασία αποστολής Email():
{0}",
                            ex.ToString());
                return false;
            }
        }
    }
}
...
```

Αρχικά, υφίστανται 2 constructors, από τους οποίους ο δεύτερος χρησιμοποιεί arguments. Συγκεκριμένα, δέχεται ως παραμέτρους, τον αποστολέα, τον παραλήπτη, το θέμα, και το κείμενο του μηνύματος. Έπειτα, ορίζεται ο server, ο οποίος δημιουργήθηκε στο πρόγραμμα hmailserver.

Η αποστολή email υλοποιείται στην μέθοδο send_mail(). Δημιουργείται αντικείμενο τύπου MailMessage, στο οποίο παρέχονται κατάλληλες παράμετροι. Έπειτα απαιτείται η δημιουργία αντικειμένου SmtpClient, το οποίο περιέχει την μέθοδο send() κι έτσι πραγματοποιείται η αποστολή του μηνύματος. Εξυπακούεται, ότι η παραπάνω διαδικασία περιλαμβάνεται σε ένα try-catch block, το οποίο εμφανίζει κατάλληλο σφάλμα σε περίπτωση αποτυχίας αποστολής του email.

Πέραν των προαναφερθέντων, η κλάση MailOperation.cs περιέχει άλλη μία μέθοδο με όνομα send_birth_mail(). Με την μέθοδο αυτή, υλοποιείται η διαδικασία αποστολής mail σε ασθενή, με σκοπό η κλινική να του ευχηθεί για τα γενέθλια του. Η διαδικασία είναι αυτοματοποιημένη και απαιτεί μόνο το email του παραλήπτη-ασθενή, ως όρισμα.

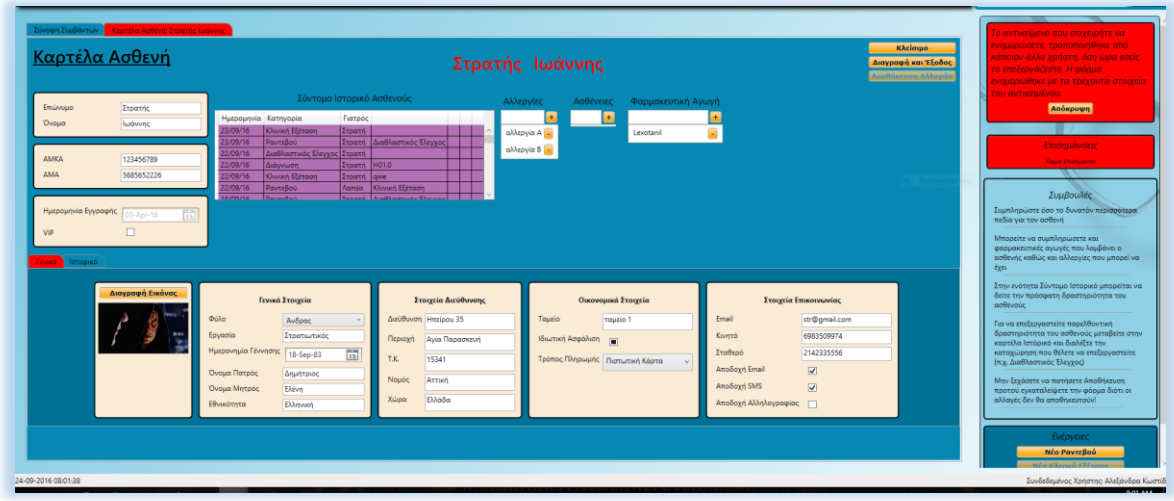
4.2.13 Υλοποίηση Concurrency

Έχει μεγάλο ενδιαφέρον να παρατηρήσουμε την συμπεριφορά της εφαρμογής σε θέματα concurrency, όπως και τον κώδικα που το υλοποιεί. Για αυτό τον λόγο, θεωρούμε το σενάριο στο οποίο υφίστανται δύο instances της εφαρμογής, όπου στο ένα είναι συνδεδεμένος ένας ιατρός και στο άλλο ένας γραμματέας. Θεωρούμε, ότι ταυτόχρονα και οι δύο χρήστες ανοίγουν την καρτέλα του ασθενή Στρατή Ιωάννη, προκειμένου να αλλάξουν την διεύθυνση του. Όπως είναι γνωστό πλέον, και στους δύο θα δημιουργηθεί ένα αντικείμενο context, το οποίο θα «φορτώσει» την οντότητα patient του ασθενή Στρατή Ιωάννη. Στην συνέχεια, θεωρούμε ότι ο ιατρός αλλάζει την διεύθυνση και αποθηκεύει τις αλλαγές, ενώ η γραμματεία πραγματοποιεί ακόμη αλλαγές στην καρτέλα του ίδιου ασθενούς. Τότε, τα δεδομένα στην ΒΔ θα ενημερωθούν μετά τις αλλαγές που εκτέλεσε ο ιατρός. Την στιγμή όμως που θα αποπειραθεί η γραμματεία να εκτελέσει αποθήκευση των αλλαγών, τότε έρχεται στην επιφάνεια θέμα concurrency των δεδομένων. Το EF το αντιλαμβάνεται, και εμφανίζει exception με κατάλληλο μήνυμα. Η εφαρμογή κάνει catch το exception αυτό και εμφανίζει με την σειρά του ένα φιλικό προς τον χρήστη μήνυμα, ενώ ταυτόχρονα εκτελεί ανανέωση των δεδομένων της καρτέλας του ασθενούς, προκειμένου να εμφανίζονται πάντα τα σωστά, τα οποία θα περιέχουν τις αλλαγές του ιατρού. Ο κώδικας που υλοποιεί τον έλεγχο concurrency καθώς και την ανανέωση του context είναι ο παρακάτω:

```
...
//kalw sinartisi gia apothikeusi kai enimerwsi dedomenwn stin vasi.
                Exception ex = StoreDB.StoreDBclinic.updateItem(context);
```

```
                if ( ex != null)
                {
                    if(ex is DbUpdateConcurrencyException)
                    {
                        var objectContext =
((IObjectContextAdapter)context).ObjectContext;
                        objectContext.Refresh(RefreshMode.StoreWins,
current_patient);
                        btn_save.IsEnabled = false;
                        dialogue.validated_Data("Το αντικείμενο που επιχειρήτε να
ενημερώσετε, τροποποιήθηκε από κάποιον άλλο χρήστη, όση ώρα εσείς το
επεξεργάζεστε. Η φόρμα ενημερώθηκε με τα τρέχοντα στοιχεία του αντικειμένου.");
                    }
                    else
                    {
                        dialogue.validated_Data("Ελέγξτε ξανά την ορθότητα των
δεδομένων που έχετε εισάγει");
                    }
                }
                else
                {
                    btn_save.IsEnabled = false;

                    dialogue.validated_Data("Οι αλλαγές αποθηκεύτηκαν με
επιτυχία");
                }
...
```

Ο παραπάνω κώδικας περιέχεται μέσα στην μέθοδο btn_save_click() και εκτελείται στο τελικό στάδιο, όπου ο χρήστης επιθυμεί να εκτελέσει αποθήκευση των δεδομένων. Έτσι, αρχικά εκτελείται η μέθοδος updateItem της StoreDB κλάσης, η οποία επιστρέφει αντικείμενο τύπου Exception. Σε περίπτωση που δεν υπάρχει exception, τότε εμφανίζεται μήνυμα στον χρήστη ότι η αποθήκευση έγινε με επιτυχία. Σε αντίθετη περίπτωση εμφανίζεται το μήνυμα του exception. Για την συγκεκριμένη περίπτωση που το exception είναι τύπου DUpdateConcurrencyException, εκτελούνται μία σειρά από διαδικασίες. Αρχικά, δημιουργείται αντικείμενο τύπου IObjectContextAdapter, με την βοήθεια του οποίου καθορίζονται οι παράμετροι για το RefreshMode, και που στην προκειμένη περίπτωση ορίζεται RefreshMode.StoreWins. Αυτό σημαίνει, ότι σε ενδεχόμενο concurrency, το context θα ανανεωθεί με δεδομένα από την ΒΔ καθώς αυτά θα θεωρηθούν έγκυρα. Τέλος, εμφανίζεται κατάλληλο μήνυμα στον χρήστη. Στην παρακάτω εικόνα φαίνεται η συμπεριφορά του προγράμματος όταν εμφανιστεί θέμα concurrency:



Εικόνα 33. Συμπεριφορά ΠΣ σε συμβάν Concurrency

Εκτός από την υλοποίηση των προαναφερθέντων λειτουργιών, το crystaleyeProject φιλοξενεί τον κώδικα για το consuming του Web Service, που σχετίζεται με την σύνδεση της εφαρμογής με εξωτερική Βάση Δεδομένων. Ωστόσο, επειδή ο κώδικας αυτός αποτελεί, ουσιαστικά, συνέχεια του κώδικα με τον οποίο υλοποιείται η εν λόγω υπηρεσία, θα αναλυθεί στο επόμενο κεφάλαιο, στο οποίο γίνεται αναφορά στις αρχές λειτουργίας του WCF Framework.

Σε αυτό το σημείο τελειώνει η ανάλυση κάποιων βασικών σημείων στον κώδικα του crystaleye Project. Επισημαίνεται, ότι στο υπόψη Project υπάρχει μεγάλος όγκος κώδικα, ο οποίος λειτουργεί βοηθητικά προς τον κώδικα που αναλύθηκε παραπάνω. Ωστόσο, δεν είναι δυνατή η εξέταση του συνόλου του κώδικα, για λόγους εξοικονόμησης χώρου και για αυτό τον λόγο υπάρχουν αρκετά σχόλια στα αρχεία του Project, προκειμένου ο αναγνώστης να μπορεί να ακολουθήσει την αλληλουχία των διαδικασιών και τον σκοπό του κάθε αλγορίθμου.

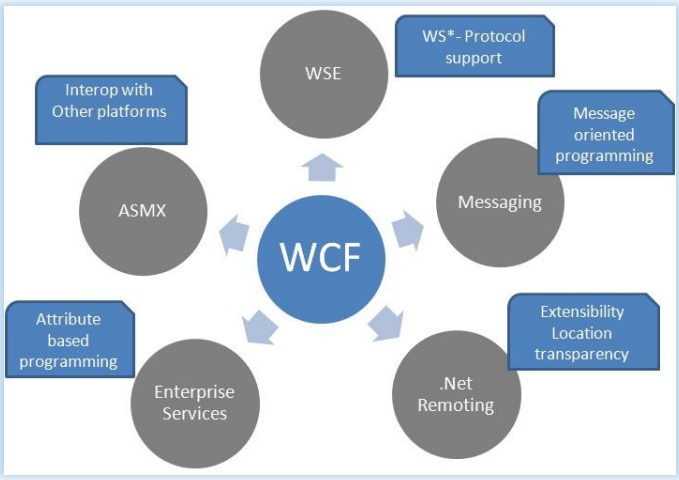
5 CrystalEyeWebService Solution

5.1 Γενικά

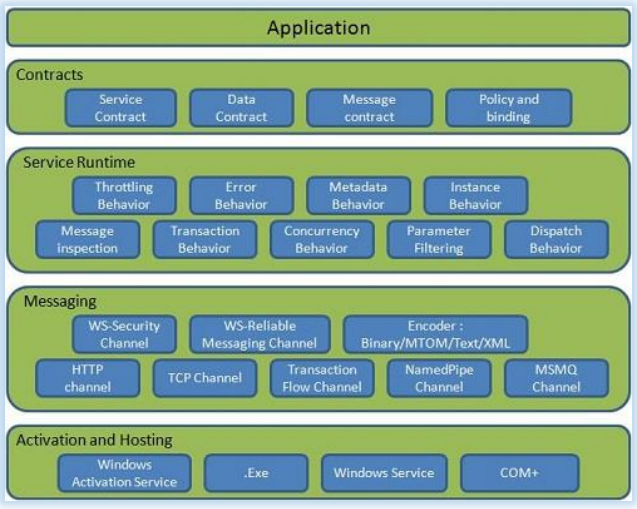
5.1.1 Windows Communication Foundation

Το Windows Communication Foundation (WCF) Framework αποτελεί μία προγραμματιστική πλατφόρμα για την δημιουργία, ρύθμιση και ανάπτυξη δικτυακών υπηρεσιών. Το κύριο χαρακτηριστικό του είναι η διαλειτουργικότητα. Αφορά ένα ενιαίο προγραμματιστικό μοντέλο, το οποίο ανήκει στο .NET Framework και συνδυάζει χαρακτηριστικά, όπως Web Service, απομακρυσμένη πρόσβαση κ.α.

Παρακάτω, παρατίθεται τα σχήματα, όπου φαίνονται οι διάφορες τεχνολογίες που συνδυάζει το WCF Framework καθώς και η αρχιτεκτονική που εφαρμόζει:



Εικόνα 34. Υπηρεσίες WCF Framework



Εικόνα 35. Αρχιτεκτονική WCF εφαρμογής

Το WCF εφαρμόζει μία αρχιτεκτονική multi-layer, όπως φαίνεται στην παραπάνω εικόνα. Συγκεκριμένα, περιλαμβάνει τα εξής:

α) Contracts Layer: Αφορά την συμφωνία μεταξύ δύο μηχανών, η οποία ορίζει τους όρους και προϋποθέσεις για τα μηνύματα που θα ανταλλάσσονται. Περιλαμβάνει τα εξής είδη contract:

i) Data Contract: Ορίζει την δομή των δεδομένων και τις παραμέτρους που χρησιμοποιούνται από τα services, προκειμένου να πραγματοποιηθεί επιτυχώς αλληλεπίδραση του service με τον client.

ii) Message Contract: Παρέχει έλεγχο στα μηνύματα SOAP που παράγονται και καταναλώνονται από το WCF. Επίσης, παρέχουν απευθείας πρόσβαση στο σώμα και τις επικεφαλίδες του μηνύματος.

iii) Service Contract: Ορίζει το είδος της λειτουργίας που υποστηρίζει η υπηρεσία. Εκθέτει πληροφορίες στον client, όπως ο τύπος δεδομένων που έχει το μήνυμα, η τοποθεσία που βρίσκονται οι μέθοδοι, οι πληροφορίες του πρωτοκόλλου που εφαρμόζεται και η δομή serialization κ.α. Ουσιαστικά, ορίζονται οι μέθοδοι της υπηρεσίας.

iv) Policy και Binding: Παρέχει πληροφορίες σχετικές με την ασφάλεια και το πρωτόκολλο.

β) Service Runtime Layer: Ορίζει και επεξεργάζεται διάφορες συμπεριφορές της υπηρεσίας, οι οποίες συμβαίνουν κατά την λειτουργία της υπηρεσίας. Ονομάζονται και service behaviors και ουσιαστικά αποτελούν αντικείμενα που επηρεάζουν τα χαρακτηριστικά της WCF υπηρεσίας.

γ) Messaging Layer: Είναι υπεύθυνο για την δομή των μηνυμάτων που ανταλλάσσονται.

δ) Hosting και Application Layer: Παρέχει μεγάλο αριθμό επιλογών σχετικά με την ενεργοποίηση και φιλοξενία της υπηρεσίας.

Πλεονεκτήματα

Τα πλεονεκτήματα του WCF Framework είναι τα εξής:

α) Διαλειτουργικότητα με άλλες υπηρεσίες .

β) Καλύτερη αξιοπιστία και ασφάλεια σε σύγκριση με τις Web Services ASMX

γ) Δεν απαιτούνται μεγάλες αλλαγές στον κώδικα, προκειμένου να εφαρμοστεί ένα μοντέλο ασφάλειας και τροποποίηση των απαιτούμενων συνδέσεων (bindings).

δ) Φέρει ενσωματωμένο μηχανισμό logging.

Βασικές Έννοιες

Οι πιο βασικές έννοιες του WCF Framework είναι οι εξής:

α) Message: Αποτελεί μία μονάδα επικοινωνίας, η οποία περιλαμβάνει διάφορα τμήματα εκτός από το κυρίως σώμα. Τα μηνύματα στέλνονται και λαμβάνονται για όλους τους τύπους επικοινωνίας, ανάμεσα σε client και service.

β) Endpoint: Ορίζει την διεύθυνση, όπου το μήνυμα στέλνεται ή λαμβάνεται. Επίσης, καθορίζει τον μηχανισμό επικοινωνίας για να καθορίσει τα μηνύματα πώς θα αποστέλλονται. Μία δομή endpoint περιλαμβάνει τα ακόλουθα τμήματα:

i) Address: Αποτελεί την ακριβή τοποθεσία, που πρόκειται να παραλάβει τα μηνύματα και ορίζεται ως Unified Resource Identifier (URI). Για παράδειγμα η

net.tcp://localhost:9000/ServiceA, αποτελεί μία διεύθυνση URI. Το κομμάτι «net.tcp», αφορά το σχήμα του πρωτοκόλλου TCP. Το domain είναι το «localhost», και το path είναι το «ServiceA».

ii) Binding: Ορίζει τον τρόπο επικοινωνίας ενός endpoint. Αποτελείται από στοιχεία σύνδεσης που δημιουργούν την υποδομή για την επικοινωνία και ορίζει το πρωτόκολλο επικοινωνίας που χρησιμοποιείται για την πρόσβαση στην υπηρεσία. Μερικά στοιχεία binding είναι τα basicHttpBinding, wsHttpBinding, wsDualHttpBinding κ.α.

iii) Contracts: Πρόκειται για μία συλλογή εργασιών, οι οποίες ορίζουν ποια λειτουργία θα εκθέτει το endpoint στον client.

γ) Hosting: Αναφέρεται στην φιλοξενία της WCF υπηρεσίας, η οποία μπορεί να πραγματοποιηθεί με ποικίλους τρόπους, όπως self-hosting, IIS hosting, WAS hosting κ.α.

δ) Metadata: Αυτή είναι μία σημαντική έννοια, δεδομένου ότι διευκολύνει την αλληλεπίδραση μεταξύ της εφαρμογής-client και της Web Service.

ε) WCF client: Αφορά μία εφαρμογή, η οποία δημιουργείται για να εκθέτει τις λειτουργίες WCF με την μορφή μεθόδων.

στ) Channel: Αφορά το μέσο στο οποίο ένας client επικοινωνεί με την υπηρεσία.

ζ) SOAP: Είναι ένα XML αρχείο, που αποτελείται από το σώμα και τις επικεφαλίδες.

Ένα WCF Service αποτελείται από ένα ή περισσότερα interfaces και τις κλάσεις που τα εφαρμόζουν. Τα interfaces περιλαμβάνουν τα contracts, όπως οριστήκαν παραπάνω. Στην συνέχεια, γίνεται η εφαρμογή των interfaces, όπου υλοποιείται με την χρήση μεθόδων, η λειτουργία που εκτελεί το Service. Στην συνέχεια, ορίζεται ο Host, που θα φιλοξενεί το Service και που θα το εκθέτει, προκειμένου να το εκμεταλλευτούν οι πιθανοί clients. Όλα τα απαιτούμενα βήματα για την επιτυχή δημιουργία ενός Web Service, θα εξηγηθούν με λεπτομέρεια κατά την ανάλυση του κώδικα των Projects.

5.1.2 Εισαγωγή - Δομή Solution

Η υπηρεσία, που αποτελεί απαίτηση στην εφαρμογή, αφορά στην δυνατότητα σύνδεσης της εφαρμογής με μία εξωτερική ΒΔ, η οποία ανήκει θεωρητικά στον ΕΟΟΠΥ. Συγκεκριμένα, όταν ο χρήστης της εφαρμογής επιθυμεί να εισάγει καινούριο ασθενή στο σύστημα, έχει την δυνατότητα να ελέγξει μέσω ΑΜΚΑ, εάν υπάρχει ο ασθενής αυτός στην ΒΔ του ΕΟΟΠΥ. Σε περίπτωση που υπάρχει, τότε η εφαρμογή προσφέρεται να συμπληρώσει αυτόματα όλα τα απαιτούμενα πεδία της καινούριας καρτέλας του ασθενή, με τις πληροφορίες που αντλεί από την ΒΔ του ΕΟΟΠΥ. Σε διαφορετική περίπτωση, θα πρέπει ο χρήστης να εισάγει μόνος του τα στοιχεία του ασθενή.

Για την υλοποίηση της παραπάνω λειτουργίας απαιτήθηκε η δημιουργία των εξής τεσσάρων Projects:

α) StoreDatabase Project,

β) WCFService Project

γ) WebServiceHost Project και

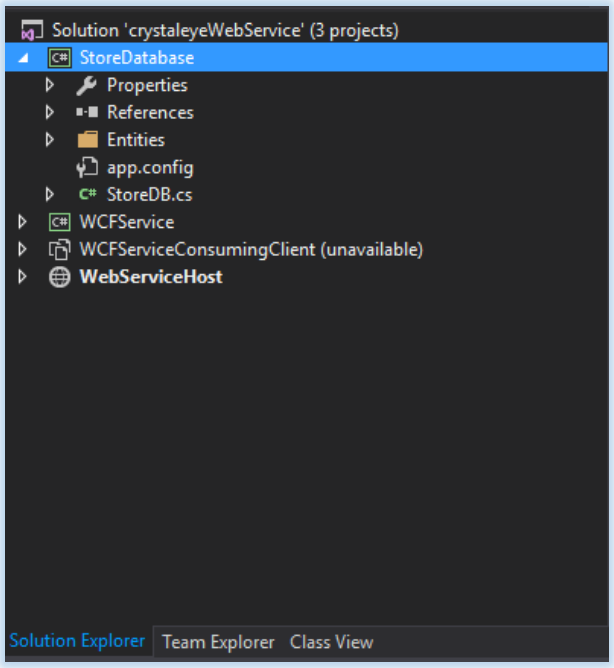
δ) WCFServiceConsumingClient Project

Το StoreDatabase Project αφορά τον κώδικα, που υλοποιεί την σύνδεση του Service με μία ΒΔ, η οποία υποθέτουμε ότι ανήκει στον ΕΟΟΠΥ. Έπειτα, το WCFService Project περιλαμβάνει τον κυρίως κώδικα, που υλοποιεί την κατασκευή του Service και τέλος στο WebServiceHost Project γίνεται το hosting. Το WCFServiceConsumingClient Project έχει δοκιμαστικό ρόλο και περιλαμβάνει κώδικα για την δοκιμή της λειτουργίας consuming του υπόψη Service και συνεπώς, δεν απαιτείται η ανάλυση του.

Στην συνέχεια, ακολουθεί η ανάλυση του κώδικα για τα τρία βασικά Projects, που αναφέρθηκαν προηγουμένως.

5.2 StoreDatabase Project

Το υπόψη Project αφορά στην σύνδεση της εφαρμογής με την εξωτερική ΒΔ και περιλαμβάνει αρχεία και φακέλους, όπως φαίνεται στην παρακάτω εικόνα:



Εικόνα 36. Δομή του StoreDatabase Project

Συγκεκριμένα, το Project περιλαμβάνει τα εξής:

α) Properies, app.config: Αφορούν αρχεία ρυθμίσεων, που σχετίζονται με το Project.

β) References: Αφορά τις απαιτούμενες συσχετίσεις του StoreDatabase Project με άλλα project και Frameworks, και επιτρέπει την χρήση συγκεκριμένων εντολών στον κώδικα.

γ) Entities: Περιλαμβάνει τις custom Οντότητες. Στην περίπτωση μας οι πληροφορίες του ατόμου επιστρέφονται με την μορφή Datatable. Ωστόσο, είναι δυνατή και η επιστροφή μέσω custom αντικειμένων, όπως μία Οντότητα.

δ) StoreDB.cs: Κλάση, η οποία περιλαμβάνει όλες τις μεθόδους με τις οποίες αντλούνται τα δεδομένα από την εξωτερική ΒΔ.

Σε αντίθεση με το crystaleye Solution, η επικοινωνία με την ΒΔ υλοποιείται με ADO.net commands προς ένα ODBC Data Source, και όχι με το Entity Framework. Ο λόγος για τον οποίο έγινε αυτή η διαφοροποίηση είναι για να παρουσιαστεί άλλη μία δυνατή επιλογή που παρέχεται στον προγραμματιστή, σε ότι αφορά την σύνδεση της εφαρμογής με την ΒΔ.

Η εξωτερική ΒΔ δημιουργήθηκε σε περιβάλλον MySQL με όνομα «eopyy_db». Περιλαμβάνει μόνο ένα πίνακα, τον «patient». Προφανώς, σε ρεαλιστικό επίπεδο, ο ΕΟΟΠΥ φέρει μία μεγάλη Βάση Δεδομένων, η οποία φιλοξενεί μεγάλη ποικιλία πληροφοριών. Ωστόσο, ο σκοπός στην παρούσα ΔΕ είναι να παρουσιαστεί ο τρόπος υλοποίησης της Web Service μέσω WCF και για αυτό απαιτείται η

ύπαρξη ενός μόνο πίνακα. Ο πίνακας αυτός, ονομάζεται «patient» και αποθηκεύει προσωπικά δεδομένα.

Παρακάτω παρατίθεται ένα απόσπασμα του κώδικα της κλάσης StoreDB.cs:

```
...
public class StoreDB
    {
        //FIELDS
        private static StoreDB storeDBclinic = new StoreDB();
        public static StoreDB StoreDBclinic
        {
            get { return storeDBclinic; }
        }
        private string conn_string =
Properties.Settings.Default.connectionstring;

        //METHODS
public DataTable return_patient(string AMKA)
        {
            //open the connection using the applications connectionstring
            OdbcConnection DbConnection = new OdbcConnection(conn_string);
            DbConnection.Open();
            //Query
            OdbcCommand querycat = new OdbcCommand("SELECT * FROM patient where
AMKA=?;", DbConnection);

            //Query parameters
            querycat.Parameters.Add("@AMKA", OdbcType.VarChar).Value = AMKA;

            //Adapter
            OdbcDataAdapter adaptercat = new OdbcDataAdapter(querycat);

            //Dataset
            DataSet ds = new DataSet();

            //gemisma tou adapter me ton pinaka Products
            adaptercat.Fill(ds, "Patient");

            DbConnection.Close();

            return ds.Tables[0];
        }
...
```

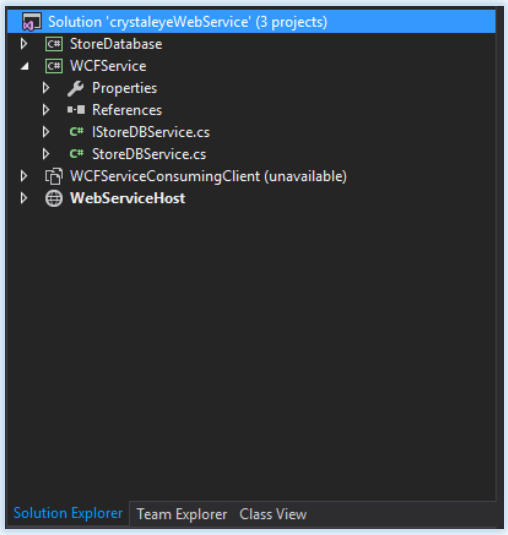
Είναι προφανές, ότι ακολουθείται η ίδια λογική με το crystaleye Solution, σε ότι αφορά την δημιουργία αντικειμένου τύπου StoreDB. Συγκεκριμένα, με την χρήση κατάλληλου κώδικα, είναι επιτρεπτή η δημιουργία ενός μόνο αντικειμένου τύπου StoreDB. Στην συνέχεια, αποθηκεύεται το connection string σε μία μεταβλητή.

Η μέθοδος return_patient(), έχει ως argument το ΑΜΚΑ του ατόμου και επιστρέφει ένα Datatable με τις πληροφορίες του από την ΒΔ, σε περίπτωση που υπάρχει άτομο με αυτό το ΑΜΚΑ. Συγκεκριμένα, δημιουργείται ένα αντικείμενο τύπου OdbcConnection με χρήση του connectionstring. Στην συνέχεια, ορίζεται το query που θα πρέπει να εκτελεστεί στην ΒΔ με την εντολή OdbcCommand. Έπειτα, ορίζονται οι παράμετροι του OdbcCommand, προκειμένου να εξαλειφθεί η περίπτωση SQL Injection. Τέλος, δημιουργείται το dataset και επιστρέφεται το datatable.

Η StoreDB περιλαμβάνει και άλλες μεθόδους, οι οποίες έχουν δοκιμαστικό χαρακτήρα. Στο επόμενο κεφάλαιο, αναλύεται ο κώδικας του WCFService Project.

5.3 WCFService Project

Το WCFService Project περιλαμβάνει τον απαιτούμενο κώδικα για τον ορισμό του απαιτούμενου service. Παρακάτω φαίνονται τα αρχεία, που περιλαμβάνει:



Εικόνα 37. Δομή του WCFService Project

Εκτός από τα γνωστά πλέον αρχεία που συνοδεύουν ένα Project (Properties και References), υπάρχουν 2 κλάσεις, οι IStoreDBService.cs και StoreDBService.cs. Το πρώτο αρχείο περιλαμβάνει τον κώδικα για την δημιουργία του contract, υπό την μορφή interface και το δεύτερο αρχείο την υλοποιεί. Συγκεκριμένα, ο κώδικας του IStoreDBService.cs είναι ο εξής:

```
using System;
using System.ServiceModel;
using System.Data;

namespace WCFService
{
    [ServiceContract]
    public interface IStoreDBService
    {
        [OperationContract]
        DataTable GetPatient(String AMKA);

    }
}
```

Αρχικά, είναι εμφανής η χρήση attributes σε διάφορα σημεία του κώδικα δημιουργίας του υπόψη interface. Το ServiceContract περιγράφει τις λειτουργίες της υπηρεσίας, που είναι διαθέσιμες στο endpoint και εκτίθενται στον έξω κόσμο. Ουσιαστικά, περιγράφει τις μεθόδους, τις οποίες μπορεί να καλέσει ένας client.

Γενικά, για να οριστεί ένα service contract απαιτείται ο ορισμός ενός Interface με μεθόδους, οι οποίες αντιπροσωπεύουν τις υπηρεσίες και στην συνέχεια η προσθήκη του attribute ServiceContract, το οποίο υποδεικνύει στην εφαρμογή ότι αφορά ένα service contract. Οι μέθοδοι που περιλαμβάνει το interface και το service contract θα πρέπει να φέρουν το attribute

OperationContract. Στην συγκεκριμένη περίπτωση το service contract, που έχει οριστεί, είναι το IStoreDBService interface, το οποίο με την σειρά του ορίζει την υπηρεσία GetPatient.

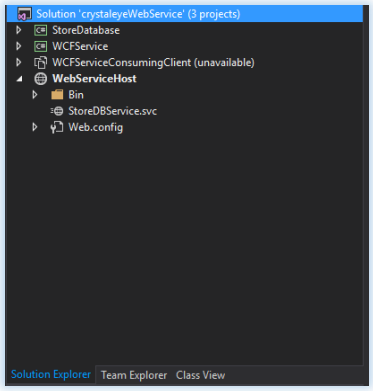
Στην συνέχεια, και αφού έχει οριστεί το interface (service contract) της επιθυμητής υπηρεσίας, απαιτείται η δημιουργία μίας κλάσης, που θα εφαρμόζει το interface. Αυτή η κλάση είναι η StoreDBService, και περιλαμβάνει τον παρακάτω κώδικα:

```
using StoreDatabase;
using System.Data;

namespace WCFService
{
    public class StoreDBService : IStoreDBService
    {
        public DataTable GetPatient(string AMKA)
        {
            //klisi stin methodo return_patient tis StoreDB
            return StoreDB.StoreDBclinic.return_patient(AMKA);
        }
    }
}
```

Η κλάση ονομάζεται StoreDBService και κληρονομεί από το interface IStoreDBService, που είναι και το service contract. Συνεπώς, όπως σε κάθε περίπτωση interface inheritance, απαιτείται η εφαρμογή των μεθόδων του Interface. Στην προκειμένη περίπτωση, εφαρμόζεται η μοναδική μέθοδος GetPatient() και πλέον αποκτάει λειτουργία. Συγκεκριμένα, εκτελεί την μέθοδο return_patient() της κλάσης StoreDB, που ανήκει στο StoreDatabase Project. Η GetPatient() επιστρέφει ένα Datatable και στα επόμενα κεφάλαια θα εξεταστεί ο κώδικας που επιτρέπει το consuming της υπόψη υπηρεσίας.

5.4 WebServiceHost Project

Το WebServiceHost Project αφορά το hosting του service που δημιουργήθηκε προκειμένου να είναι εκτεθειμένο σε πιθανούς clients. Τα αρχεία που περιλαμβάνει φαίνονται στην παρακάτω εικόνα:



Εικόνα 38. Δομή WebServiceHost Project

Το Project δημιουργήθηκε σε περιβάλλον ASP.net, συνεπώς το hosting γίνεται με την βοήθεια του IIS. Όπως είναι εμφανές, το Project περιλαμβάνει λίγα αρχεία. Το αρχείο όμως, που

ουσιαστικά υλοποιεί το hosting του service είναι το StoreDBService.svc. σε συνδυασμό φυσικά με κάποιες ιδιότητες που προστέθηκαν στο web.config αρχείο ρυθμίσεων.

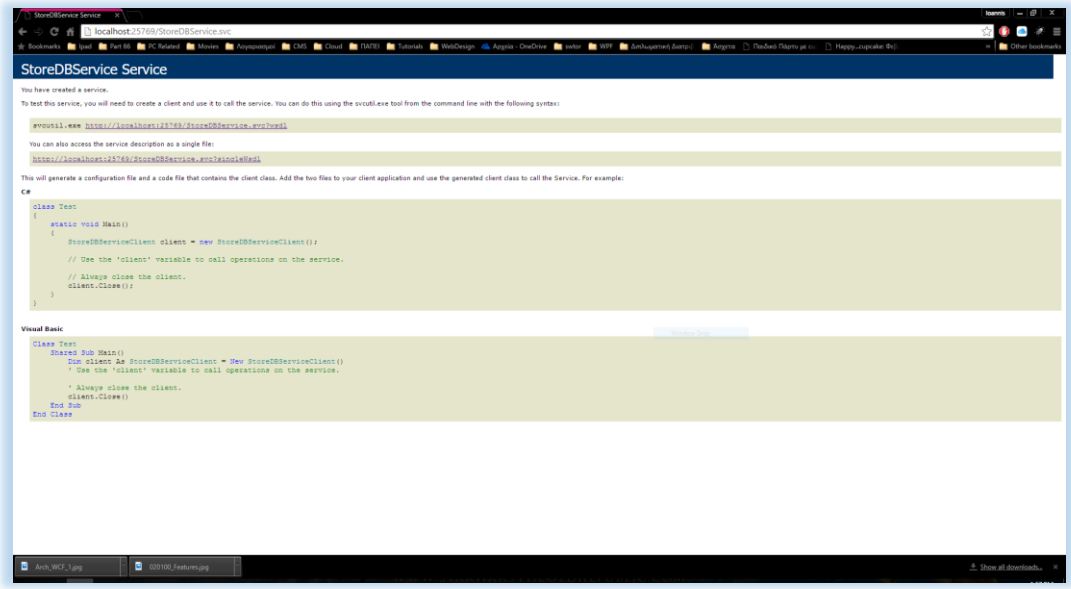
Στις περιπτώσεις που το Hosting γίνεται μέσω IIS, απαιτείται το αρχείο τύπου .svc., το οποίο χειρίζεται ο IIS. Στην περίπτωση μας το αρχείο περιλαμβάνει μία μόνο γραμμή κώδικα, την εξής:

```
<%@ServiceHost Service="WCFService.StoreDBService"%>
```

Στην συνέχεια, παρατίθεται ο κώδικας από το αρχείο web.config, με το οποίο ολοκληρώνεται το hosting της υπηρεσίας:

```
<?xml version="1.0"?>

<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=169433
  -->

<configuration>

    <system.web>
      <compilation debug="true" targetFramework="4.5.2" />
      <httpRuntime targetFramework="4.5.2" />
    </system.web>
 <system.webServer>
    <modules runAllManagedModulesForAllRequests="true"/>
    <directoryBrowse enabled="true" />
  </system.webServer>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="MyServiceTypeBehaviors">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="WCFService.StoreDBService"
behaviorConfiguration="MyServiceTypeBehaviors">
        <endpoint address="" binding="wsHttpBinding"
contract="WCFService.IStoreDBService"/>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Το βασικό στοιχείο, που αφορά την υπηρεσία είναι το system.serviceModel. Συγκεκριμένα, ορίζεται ένα endpoint για την υπηρεσία, το οποίο περιλαμβάνει τις ιδιότητες address, binding και contract (ABC). Επειδή, η υπηρεσία γίνεται Host τοπικά, η ιδιότητα address στο endpoint είναι κενή. Εάν το host υλοποιούνταν απομακρυσμένα, τότε η ιδιότητα address θα περιλάμβανε την διεύθυνση του αρχείου svc. Η τιμή wsHttpBinding στο attribute binding, προσδιορίζει ένα ασφαλή και αξιόπιστο είδος binding για non-dublex service contracts. Τέλος, ως contract τίθεται το interface IstoreDBService. Παρακάτω παρατίθεται screenshot από το επιτυχές hosting της υπηρεσίας:



Εικόνα 39. Επιτυχές WCF Service hosting

Πλέον, έχει ολοκληρωθεί το hosting της υπηρεσίας. Απομένει η εξέταση του κώδικα που αφορά την κατανάλωση της υπηρεσίας από το ΠΣ.

5.5 WebService Consuming

Για την ανάλυση του τρόπου εκμετάλλευσης της υπηρεσίας από την εφαρμογή, θα πρέπει να γίνει εκ νέου αναφορά στο crystaleye Solution και συγκεκριμένα στο crystaleye Project. Συγκεκριμένα, ο μοναδικός φάκελος του Project, που δεν εξετάστηκε σκοπίμως στην ανάλυση του crystaleye Project, ήταν ο WCFDB_Service, ο οποίος περιλαμβάνει τα εξής δύο αρχεία:

α) output.config: αποτελεί το configuration αρχείο προκειμένου να είναι εφικτό το consuming της υπηρεσίας.

β) StoreDBService.cs: περιλαμβάνει τον κώδικα, που υλοποιεί το consuming της υπηρεσίας από την εφαρμογή. Δημιουργείται αυτόματα από το Visual Studio.

Στην συνέχεια παρατίθεται ο κώδικας, που περιλαμβάνει το αρχείο output.config:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.serviceModel>
        <bindings>
            <wsHttpBinding>
                <binding name="WSHttpBinding_IStoreDBService" />
            </wsHttpBinding>
        </bindings>
        <client>
            <endpoint address="http://localhost:25769/StoreDBService.svc"
                binding="wsHttpBinding"
bindingConfiguration="WSHttpBinding_IStoreDBService"
                contract="ServiceReference.IStoreDBService"
name="WSHttpBinding_IStoreDBService">
                <identity>
                    <userPrincipalName value="FOVOSASUS\Fovos" />
```

```
                </identity>
            </endpoint>
        </client>
    </system.serviceModel>
</configuration>
```

Είναι εμφανές ότι ορίζονται σε γενικές γραμμές τα ίδια attributes, που είχαν οριστεί και στο web.config αρχείο στο Project που αφορούσε το hosting. Συγκεκριμένα, ορίζεται το endpoint με address την διεύθυνση του svc αρχείου, το contract με τιμή το interface (service contract) και κάποια επιπλέον στοιχεία.

Το στοιχείο identity αφορά στην ταυτότητα που πρόκειται να κάνει consume την υπηρεσία και λαμβάνει σημαντικό ρόλο κατά την διάρκεια handshaking μεταξύ του client και του service. Η δομή του WCF επιβεβαιώνει ότι οι ιδιότητες της αναμενόμενης υπηρεσίας ταιριάζουν με αυτές του identity στοιχείου και συνεπώς μπορεί να γίνει αυθεντικοποίηση.

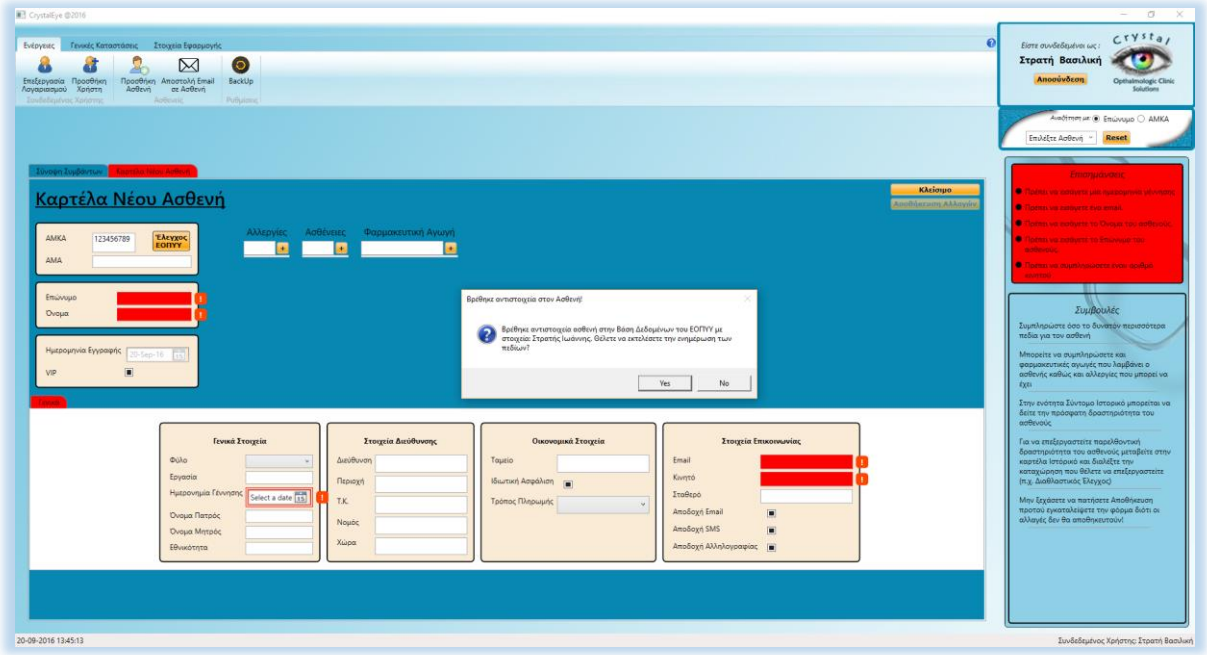
Στην συνέχεια, παρατίθεται η μέθοδος btn_check_Service_Click(), που ανήκει στο Usercontrol UC_patient_NEW, και ενεργοποιεί το consuming της υπηρεσίας:

```
...
private void btn_check_Service_Click(object sender, RoutedEventArgs e)
        {
            DataTable patient = new DataTable();
            StoreDBServiceClient client = new StoreDBServiceClient();

            if (txt_AMKA.Text.Length > 0)
            {
                try
                {
                    patient = client.GetPatient(txt_AMKA.Text);
                }
                catch(Exception ex)
                {
                    MessageBox.Show(ex.InnerException.ToString(), "Σφάλμα",
MessageBoxButton.OK, MessageBoxImage.Error);
                    return;
                }


                if (patient.Rows.Count>0)
                {
                    MessageBoxResult result = MessageBox.Show("Βρέθηκε
αντιστοιχεία ασθενή στην Βάση Δεδομένων του ΕΟΠΥΥ με στοιχεία:
"+patient.Rows[0]["surname"]+" "+patient.Rows[0]["name"]+". Θέλετε να εκτελέσετε
την ενημέρωση των πεδίων?", "Βρέθηκε αντιστοιχεία στον Ασθενή!",
MessageBoxButton.YesNo, MessageBoxImage.Question);
                    if (result == MessageBoxResult.Yes)
                    {
                        fill_patient_with_Data_from_eopi(patient);
                    }

                }
                else
                {
                    MessageBox.Show("Δεν βρέθηκε ασθενής στην Βάση Δεδομένων του
ΕΟΟΠΥΥ");
                    return;
                }
            }
            else
            {
```

```
                MessageBox.Show("Συμπληρώστε πρώτα το ΑΜΚΑ προκειμένου να γίνει ο
έλεγχος με την Βάση Δεδομένων του ΕΟΠΥΥ");
            }

        }
...
```

Αρχικά, δημιουργείται ένα αντικείμενο τύπου Datatable και ένα αντικείμενο τύπου StoreDBServiceClient. Η κλάση StoreDBServiceClient περιλαμβάνεται στο αρχείο StoreDBService.cs. και δίνει πρόσβαση τις μεθόδους του service contract. Συγκεκριμένα, με την εντολή patient = client.GetPatient(txt_AMKA.Text);, αποθηκεύεται στο αντικείμενο patient, το datatable που επιστρέφει το OperationContract, που έχει δημιουργηθεί. Έπειτα, υλοποιείται κώδικας, ο οποίος ενημερώνει τον χρήστη εάν βρέθηκε χρήστης με το συγκεκριμένο ΑΜΚΑ και προτρέπει να ενημερώσει τα πεδία του νέου χρήστη αυτόματα.

Παρακάτω παρατίθεται screenshot, η οποία αναδεικνύει την επιτυχή λειτουργία της υπηρεσίας που δημιουργήθηκε. Αρχικά, ο χρήστης ανοίγει την καρτέλα δημιουργίας νέου ασθενή, έπειτα εισάγει ένα ΑΜΚΑ και τέλος πατάει το κουμπί «Έλεγχος ΕΟΟΠΥ»:



Εικόνα 40. Απεικόνιση επιτυχούς λειτουργίας Web Service

6 Συμπεράσματα – Μελλοντικές Επεκτάσεις

Στην παρούσα Διπλωματική Εργασία έγινε μία προσπάθεια κατασκευής της ραχοκοκαλιάς ενός Ιατρικού Πληροφοριακού Συστήματος, αξιοποιώντας τις καινούριες προγραμματιστικές τάσεις και τεχνολογίες. Δόθηκε περισσότερο έμφαση στον τρόπο εκμετάλλευσης και εφαρμογής των νέων αυτών τεχνολογιών, αναδεικνύοντας τα οφέλη, τόσο από την οπτική γωνία του δημιουργού της εφαρμογής, όσο και από την εμπειρία χρήσης της, που προσφέρεται στον τελικό χρήστη. Η χρήση των Frameworks που χρησιμοποιήθηκαν και συγκεκριμένα τα WPF, WCF και Entity, διαφοροποιούν σε μεγάλο βαθμό την προγραμματιστική προσέγγιση της σχεδίασης, ανάλυσης και υλοποίησης της εφαρμογής, συγκριτικά με τις αντίστοιχες προγραμματιστικές μεθόδους του παρελθόντος.

Πρόκειται για ένα Πληροφοριακό Σύστημα, το οποίο λειτουργεί υπό την μορφή client-server και συνεπώς απαιτεί την ύπαρξη ενός διακομιστή, ο οποίος θα φιλοξενεί την Βάση Δεδομένων. Η εφαρμογή θα είναι εγκατεστημένη σε workstations του προσωπικού της κλινικής και αναλόγως της θέσεως που στελεχώνουν, φέρουν και διαφορετικά δικαιώματα χρήσης των λειτουργιών της εφαρμογής.

Αρχικά, δόθηκε έμφαση στην σωστή κατασκευή του σχήματος της Βάσης Δεδομένων, εφαρμόζοντας τους κανόνες Boyce-Codd, προκειμένου να εξαλείψουμε, αφενός μεν, την πιθανότητα άσκοπης επανάληψης πληροφορίας, αφετέρου δε, την περίπτωση απώλειας δεδομένων. Η σωστή δημιουργία μίας Βάσης Δεδομένων, απαιτεί την εφαρμογή όλων εκείνων των προβλεπόμενων κανόνων που την διέπουν, προκειμένου η υλοποίηση της εφαρμογής να έχει τα απαραίτητα εχέγγυα να χαρακτηριστεί ως αξιόπιστη.

Στην συνέχεια, εκμεταλλευτήκαμε τα οφέλη που παρέχει το Entity Framework. Πλέον, έχουμε ένα μοντέλο, το οποίο απεικονίζει την Βάση Δεδομένων, και είναι προγραμματιστικά διαχειρίσιμο και εκμεταλλεύσιμο. Συνεπώς, αποφεύγονται άσκοπα σφάλματα του παρελθόντος, όπου απαιτούνταν ο απευθείας χειρισμός της Βάσης Δεδομένων από τον κώδικα, προκειμένου να εκτελεστούν βασικές SQL ενέργειες. Πλέον, ο προγραμματιστής δεν ασχολείται με τεχνικά στοιχεία της σύνδεσης της Βάσης Δεδομένων με την εφαρμογή. Αντίθετα, χειρίζεται Οντότητες, οι οποίες αποτελούν μία απεικόνιση της Βάσης Δεδομένων. Επιπρόσθετα, η αντικειμενοστραφής προσέγγιση, με την οποία σχεδιάστηκε το παρόν ΠΣ, επεκτείνεται και στην υλοποίηση των λειτουργιών που αφορούν την Βάση Δεδομένων, παρέχοντας όλα τα οφέλη, τα οποία φέρει ο αντικειμενοστραφής προγραμματισμός.

Έπειτα, παρουσιάστηκε η ιδιαίτερη φιλοσοφία ανάπτυξης λογισμικού που προσφέρει το WPF Framework. Συγκεκριμένα, η σχεδίαση του GUI μίας εφαρμογής διαχωρίζεται πλήρως από την υλοποίηση των λειτουργιών της και γενικότερα της συμπεριφοράς της. Συνεπώς, είναι δυνατό να υφίστανται 2 ομάδες ανθρώπων, οι οποίοι να εργάζονται πάνω στην ίδια εφαρμογή ταυτόχρονα, επισπεύδοντας την ολοκλήρωση και τελική παράδοση της εφαρμογής. Σε ότι αφορά την γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η C#. Η γλώσσα C# είναι αυτή τη στιγμή μία από τις κορυφαίες γλώσσες, που αντιπροσωπεύουν τον αντικειμενοστραφή προγραμματισμό και εφαρμόζει λειτουργίες, που πραγματικά διευκολύνουν την δημιουργία αξιόπιστων εφαρμογών, μερικές από τις οποίες είναι οι εξής:

α) Η εφαρμογή μεταγλωττίζεται σε μία ενδιάμεση γλώσσα (CIL), ανεξαρτήτου της γλώσσας ανάπτυξης της εφαρμογής ή την αρχιτεκτονική του λειτουργικού συστήματος, στο οποίο θα λειτουργήσει.

β) Αυτόματος garbage collector.

γ) Δεν υπάρχει πλέον απαίτηση για χρήση pointers,

δ) Ο ορισμός των κλάσεων και των μεθόδων πραγματοποιείται με οποιαδήποτε σειρά,

ε) Κλάσεις μπορούν να οριστούν μέσα σε άλλες κλάσεις,

στ) Χρήστη get-set μεθόδων,

ζ) Περισσότερο καθαρή διαχείριση των events, με την χρήση delegates, κ.α.

Η ανάπτυξη λογισμικού σε περιβάλλον WPF σε συνδυασμό με τα οφέλη της C#, προσδίδει αξιοπιστία στο λογισμικό, προϋποθέτοντας ότι ακολουθήθηκαν βασικές προγραμματιστικές αρχές.

Στην συνέχεια, παρουσιάστηκαν τα οφέλη του WCF Framework, με αφορμή την απαίτηση σύνδεσης της εφαρμογής με εξωτερική Βάση Δεδομένων. Τα συμπεράσματα είναι άκρως αισιόδοξα καθώς με την συγκεκριμένη τεχνολογία είναι δυνατή η ανταλλαγή πληροφοριών δύο ανεξαρτήτων συστημάτων. Εάν συνυπολογίσουμε το γεγονός, ότι διαχειριζόμαστε Ιατρική πληροφορία, τότε τα οφέλη είναι τεράστια. Συγκεκριμένα, μεγάλα νοσοκομεία, κλινικές και ιατρεία μπορούν να ανταλλάσσουν πληροφορίες μεταξύ τους αλλά και με μία κεντρική Βάση Δεδομένων, η οποία θα ελέγχεται σε κρατικό επίπεδο. Επιπρόσθετα, οφέλη θα έχει και ο ίδιος ο ασθενής αφού με αυτό τον τρόπο θα μπορεί να έχει πρόσβαση ανα πάσα στιγμή στο δικό του ιατρικό ιστορικό, εκτελώντας ερωτήματα σε αυτή την κεντρική Βάση Δεδομένων. Έτσι, έχουμε την διαχείριση πληροφορίας, η οποία:

α) είναι διαβαθμισμένη,

β) δεν γίνεται να χαθεί,

γ) ολοκληρώνει την άποψη ενός ιατρού, οδηγώντας σε πιο ακριβείς διαγνώσεις και συμπεράσματα κ.α.

Συνεπώς, η χρήση Web Services στις Ιατρικές Εφαρμογές, μόνο ευεργετικά μπορεί να λειτουργήσει στην υγειονομική περίθαλψη ενός κράτους.

Δεδομένου ότι η εφαρμογή που αναπτύχθηκε αποτελεί αντικείμενο Διπλωματικής Εργασίας και γνωρίζοντας ότι η ανάπτυξη ενός Ολοκληρωμένου Πληροφοριακού Συστήματος επιβάλλει την συνεργασία πολλών ατόμων, εξάγεται το συμπέρασμα ότι η παρούσα εργασία μπορεί να αποτελέσει σημείο έναρξης για προσθήκη και άλλων υπηρεσιών ή εμπλουτισμού των ήδη υπαρχουσών.

Συγκεκριμένα, θα μπορούσε να προστεθεί ένα ολοκληρωμένο module, το οποίο να διαχειρίζεται τις οικονομικές συναλλαγές των ασθενών καθώς και τις οικονομικές υποχρεώσεις της κλινικής. Αυτομάτως, μία τέτοια αναβάθμιση θα πρόσθετε επιπλέον στατιστικά δεδομένα προς επεξεργασία από το προσωπικό της κλινικής. Επιπλέον, θα απαιτούνταν η δημιουργία νέων αναφορών, όπως εξαγωγή απόδειξης ή τιμολογίου. Ωστόσο, είναι προφανές ότι η εν λόγω προσθήκη θα επηρέαζε και την υφιστάμενη Βάση Δεδομένων, αφού θα απαιτούσε την δημιουργία νέων πινάκων χωρίς ωστόσο την τροποποίηση των ήδη υπαρχόντων. Τέλος, προκειμένου η πρόβλεψη των οικονομικών δραστηριοτήτων της κλινικής να είναι ολοκληρωμένη, επιβάλλεται και η προσθήκη ανάλογων λειτουργιών και στον υπάρχοντα Ιστότοπο της κλινικής.

Στην παρούσα ΔΕ, το Web Service, το οποίο ανακτήθηκε αφορούσε στην αυτοματοποίηση εισαγωγής των δημογραφικών στοιχείων ενός ασθενή, με έγκυρα δεδομένα από μία εξωτερική Βάση Δεδομένων. Συνεπώς, θα μπορούσε να υλοποιηθεί μία επέκταση της υπόψη λειτουργίας και σε ανταλλαγή πληροφοριών με ιατρικό περιεχόμενο και όχι μόνο. Συγκεκριμένα, η εξωτερική Βάση Δεδομένων θα μπορούσε να αποθηκεύει το βασικό ιατρικό ιστορικό ενός ασθενή, στο οποίο να έχει πρόσβαση ο οφθαλμίατρος μέσα στο ΠΣ. Επιπλέον, θα μπορούσε να δημιουργηθεί μία ροή πληροφορίας με κατεύθυνση προς την κεντρική Βάση Δεδομένων, παρέχοντας δεδομένα, όπως εισιτήριο/εξιτήριο ασθενή, προσθήκη οφθαλμολογικής πάθησης κ.α., εμπλουτίζοντας έτσι την ιατρική πληροφορία του ασθενούς. Μία τέτοια προσθήκη, προϋποθέτει την διεύρυνση της υφιστάμενης εξωτερικής Βάσης Δεδομένων.

Επιπλέον, θα ήταν δυνατό να προβλεφθεί εσωτερική επικοινωνία του προσωπικού της κλινικής μεταξύ τους μέσω chat, και να υλοποιείται εντός της εφαρμογής. Η λειτουργία αυτή με την βοήθεια του WCF Framework είναι εφικτό να υλοποιηθεί και λύνει επιπλέον αντικειμενικά προβλήματα που υφίστανται στην λειτουργία μίας κλινικής. Συγκεκριμένα, είναι γνωστό ότι συνήθως η γραμματειακή υποστήριξη της κλινικής είναι σε ξεχωριστό γραφείο από αυτό των ιατρών και των

εξεταστηρίων , όπως επίσης και ότι υπάρχει διαρκώς η απαίτηση για τηλεφωνική επικοινωνία μεταξύ του προσωπικού κατά την διεκπεραίωση των καθημερινών υποχρεώσεων της κλινικής. Συνεπώς, μία τέτοια προσθήκη θα μείωνε την άσκοπη χρήση της τηλεφωνικής επικοινωνίας εντός της κλινικής ενώ η αμεσότητα στην ανταλλαγή πληροφοριών που προσφέρει η λειτουργία chat, θα ευνοούσε την αυτοματοποίηση κάποιων εκ των καθημερινών λειτουργιών της κλινικής.

Επίσης, θα μπορούσε να δοθεί έμφαση στο κομμάτι εμπειρίας του χρήστη με το γραφικό περιβάλλον του ΠΣ (GUI). Στην παρούσα ΔΕ, δόθηκε έμφαση στην διατήρηση ενός απλού και λιτού UI. Ωστόσο, η φιλοσοφία που συνοδεύει την δημιουργία εφαρμογών με το WPF Framework, με τα προτερήματα που αναφέρθηκαν σε προηγούμενα κεφάλαια, δίνει πολλές επιλογές στην παραμετροποίηση του γραφικού περιβάλλοντος της εφαρμογής. Για τον λόγο αυτό, θα μπορούσε κάποιος να επωφεληθεί από το πακέτο Blend, με το οποίο μπορεί να σχεδιαστεί ένα άκρως επαγγελματικό UI.

Τέλος, θα μπορούσε να γίνει επέκταση των υφιστάμενων λειτουργιών που προσφέρει ο υπάρχον Ιστότοπος-portal της κλινικής ή και απεικόνιση μερικών λειτουργιών του ΠΣ σε περιβάλλον mobile.

Εν κατακλείδι, το κύριο μέλημα του δημιουργού της παρούσας ΔΕ ήταν η ενσωμάτωση μερικών καινούριων τεχνολογιών στην ανάπτυξη μίας εφαρμογής, η οποία να διαχειρίζεται ευαίσθητα δεδομένα και η αυτοματοποίηση μερικών βασικών διαδικασιών που ακολουθεί μία οφθαλμολογική κλινική στις καθημερινές υποχρεώσεις της.

Λαμβάνοντας υπόψη όλα τα παραπάνω, η παρούσα ΔΕ αποτελεί παρακαταθήκη για το Πανεπιστήμιο Πειραιώς και σημείο έναρξης για προσθήκη επιπλέον λειτουργιών στο Πληροφοριακό Σύστημα που αναπτύχθηκε.

7 Εγκατάσταση του Συστήματος

Η εγκατάσταση του ΠΣ δύναται να λάβει χώρα σε υπολογιστή, ο οποίος απαιτείται να έχει τουλάχιστον τα παρακάτω χαρακτηριστικά:

α) Εγκατεστημένο Λειτουργικό Σύστημα το Microsoft Windows 7 ή μεταγενέστερες εκδόσεις

β) Εγκατεστημένο το .Net Framework 4.5 ή μεταγενέστερες εκδόσεις

Σε ότι αφορά την Βάση Δεδομένων, απαιτείται η δημιουργία ενός MySQL Server. Το απαιτούμενο λογισμικό, είναι διαθέσιμο στην διεύθυνση http://dev.mysql.com/downloads, και προτείνεται η λήψη και εγκατάσταση της Community Edition που παρέχει η MySQL. Στην συνέχεια, θα πρέπει να δημιουργηθεί το σχήμα της Βάσης Δεδομένων στον server και για να γίνει αυτό χρησιμοποιούμε την εντολή Data Import/Restore στην κατηγορία Management. Στην συνέχεια, επιλέγουμε το αρχείο crystaleye.sql και γίνεται το restore της Βάσης Δεδομένων. Κατά τον ίδιο τρόπο επιλέγουμε το αρχείο eopyy.sql, προκειμένου να γίνει restore της εξωτερικής Βάσης Δεδομένων.

Στην συνέχεια απαιτείται η εγκατάσταση της εφαρμογής hmailserver, η οποία δημιουργεί τοπικά έναν mail server. Η εφαρμογή είναι διαθέσιμη προς λήψη από την διεύθυνση https://www.hmailserver.com/download. Μετά την εγκατάσταση της, προχωράμε στην δημιουργία νέου domain με όνομα mail.local και τέλος δημιουργούμε τα accounts, τα οποία θα μπορούν να στείλουν και να λάβουν email. Επισημαίνεται, ότι τα συγκεκριμένα accounts θα πρέπει να αντιστοιχούν είτε σε προσωπικό είτε σε ασθενείς της κλινικής και για αυτό τον λόγο θα πρέπει να ενημερωθούν οι σχετικές εγγραφές στην Βάση Δεδομένων.

8 Παράρτημα Α

Παρακάτω παρατίθενται οι εντολές SQL για την δημιουργία των πινάκων της ΒΔ «eyecrystal»:

ΠΙΝΑΚΑΣ “appointment”

```
DROP TABLE IF EXISTS `appointment`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `appointment` (
  `AMKA` varchar(10) NOT NULL,
  `Appointment_Date` date NOT NULL,
  `doctor` varchar(10) DEFAULT NULL,
  `Appointment_Time` time NOT NULL,
  `Appointment_Type` varchar(45) NOT NULL,
  `Appointment_Tmima` varchar(45) DEFAULT NULL,
  `Status` bit(1) DEFAULT b'0' COMMENT '0: εκκρεμεί\n1: ολοκληρώθηκε',
  `Date_Submitted` datetime DEFAULT CURRENT_TIMESTAMP,
  `Concurrency` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`AMKA`,`Appointment_Date`),
  UNIQUE KEY `custom1` (`AMKA`,`Appointment_Date`,`Appointment_Time`) COMMENT 'Ένας ασθενής μπορεί να έχει ραντεβού μία φορά για συγκεκριμένο συνδυασμό ημερομηνίας και ώρας.',
  UNIQUE KEY `custom2` (`doctor`,`Appointment_Date`,`Appointment_Time`) COMMENT 'Ένας ιατρός μπορεί να έχει ραντεβού μια φορά για συγκεκριμένο συνδυασμό ημερομηνίας και ώρας',
  KEY `fcappdoc_idx` (`doctor`),
  KEY `fkapp2_idx` (`Appointment_Type`),
  CONSTRAINT `fkapp1` FOREIGN KEY (`AMKA`) REFERENCES `patient` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fkapp2` FOREIGN KEY (`Appointment_Type`) REFERENCES `z_appointment_type` (`Appointment_Name`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fkappdoc` FOREIGN KEY (`doctor`) REFERENCES `doctor` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “clinical_examination”

```
DROP TABLE IF EXISTS `clinical_examination`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `clinical_examination` (
  `AMKA` varchar(10) NOT NULL,
  `Examination_Date` date NOT NULL,
  `doctor` varchar(10) NOT NULL,
  `Examination_Place` varchar(45) DEFAULT NULL,
```

```
  `Reexamined` bit(1) DEFAULT b'0',
  `Aitia_Eisodou` varchar(100) DEFAULT NULL,
  `Vasiki_Vlavi` varchar(100) DEFAULT NULL,
  `Symptom` varchar(100) DEFAULT NULL,
  `PH_AO_Vlefaro` varchar(45) DEFAULT NULL,
  `PH_AO_Keratoeidis` varchar(20) DEFAULT NULL COMMENT 'IMAGE PATH',
  `PH_AO_Epipefikotas` varchar(45) DEFAULT NULL,
  `PH_AO_PTHGwnia` varchar(15) DEFAULT NULL,
  `PH_AO_Fakos` varchar(45) DEFAULT NULL,
  `PH_AO_Irida` varchar(45) DEFAULT NULL,
  `PH_DO_Vlefaro` varchar(45) DEFAULT NULL,
  `PH_DO_Keratoeidis` varchar(20) DEFAULT NULL COMMENT 'IMAGE PATH',
  `PH_DO_Epipefikotas` varchar(45) DEFAULT NULL,
  `PH_DO_PTHGwnia` varchar(15) DEFAULT NULL,
  `PH_DO_Fakos` varchar(45) DEFAULT NULL,
  `PH_DO_Irida` varchar(45) DEFAULT NULL,
  `OH_AO_OxraKilida` varchar(45) DEFAULT NULL,
  `OH_AO_Aggeia` varchar(45) DEFAULT NULL,
  `OH_AO_OptikoNCD` varchar(20) DEFAULT NULL COMMENT 'IMAGE PATH',
  `OH_AO_Yaloeides` varchar(45) DEFAULT NULL,
  `OH_AO_Perifereia` varchar(45) DEFAULT NULL,
  `OH_DO_OxraKilida` varchar(45) DEFAULT NULL,
  `OH_DO_Aggeia` varchar(45) DEFAULT NULL,
  `OH_DO_OptikoNCD` varchar(20) DEFAULT NULL COMMENT 'IMAGE PATH',
  `OH_DO_Yaloeides` varchar(45) DEFAULT NULL,
  `OH_DO_Perifereia` varchar(45) DEFAULT NULL,
  `EOP_AO` tinyint(3) unsigned DEFAULT NULL COMMENT 'ενδοφθάλμια πίεση \nτιμές 0
έως 80',
  `EOP_Method_AO` varchar(45) DEFAULT NULL COMMENT 'τιμές :\nGOLDMAN\nAIR
TONOMETRY\n ΔΑΚΤΥΛΙΚΑ',
  `Paximetria_AO` smallint(6) DEFAULT NULL COMMENT 'τιμές:\n350-700',
  `therapy_AO` varchar(100) DEFAULT NULL,
  `EOP_Time_AO` time DEFAULT NULL,
  `EOP_DO` tinyint(3) unsigned DEFAULT NULL COMMENT 'ενδοφθάλμια πίεση \nτιμές 0
έως 80',
  `EOP_Method_DO` varchar(45) DEFAULT NULL COMMENT 'τιμές :\nGOLDMAN\nAIR
TONOMETRY\n ΔΑΚΤΥΛΙΚΑ',
  `Paximetria_DO` smallint(6) DEFAULT NULL COMMENT 'τιμές:\n350-700',
  `therapy_DO` varchar(100) DEFAULT NULL,
  `EOP_Time_DO` time DEFAULT NULL,
  `Colors_AO` varchar(45) DEFAULT NULL COMMENT 'τιμές:\nΚΦ, ελαττωμένη κλπ',
  `Pullary_Reactions_AO` varchar(45) DEFAULT NULL COMMENT 'τιμές:\nΆμμεσο, Έμμεσο
κλπ',
  `RAPD_AO` varchar(1) DEFAULT NULL COMMENT 'τιμές:\n+\n-',
  `Colors_DO` varchar(45) DEFAULT NULL COMMENT 'τιμές:\nΚΦ, ελαττωμένη κλπ',
```

```
  `Pullary_Reactions_DO` varchar(45) DEFAULT NULL COMMENT 'τιμές:\nΆμεσο, Έμμεσο κλπ',
  `RAPD_DO` varchar(1) DEFAULT NULL COMMENT 'τιμές:\n+\n-',
  `Schimmer_Test_mm` tinyint(3) unsigned DEFAULT NULL COMMENT 'τιμές: \n0-15',
  `Schimmer_Test_Mins` tinyint(3) unsigned DEFAULT NULL COMMENT 'τιμές: \n1-15',
  `TBUT` tinyint(3) unsigned DEFAULT NULL COMMENT 'τιμές: \n1-20',
  `Concurrency` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`AMKA`,`Examination_Date`),
  KEY `FKCvalue1_idx` (`PH_AO_PTHGwnia`),
  KEY `FKCvaluepthgwnia2_idx` (`PH_DO_PTHGwnia`),
  KEY `FKC2_idx` (`doctor`),
  CONSTRAINT `FKC1` FOREIGN KEY (`AMKA`) REFERENCES `patient` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FKC2` FOREIGN KEY (`doctor`) REFERENCES `doctor` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FKCvaluepthgwnia1` FOREIGN KEY (`PH_AO_PTHGwnia`) REFERENCES `z_tb_values_pthgwnia` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `FKCvaluepthgwnia2` FOREIGN KEY (`PH_DO_PTHGwnia`) REFERENCES `z_tb_values_pthgwnia` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “diagnosis”

```
DROP TABLE IF EXISTS `diagnosis`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `diagnosis` (
  `AMKA` varchar(10) NOT NULL,
  `Diagnosis_Date` date NOT NULL,
  `ICD10_Code` varchar(5) NOT NULL,
  `Paratiriseis` varchar(200) DEFAULT NULL,
  `Doctor` varchar(10) NOT NULL,
  `Concurrency` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`AMKA`,`Diagnosis_Date`,`ICD10_Code`),
  KEY `FCD2_idx` (`ICD10_Code`),
  KEY `FCD3_idx` (`Doctor`),
  CONSTRAINT `FCD1` FOREIGN KEY (`AMKA`) REFERENCES `patient` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FCD2` FOREIGN KEY (`ICD10_Code`) REFERENCES `tb_data_icd` (`col7`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `FCD3` FOREIGN KEY (`Doctor`) REFERENCES `doctor` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “diathlastikos_elegxos”

```
DROP TABLE IF EXISTS `diathlastikos_elegxos`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `diathlastikos_elegxos` (
  `AMKA` varchar(10) NOT NULL,
  `Examination_Date` date NOT NULL,
  `doctor` varchar(10) NOT NULL,
  `Examination_Place` varchar(45) DEFAULT NULL,
  `Pd_Far` decimal(3,2) DEFAULT NULL COMMENT 'απόσταση ανάμεσα στα δύο κέντρα για μακριά',
  `Pd_Near` decimal(3,2) DEFAULT NULL COMMENT 'απόσταση ανάμεσα στα δύο κέντρα για κοντά',
  `OD_SC` tinyint(3) unsigned DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nΟπτική Οξύτητα χωρίς διόρθωση πχ 10/10',
  `OD_CC` tinyint(3) unsigned DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nΟπτική Οξύτητα με διόρθωση',
  `OD_BC` tinyint(3) unsigned DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nΟπτική Οξύτητα best corrected',
  `OD_pH` tinyint(3) unsigned DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nΟπτική Οξύτητα με pinhole',
  `OD_NEWM_Sph` decimal(4,2) DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nσφάιρωμα υπερμετροπία/μειωπία',
  `OD_NEWM_Cyl` decimal(4,2) DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nκύλινδρος αστιγματισμός ',
  `OD_NEWM_Axis` tinyint(3) unsigned DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nάξονας αστιγματισμού',
  `OS_SC` tinyint(3) unsigned DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nΟπτική Οξύτητα χωρίς διόρθωση πχ 10/10',
  `OS_CC` tinyint(3) unsigned DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nΟπτική Οξύτητα με διόρθωση',
  `OS_BC` tinyint(3) unsigned DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nΟπτική Οξύτητα best corrected',
  `OS_pH` tinyint(3) unsigned DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nΟπτική Οξύτητα με pinhole',
  `OS_NEWM_Sph` decimal(4,2) DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nσφάιρωμα υπερμετροπία/μειωπία',
  `OS_NEWM_Cyl` decimal(4,2) DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nκύλινδρος αστιγματισμός ',
  `OS_NEWM_Axis` tinyint(3) unsigned DEFAULT NULL COMMENT 'Μακρινή Διόρθωση\n\nάξονας αστιγματισμού',
  `OD_NEW_Jaeger` tinyint(3) unsigned DEFAULT NULL COMMENT 'Κοντινά Γυαλιά',
  `OD_NEW_Add` decimal(3,2) DEFAULT NULL COMMENT 'Κοντινά Γυαλιά\n  \nπόσο προσθέτεις για το κοντινό γυαλί',
  `OD_NEW_Sph` decimal(4,2) DEFAULT NULL COMMENT 'Κοντινά Γυαλιά\n  \nσφάιρωμα υπερμετροπία/μειωπία',
  `OD_NEW_Cyl` decimal(4,2) DEFAULT NULL COMMENT 'Κοντινά Γυαλιά\n \nκύλινδρος αστιγματισμός ',
```

```
  `OD_NEW_Axis` tinyint(3) unsigned DEFAULT NULL COMMENT 'Κοντινά Γυαλιά\n \nάξονας αστιγματισμού',
  `OS_NEW_Jaeger` tinyint(3) unsigned DEFAULT NULL COMMENT 'Κοντινά Γυαλιά',
  `OS_NEW_Add` decimal(3,2) DEFAULT NULL COMMENT 'Κοντινά Γυαλιά\n  \nπόσο προσθέτεις για το κοντινό γυαλί',
  `OS_NEW_Sph` decimal(4,2) DEFAULT NULL COMMENT 'Κοντινά Γυαλιά\n  \nσφάιρωμα υπερμετροπία/μειωπία',
  `OS_NEW_Cyl` decimal(4,2) DEFAULT NULL COMMENT 'Κοντινά Γυαλιά\n \nκύλινδρος αστιγματισμός ',
  `OS_NEW_Axis` tinyint(3) unsigned DEFAULT NULL COMMENT 'Κοντινά Γυαλιά\n \nάξονας αστιγματισμού',
  `OD_K1_Mm` decimal(4,2) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο ',
  `OD_K1_D` decimal(3,1) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\ndieopter',
  `OD_K1_Degrees` tinyint(3) unsigned DEFAULT NULL COMMENT 'Ρεφροκτόμετρο ',
  `OD_K2_Mm` decimal(4,2) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο ',
  `OD_K2_D` decimal(3,1) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\ndieopter',
  `OD_K2_Degrees` tinyint(3) unsigned DEFAULT NULL COMMENT 'Ρεφροκτόμετρο ',
  `OD_Kavg` decimal(3,1) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο ',
  `OS_K1_Mm` decimal(4,2) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο ',
  `OS_K1_D` decimal(3,1) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\ndieopter',
  `OS_K1_Degrees` tinyint(3) unsigned DEFAULT NULL COMMENT 'Ρεφροκτόμετρο ',
  `OS_K2_Mm` decimal(4,2) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο ',
  `OS_K2_D` decimal(3,1) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\ndieopter',
  `OS_K2_Degrees` tinyint(3) unsigned DEFAULT NULL COMMENT 'Ρεφροκτόμετρο ',
  `OS_Kavg` decimal(3,1) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο ',
  `OD_Cyclo` bit(1) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο ',
  `OD_Cyclo_Sph` decimal(4,2) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\nσφάιρωμα υπερμετροπία/μειωπία',
  `OD_Cyclo_Cyl` decimal(4,2) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\n κύλινδρος αστιγματισμός ',
  `OD_Cyclo_Axis` tinyint(3) unsigned DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\nάξονας αστιγματισμού',
  `OD_NotCyclo_Sph` decimal(4,2) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\nσφάιρωμα υπερμετροπία/μειωπία',
  `OD_NotCyclo_Cyl` decimal(4,2) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\nκύλινδρος αστιγματισμός ',
  `OD_NotCyclo_Axis` tinyint(3) unsigned DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\nάξονας αστιγματισμού',
  `OS_Cyclo` bit(1) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο ',
  `OS_Cyclo_Sph` decimal(4,2) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\nσφάιρωμα υπερμετροπία/μειωπία',
  `OS_Cyclo_Cyl` decimal(4,2) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\n κύλινδρος αστιγματισμός ',
  `OS_Cyclo_Axis` tinyint(3) unsigned DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\nάξονας αστιγματισμού',
  `OS_NotCyclo_Sph` decimal(4,2) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\nσφάιρωμα υπερμετροπία/μειωπία',
```

```
  `OS_NotCyclo_Cyl` decimal(4,2) DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\nκύλινδρος αστιγματισμός ',
  `OS_NotCyclo_Axis` tinyint(3) unsigned DEFAULT NULL COMMENT 'Ρεφροκτόμετρο \n\nάξονας αστιγματισμού',
  `OD_Info` mediumtext,
  `OS_Info` mediumtext,
  `OD_Cover_Test` varchar(45) DEFAULT NULL COMMENT 'Ορθοπτικός Έλεγχος',
  `OS_Cover_Test` varchar(45) DEFAULT NULL COMMENT 'Ορθοπτικός Έλεγχος',
  `OD_Alternate_Cover_Test` varchar(45) DEFAULT NULL COMMENT 'Ορθοπτικός Έλεγχος',
  `OS_Alternate_Cover_Test` varchar(45) DEFAULT NULL COMMENT 'Ορθοπτικός Έλεγχος',
  `Concurrency` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`AMKA`,`Examination_Date`),
  KEY `fkvalidvalue3_idx` (`OD_NEWM_Sph`),
  KEY `fkvalidvalue4_idx` (`OD_NEWM_Cyl`),
  KEY `fkvalidvalue7_idx` (`OS_NEWM_Sph`),
  KEY `fkvalidvalue8_idx` (`OS_NEWM_Cyl`),
  KEY `fkvalidvalue10_idx` (`OD_NEW_Sph`),
  KEY `fkvalidvalue12_idx` (`OD_NEW_Cyl`),
  KEY `fkvalidvalue14_idx` (`OS_NEW_Sph`),
  KEY `fkvalidvalue16_idx` (`OS_NEW_Cyl`),
  KEY `fkvalidvalue17_idx` (`OD_Cyclo_Sph`),
  KEY `fkvalidvalue18_idx` (`OD_Cyclo_Cyl`),
  KEY `fkvalidvalue19_idx` (`OD_NotCyclo_Sph`),
  KEY `fkvalidvalue20_idx` (`OD_NotCyclo_Cyl`),
  KEY `fkvalidvalue21_idx` (`OS_Cyclo_Sph`),
  KEY `fkvalidvalue22_idx` (`OS_Cyclo_Cyl`),
  KEY `fkvalidvalue23_idx` (`OS_NotCyclo_Sph`),
  KEY `fkvalidvalue24_idx` (`OS_NotCyclo_Cyl`),
  KEY `fkd2_idx` (`doctor`),
  KEY `fkadd1_idx1` (`OD_NEW_Add`),
  KEY `fkadd4_idx` (`OS_NEW_Add`),
  CONSTRAINT `fkadd2` FOREIGN KEY (`OD_NEW_Add`) REFERENCES `z_tb_values_add` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `fkadd4` FOREIGN KEY (`OS_NEW_Add`) REFERENCES `z_tb_values_add` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `fkd1` FOREIGN KEY (`AMKA`) REFERENCES `patient` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fkd2` FOREIGN KEY (`doctor`) REFERENCES `doctor` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fkvalidvalue10` FOREIGN KEY (`OD_NEW_Sph`) REFERENCES `z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `fkvalidvalue12` FOREIGN KEY (`OD_NEW_Cyl`) REFERENCES `z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT `fkvalidvalue14` FOREIGN KEY (`OS_NEW_Sph`) REFERENCES `z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
```

```
  CONSTRAINT    `fkvalidvalue16`    FOREIGN    KEY    (`OS_NEW_Cyl`)    REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT    `fkvalidvalue17`    FOREIGN    KEY    (`OD_Cyclo_Sph`)    REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT    `fkvalidvalue18`    FOREIGN    KEY    (`OD_Cyclo_Cyl`)    REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT   `fkvalidvalue19`   FOREIGN   KEY   (`OD_NotCyclo_Sph`)   REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT   `fkvalidvalue20`   FOREIGN   KEY   (`OD_NotCyclo_Cyl`)   REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT    `fkvalidvalue21`    FOREIGN    KEY    (`OS_Cyclo_Sph`)    REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT    `fkvalidvalue22`    FOREIGN    KEY    (`OS_Cyclo_Cyl`)    REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT   `fkvalidvalue23`   FOREIGN   KEY   (`OS_NotCyclo_Sph`)   REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT   `fkvalidvalue24`   FOREIGN   KEY   (`OS_NotCyclo_Cyl`)   REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT    `fkvalidvalue3`    FOREIGN    KEY    (`OD_NEWM_Sph`)    REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT    `fkvalidvalue4`    FOREIGN    KEY    (`OD_NEWM_Cyl`)    REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT    `fkvalidvalue7`    FOREIGN    KEY    (`OS_NEWM_Sph`)    REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION,
  CONSTRAINT    `fkvalidvalue8`    FOREIGN    KEY    (`OS_NEWM_Cyl`)    REFERENCES
`z_tb_values_sph_cyl` (`Value`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “doctor”

```
DROP TABLE IF EXISTS `doctor`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `doctor` (
  `AMKA` varchar(10) NOT NULL,
  `Active` bit(1) NOT NULL DEFAULT b'1' COMMENT 'b''0'' : INACTIVE\nb''1'' :
Active',
  PRIMARY KEY (`AMKA`),
  CONSTRAINT `fkdoc1` FOREIGN KEY (`AMKA`) REFERENCES `personel` (`AMKA`) ON DELETE
CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=greek COMMENT='Χαρακτηριστικά Ιατρών';
```

ΠΙΝΑΚΑΣ “eswteriki_allilografia”

```
DROP TABLE IF EXISTS `eswteriki_allilografia`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `eswteriki_allilografia` (
  `Sender` varchar(10) NOT NULL,
```

```
  `Receiver` varchar(10) NOT NULL,
  `Message` mediumtext,
  `Date_Sent` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `Thema` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`Sender`,`Date_Sent`,`Receiver`),
  KEY `FKAL2_idx` (`Receiver`),
  CONSTRAINT `FKAL2` FOREIGN KEY (`Receiver`) REFERENCES `personel` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FKALL1` FOREIGN KEY (`Sender`) REFERENCES `personel` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “p_allergies”

```
DROP TABLE IF EXISTS `p_allergies`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `p_allergies` (
  `AMKA` varchar(10) NOT NULL,
  `Allergy` varchar(60) NOT NULL,
  PRIMARY KEY (`AMKA`,`Allergy`),
  CONSTRAINT `fkal1` FOREIGN KEY (`AMKA`) REFERENCES `patient` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=greek COMMENT='Table concerning Allergies';
```

ΠΙΝΑΚΑΣ “p_disease”

```
DROP TABLE IF EXISTS `p_disease`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `p_disease` (
  `AMKA` varchar(10) NOT NULL,
  `disease` varchar(60) NOT NULL,
  PRIMARY KEY (`disease`,`AMKA`),
  KEY `fkds1_idx` (`AMKA`),
  CONSTRAINT `fkds1` FOREIGN KEY (`AMKA`) REFERENCES `patient` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “p_medication”

```
DROP TABLE IF EXISTS `p_medication`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `p_medication` (
  `AMKA` varchar(10) NOT NULL,
  `medication` varchar(60) NOT NULL,
```

```
  PRIMARY KEY (`AMKA`,`medication`),
  CONSTRAINT `fkmd1` FOREIGN KEY (`AMKA`) REFERENCES `patient` (`AMKA`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “patient”

```
DROP TABLE IF EXISTS `patient`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `patient` (
  `AMKA` varchar(10) NOT NULL,
  `AMA` varchar(10) DEFAULT NULL COMMENT 'Arithmos Tautotitas',
  `Name` varchar(25) DEFAULT NULL,
  `Surname` varchar(25) DEFAULT NULL,
  `Gender` varchar(1) DEFAULT NULL,
  `Occupation` varchar(45) DEFAULT NULL,
  `Date_Birth` date DEFAULT NULL,
  `Father_Name` varchar(25) DEFAULT NULL,
  `Mother_Name` varchar(25) DEFAULT NULL,
  `Nationality` varchar(25) DEFAULT NULL,
  `Address` varchar(50) DEFAULT NULL,
  `Address_Location` varchar(50) DEFAULT NULL,
  `TK` varchar(5) DEFAULT NULL,
  `Nomos` varchar(50) DEFAULT NULL,
  `Country` varchar(50) DEFAULT NULL,
  `Tameio` varchar(25) DEFAULT NULL,
  `Private_Insurance` bit(1) DEFAULT NULL,
  `Date_Patient_Created` date DEFAULT NULL,
  `Payment_Way` varchar(25) DEFAULT NULL,
  `Email` varchar(50) NOT NULL,
  `Email_Accept` bit(1) DEFAULT b'1' COMMENT '0:Accept\n1:Reject',
  `SMS_Accept` bit(1) DEFAULT b'0' COMMENT '0:Accept\n1:Reject',
  `Mail_Accept` bit(1) DEFAULT b'0' COMMENT '0:Accept\n1:Reject',
  `Account_Type` bit(1) DEFAULT b'0' COMMENT '0:regular member\n2:golden member',
  `Photo` varchar(20) DEFAULT NULL COMMENT 'IMAGE PATH',
  `Cellphone` varchar(10) NOT NULL,
  `Phone_Home` varchar(10) DEFAULT NULL,
  `username` varchar(20) DEFAULT NULL,
  `password` varchar(20) DEFAULT NULL,
  `Concurrency` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`AMKA`),
  UNIQUE KEY `Email_UNIQUE` (`Email`),
  UNIQUE KEY `Cellphone_UNIQUE` (`Cellphone`),
```

```
  UNIQUE KEY `AMA_UNIQUE` (`AMA`),
  UNIQUE KEY `username_UNIQUE` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=greek COMMENT='Table concerning patient main info';
```

ΠΙΝΑΚΑΣ “personel”

```
DROP TABLE IF EXISTS `personel`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `personel` (
  `AMKA` varchar(10) DEFAULT NULL,
  `AMA` varchar(10) DEFAULT NULL COMMENT 'Arithmos Tautotitas',
  `Name` varchar(25) DEFAULT NULL,
  `Surname` varchar(25) DEFAULT NULL,
  `Gender` varchar(1) DEFAULT NULL,
  `Date_Birth` date DEFAULT NULL,
  `Father_Name` varchar(25) DEFAULT NULL,
  `Mother_Name` varchar(25) DEFAULT NULL,
  `Nationality` varchar(25) DEFAULT NULL,
  `Address` varchar(50) DEFAULT NULL,
  `Address_Location` varchar(50) DEFAULT NULL,
  `TK` varchar(5) DEFAULT NULL,
  `Nomos` varchar(50) DEFAULT NULL,
  `Country` varchar(50) DEFAULT NULL,
  `Tameio` varchar(25) DEFAULT NULL,
  `Date_Personel_Created` date DEFAULT NULL,
  `Payment_Way` varchar(25) DEFAULT NULL,
  `Email` varchar(50) NOT NULL,
  `Email_Accept` bit(1) DEFAULT b'1' COMMENT '0:Accept\n1:Reject',
  `SMS_Accept` bit(1) DEFAULT b'0' COMMENT '0:Accept\n1:Reject',
  `Mail_Accept` bit(1) DEFAULT b'0' COMMENT '0:Accept\n1:Reject',
  `Photo` varchar(20) DEFAULT NULL COMMENT 'IMAGE PATH',
  `Cellphone` varchar(10) NOT NULL,
  `Phone_Home` varchar(10) DEFAULT NULL,
  `username` varchar(20) NOT NULL,
  `password` varchar(20) NOT NULL,
  `Concurrency` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`username`),
  UNIQUE KEY `Email_UNIQUE` (`Email`),
  UNIQUE KEY `AMKA` (`AMKA`),
  UNIQUE KEY `AMA_UNIQUE` (`AMA`)
) ENGINE=InnoDB DEFAULT CHARSET=greek COMMENT='Γενικά χαρακτηριστικά προσωπικού
ιατρίου/κλινικής';
```

ΠΙΝΑΚΑΣ “skeyasma”

```
DROP TABLE IF EXISTS `skeyasma`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `skeyasma` (
  `Medicine_Code` tinyint(4) NOT NULL AUTO_INCREMENT,
  `Name` varchar(80) DEFAULT NULL,
  `Type` varchar(45) DEFAULT NULL,
  `Manufacturer` varchar(45) DEFAULT NULL,
  `Description` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`Medicine_Code`)
) ENGINE=InnoDB AUTO_INCREMENT=25 DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “tb_data_icd”

```
DROP TABLE IF EXISTS `tb_data_icd`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `tb_data_icd` (
  `col1` int(1) DEFAULT NULL,
  `col2` varchar(1) CHARACTER SET utf8 DEFAULT NULL,
  `col3` varchar(1) CHARACTER SET utf8 DEFAULT NULL,
  `col4` int(2) DEFAULT NULL,
  `col5` varchar(3) CHARACTER SET utf8 DEFAULT NULL,
  `col6` varchar(6) CHARACTER SET utf8 DEFAULT NULL,
  `col7` varchar(5) NOT NULL,
  `col8` varchar(4) CHARACTER SET utf8 DEFAULT NULL,
  `col9` varchar(185) CHARACTER SET utf8 DEFAULT NULL,
  `col10` varchar(5) CHARACTER SET utf8 DEFAULT NULL,
  `col11` varchar(5) CHARACTER SET utf8 DEFAULT NULL,
  `col12` varchar(5) CHARACTER SET utf8 DEFAULT NULL,
  `col13` varchar(5) CHARACTER SET utf8 DEFAULT NULL,
  `col14` varchar(5) CHARACTER SET utf8 DEFAULT NULL,
  `col15` bit(1) DEFAULT NULL,
  PRIMARY KEY (`col7`)
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “therapia”

```
DROP TABLE IF EXISTS `therapia`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `therapia` (
  `AMKA` varchar(10) NOT NULL,
  `Diagnosis_Date` date NOT NULL,
  `ICD10_Code` varchar(5) NOT NULL,
```

```
  `Medicine_Code` tinyint(4) NOT NULL,
  `thema` varchar(45) NOT NULL,
  `Therapy_Duration` varchar(45) DEFAULT NULL,
  `Date_Entry` date DEFAULT NULL,
  `Ofthalmos` varchar(20) NOT NULL,
  `Posotita_Farmakou` tinyint(4) DEFAULT NULL,
  `Concurrency` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (`Medicine_Code`,`AMKA`,`Diagnosis_Date`,`ICD10_Code`),
  KEY `fkt1_idx` (`AMKA`,`Diagnosis_Date`,`ICD10_Code`),
  KEY `fkt3_idx` (`thema`),
  CONSTRAINT `fkt2` FOREIGN KEY (`Medicine_Code`) REFERENCES `skeyasma`
(`Medicine_Code`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fktt1` FOREIGN KEY (`AMKA`, `Diagnosis_Date`, `ICD10_Code`)
REFERENCES `diagnosis` (`AMKA`, `Diagnosis_Date`, `ICD10_Code`) ON DELETE CASCADE
ON UPDATE CASCADE,
  CONSTRAINT `fktt3` FOREIGN KEY (`thema`) REFERENCES `z_thema_therapeias`
(`thema_Name`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “z_appointment_type”

```
DROP TABLE IF EXISTS `z_appointment_type`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `z_appointment_type` (
  `Appointment_Name` varchar(45) NOT NULL,
  `Appointment_Duration` varchar(45) DEFAULT NULL,
  `Appointment_Color` varchar(7) DEFAULT NULL,
  PRIMARY KEY (`Appointment_Name`)
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “z_tb_values_add”

```
DROP TABLE IF EXISTS `z_tb_values_add`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `z_tb_values_add` (
  `Value` decimal(3,2) NOT NULL,
  PRIMARY KEY (`Value`)
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “z_tb_values_pthgwnia”

```
DROP TABLE IF EXISTS `z_tb_values_pthgwnia`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `z_tb_values_pthgwnia` (
```

```
  `Value` varchar(15) NOT NULL,
  PRIMARY KEY (`Value`)
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “z_tb_values_Sph_cyl”

```
DROP TABLE IF EXISTS `z_tb_values_sph_cyl`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `z_tb_values_sph_cyl` (
  `Value` decimal(4,2) NOT NULL,
  PRIMARY KEY (`Value`)
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

ΠΙΝΑΚΑΣ “z_thema_therapeias”

```
DROP TABLE IF EXISTS `z_thema_therapeias`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `z_thema_therapeias` (
  `thema_Name` varchar(45) NOT NULL,
  PRIMARY KEY (`thema_Name`)
) ENGINE=InnoDB DEFAULT CHARSET=greek;
```

9 Παράρτημα Β

Παρακάτω παρατίθενται οι εντολές SQL για την δημιουργία των πινάκων της ΒΔ «eopyy_db»:

ΠΙΝΑΚΑΣ “patient”

```
DROP TABLE IF EXISTS `patient`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `patient` (
  `AMKA` varchar(10) NOT NULL,
  `AMA` varchar(10) DEFAULT NULL COMMENT 'Arithmos Tautotitas',
  `Name` varchar(25) DEFAULT NULL,
  `Surname` varchar(25) DEFAULT NULL,
  `Gender` varchar(1) DEFAULT NULL,
  `Occupation` varchar(45) DEFAULT NULL,
  `Date_Birth` date DEFAULT NULL,
  `Father_Name` varchar(25) DEFAULT NULL,
  `Mother_Name` varchar(25) DEFAULT NULL,
  `Nationality` varchar(25) DEFAULT NULL,
  `Address` varchar(50) DEFAULT NULL,
  `Address_Location` varchar(50) DEFAULT NULL,
```

```
  `TK` varchar(5) DEFAULT NULL,
  `Nomos` varchar(50) DEFAULT NULL,
  `Country` varchar(50) DEFAULT NULL,
  `Tameio` varchar(25) DEFAULT NULL,
  `Private_Insurance` bit(1) DEFAULT NULL,
  `Email` varchar(50) NOT NULL,
  `Cellphone` varchar(10) NOT NULL,
  `Phone_Home` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`AMKA`),
  UNIQUE KEY `Email_UNIQUE` (`Email`),
  UNIQUE KEY `Cellphone_UNIQUE` (`Cellphone`),
  UNIQUE KEY `AMA_UNIQUE` (`AMA`)
) ENGINE=InnoDB DEFAULT CHARSET=greek;
/*!40101 SET character_set_client = @saved_cs_client */;
```

10 Βιβλιογραφία

[1] Pro WPF 4.5 in CSharp 4th edition by Apress

[2] The C# Player's Guide 2nd edition by RB Whitaker

[3] Pro WCF 4.5 2nd edition by Apress

[4] MySQL Cookbook Solutions for Database Developers and Administrators by Paul DuBois

[5] http://www.codeproject.com/Articles/140611/WPF-Tutorial-Beginning

[6] http://www.wpf-tutorial.com/data-binding/using-the-datacontext/

[7] https://msdn.microsoft.com/en-us/library/ms171923%28v=vs.110%29.aspx?f=255&MSPPError=-2147217396

[8] http://www.codeproject.com/Articles/30905/WPF-DataGrid-Practical-Examples#validation

[9] https://www.simple-talk.com/dotnet/.net-tools/entity-framework-performance-and-what-you-can-do-about-it/

[10] http://www.codeproject.com/Articles/531332/Implementing-a-Basic-Hello-World-WCF-Service-v4-5

[11] https://blogs.msdn.microsoft.com/jaimer/2009/01/20/styling-microsofts-wpf-datagrid/

[12] http://www.c-sharpcorner.com/UploadFile/nipuntomar/report-viewer-control-in-wpf/

[13] http://www.c-sharpcorner.com/UploadFile/mahesh/charting-in-wpf/

[14] http://www.codeproject.com/Tips/876349/WPF-Validation-using-INotifyDataErrorInfo

[15] http://www.codeproject.com/Articles/325911/A-Simple-Pattern-for-Creating-Re-useable-UserContr

[16] http://www.jonathanantoine.com/2011/09/18/wpf-4-5-asynchronous-data-validation/