Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Κατασκευή backend ιστοσελίδας διαχείρισης προσωπικού προφίλ υγείας με Python / Sqlite** **Construction of webpage backend for the management of personal health profile utilizing Python and Sqlite**
Ονοματεπώνυμο Φοιτητή	**ΔΟΥΛΓΕΡΙΔΗΣ ΠΑΝΑΓΙΩΤΗΣ**
Πατρώνυμο	**ΙΩΑΝΝΗΣ**
Αριθμός Μητρώου	**ΜΠΠΛ/ 20015**
Επιβλέπων	**ΕΥΘΥΜΙΟΣ ΑΛΕΠΗΣ, ΚΑΘΗΓΗΤΗΣ**

Ημερομηνία Παράδοσης **Ιούνιος 2024**

Τριμελής Εξεταστική Επιτροπή

Ευθύμιος Αλέπης	Μαρία Βίρβου	Κωνσταντίνος Πατσάκης
Καθηγητής	Καθηγήτρια	Αναπληρωτής Καθηγητής

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Περιεχόμενα

ΠΕΡΙΛΗΨΗ

Στην εργασία γίνεται προσομοίωση της τυποποιημένης διαδικασίας για δημιουργία νέων λύσεων software ειδικά σε μεγάλο εταιρικό επίπεδο. Ακολουθείται η μεθοδολογία καταγραφής των απαιτήσεων, επιλογή αρχιτεκτονικής, σχεδίασης της λύσης σε high level, σχεδίασης της λύσης σε low level, υλοποίησης και ελέγχων. Σκοπός ήταν η ανάπτυξη μια εφαρμογής η οποία θα δρα ως health assistant / planner και θα παρέχει λειτουργικότητες παρακολούθησης, προτάσεων και κοστολόγησης της διαχείρισης της υγείας ενός ανθρώπου.

Η λύση έχει σχεδιαστεί με μοντέρνες λειτουργικότητες αναφορά με το testing και deploy και η σχεδίαση της της δίνει μεγάλες δυνατότητες portability από περιβάλλον σε περιβάλλον και ειδικά για εφαρμογή σε cloud περιβάλλοντα. Έχουν χρησιμοποιηθεί τεχνολογίες containerization / OS virtualization για να προσομοιώσουνε τα απαιτούμενα περιβάλλοντα. Η εφαρμογή έχει

σχεδιαστεί να είναι Portable και ευκολά deployable και σε mobile περιβάλλοντα, κατά την ίδια λογική.

Για την υποστήριξη του πλήθους των λειτουργικοτήτων που περιέχονται βασιζόμαστε είτε σε ίδιες λύσεις είτε σε επίσημα Software-as-a-Service apis που μας παρέχουν τις αντίστοιχες λειτουργικότητες, όπως υπολογισμός ιατρικών μεγεθών, ειδοποιήσεις, ανάλυση κόστους, και προτάσεις για εκγύμναση και δίαιτα.

In the scope of this thesis, we have a simulation of the standardized procedure for the development of new software solutions in large-scale corporate environments. We follow the methodology of observing and recording business requirements, selecting the applicable software architecture, designing the solution in high level, designing the solution and each component in low level, implementing the solution as well as the testing suites. The purpose of this was the creation of an application that will act as a Health assistant / planner that will offer functionalities for the observation of a person's health, as well as suggestions and costing for the management of a person's health.

The software solution has been designed with modern functionalities in regards to testing and deployment, that give increase capabilities for portability between different environments and especially deployments in cloud environments. It has been implemented with containerization / OS virtualization technologies to simulate the necessary environments, the basis of the design is portability and easy deployment as a backend to both conventional as well as software environments.

For the support of the various functionalities we offer, extensive use of official Software-As-A-Service solutions are utilized to achieve base functionalities, like notifications, sms, email as well as functionalities pertaining to specialized health calculations and database data needed to provide the total of the functionality.

ΕΙΣΑΓΩΓΗ / ΠΕΡΙΓΡΑΦΗ ΠΡΟΒΛΗΜΑΤΟΣ

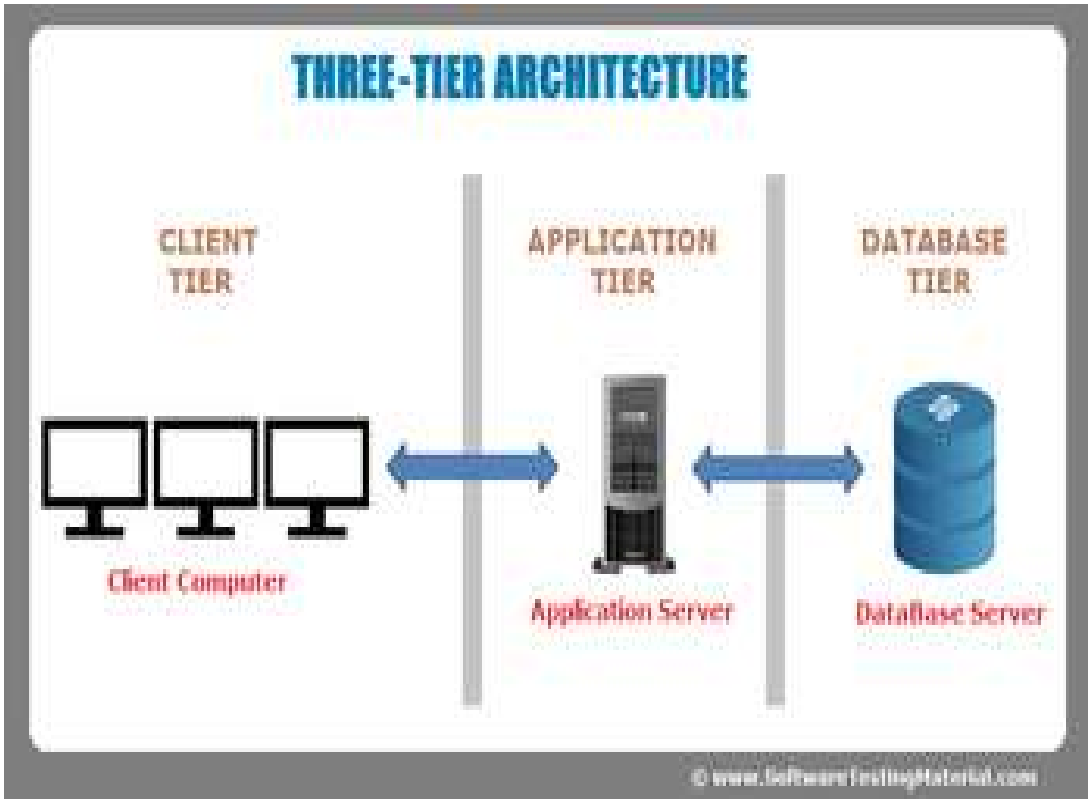
Αντικείμενο της παρούσας διπλωματικής είναι η πλήρης κάλυψης ενός εταιρικού πακέτου development delivery, από το στάδιο της απαίτησης, στο στάδιο της ανάλυσης, σχεδίασης, υλοποίησης και testing, και εν τέλει κάποιες προτάσεις για περαιτέρω βελτίωση. Με βάση αυτό έγινε η σύλληψη της αρχικής ιδέας και σχεδίαση της εφαρμογής, και επιλέχθηκε ως άξονας ο κλάδος της υγείας καθώς θεωρήσαμε ότι με τις νεες τεχνολογίας θα μπορέσουμε να αφομοιώσουμε αρκετές χρήσιμες λειτουργικότητες μέσα οι οποίες θα μπορέσουνε να βοηθήσουνε παθόντες να έχουν καλύτερη διαχείριση της υγείας τους, καθώς και το ιατρικό προσωπικό που τους παρακολουθεί να έχει μια καλύτερη και πιο σωστά καταγεγραμμένη ιστορικότητα όλου του ιατρικού ιστορικού του παθόντα.

Αρχικά θα έχουμε μια καταγραφή όλου του θεωρητικού υποβάθρου που σχετίζεται με την επιλογή μιας n-tier αρχιτεκτονικής, τα διάφορα implications καθώς και αντίστοιχα το θεωρητικό υπόβαθρο επιλογής σχεσιακής βάσης. Η εφαρμογή έχει στηθεί με γνώμονα το deployability και το extensibility χωρίς να επιφέρει μεγάλο developer κόστος όπως οι event-driven αρχιτεκτονικές.

Κατά το στάδιο της απαίτησης θα παραθέσουμε τμήμα ενός τυποποιημένου εγγράφου παράθεσης εμπορικών προδιαγραφών για την εφαρμογή μας, οι οποίες θα αφορούνε όλα τα layers της εφαρμογής από το back-end, έως και το front-end καθώς και όλες τις υποστηρικτικές υπηρεσίες που θα υποστηρίζουν το προϊόν μας. Ως τέτοιες σε πρώτη φάση αναγνωρίστηκαν ανάγκες για διεπαφη με Software-as-a-Service πλατφόρμες για αποστολή ειδοποιήσεων μέσω email/Sms, καθώς και integration με αντίστοιχες πλατφόρμες που προσφέρουν aggregated medical data processing. Οι πιο πολλές υπηρεσίες από αυτές είναι συνδρομητικές και αντίστοιχες λειτουργικότητες έχουν αγοραστεί για τις ανάγκες του demo.

Κατά το στάδιο της ανάλυσης θα παραθέσουμε όλη την ανάλυση της λειτουργίας της εφαρμογής, με παράθεση workflow charts και όλων των σχετικών complications που μπορεί να προκύψουν. Στο στάδιο αυτό παρατίθενται και τα αντίστοιχα sequence diagrams που δείχνουν την αλληλεπίδραση όλων των components της εφαρμογής. Θα αναλύσουμε ακόμα το γιατί επιλέξαμε την κάθε τεχνολογία, με γνώμονα ποιες παραμέτρους, και με τι αναμενόμενα οφέλη. Θα παρατεθούν τα βασικά client flows καθώς και τα αντίστοιχα διαχειριστικά flows που θα εξυπηρετήσουν την εφαρμογή.

Κατά το στάδιο της σχεδίασης θα δώσουμε έμφαση στην ευρύτερη αρχιτεκτονική της λύσης, καθώς και την αντίστοιχη αρχιτεκτονική του integration/deployment, με παράθεση αρχιτεκτονικών διαγραμμάτων και προδιαγραφών για όλες τις τεχνολογίες που χρησιμοποιήθηκαν, επεξηγήσεις και περαιτέρω πληροφορίες για να έχουμε μια ολοκληρωμένη αρχιτεκτονική εικόνα. Θα γίνει πλήρης ανάλυση του αρχιτεκτονικού μοντέλου καθώς και θα επεξηγηθεί η ευρύτερη επιλογή της αρχιτεκτονικής καθώς και ανάλυση των σχετικών αρχιτεκτονικών παραμέτρων σε σχέση με την δεδομένη αρχιτεκτονική. (ie scalability, elasticity etc)



Εικόνα 1: Simple version of n-tier architecture. The database, application and client tier are visible. Often there is an extra 4th tier that contains the business logic.

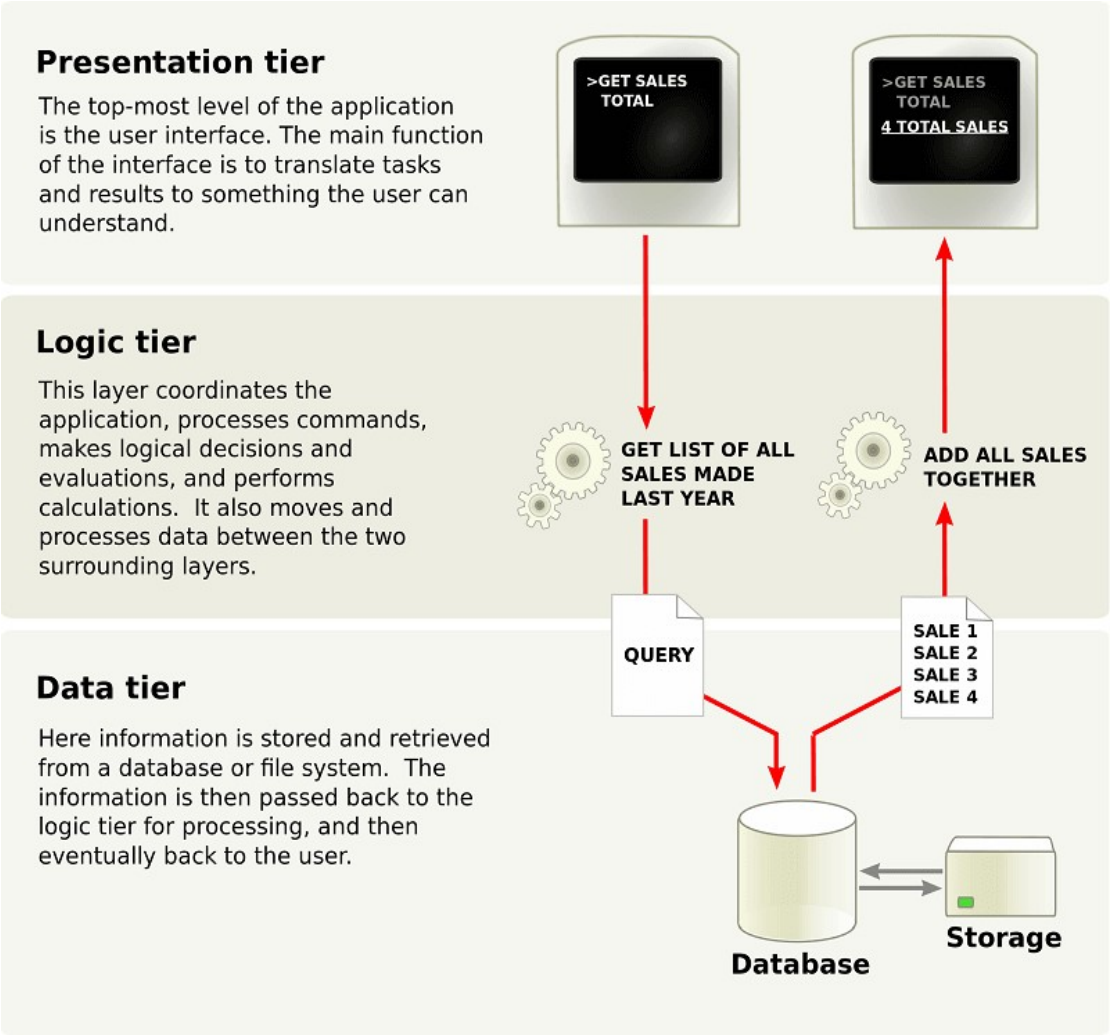
Κατά το στάδιο της υλοποίησης θα δώσουμε έμφαση στην υλοποίηση της λύσης και στην περαιτέρω επεξήγηση των διάφορων code stacks που χρησιμοποιήσανε καθώς και παραδοχών καθώς και περαιτέρω λειτουργιών που προστέθηκαν σε αυτή. Ακόμα θα γίνει μια αναφορά σε developer challenges, μηχανισμούς authentication/authorization, μηχανισμούς για Integration με SaaS πλατφόρμες για email / SMS και αλλά. Θα γίνει αναλυτική επεξήγηση του integration με όλα τα σχετικά συστήματα με παράθεση όλης της αντίστοιχης τεχνικής πληροφορίας.

Κατά το στάδιο του testing θα δοθεί έμφαση στα διαφορά σενάρια που χρησιμοποιήθηκαν για να τεστάρουνε την εφαρμογή καθώς και σε διαφορετικά είδη testing που χρησιμοποιήθηκαν (load testing / edge case testing / stress testing etc.), και την αυτοματοποίηση αυτών ώστε να μπορούν να αφομοιωθούν αντίστοιχα από το release pipeline.

Προσπάθεια μας είναι να δείξουμε τον πλήρη κύκλο development μιας μοντέρνας εφαρμογής με πλήρως αυτοματοποιημένο devops κομμάτι, integration με SaaS εφαρμογές, industry-standard auth και άλλες λειτουργικότητες με έναν τρόπο δομημένο που να μπορεί να αποτελέσει template for future development. Η εφαρμογή στήθηκε με βάση το web αλλά μπορεί εύκολα να μεταπτωσει και σε Mobile καθώς η αρχιτεκτονική που έχει επιλεχθεί (2-tier layered) έχει πλήρες separation of concerns για τα διάφορα layers (i.e. frontend/back-end).

Στις προτάσεις περαιτέρω βελτίωσης θα παραθέσουμε ιδέες για μελλοντικές επεκτάσεις και προσθήκες καθώς και προτάσεις για την υλοποίηση τους, σχετικά concerns και Implementation notes που θα χρειαστούν έτσι ώστε να γίνει scale up η εφαρμογή μας καλύτερα.

Η πλειοψηφία των διαγραμμάτων είναι από σχετικά site και δεν έχει copyrighted scope, όπου έγινε χρήση τέτοιου έγινε αντίστοιχη παράθεση στο site καθώς και στα σχετικά documents στην ενότητα «Βιβλιογραφία».



Εικόνα 2: Βασικό μοντέλο n-tier αρχιτεκτονικής.

Σημαντικό είναι να αναφέρουμε ότι η αρχική ιδέα προήλθε από το μάθημα του μεταπτυχιακού «Ιατρική πληροφορική» και ουσιαστικά είχαμε ένα porting της εφαρμογής από .net stack σε Hybrid python stack και επέκταση τους σε διάφορες λειτουργικότητες που η αρχική υλοποίηση δεν παρείχε. Τέτοιες λειτουργικότητες είναι:

1. Implementation/Integration with authentication/authorization services.
2. Implementation of a full schema on our relational database that conforms with industry standards.
3. Implementation of a 3-tier separated architecture, with front-end, logic-tier, and back-end components which are independently deployable and extensible.
4. Incorporation of devops tools and practices to simulate a real production pipeline with containerization as well as container automation that leads to scalable solutions.
5. Development of a fully functional front-end solution that adheres to modern front-end developer practices, using a modern stack.
6. Integration with various SaaS services for data processing and notification handling.

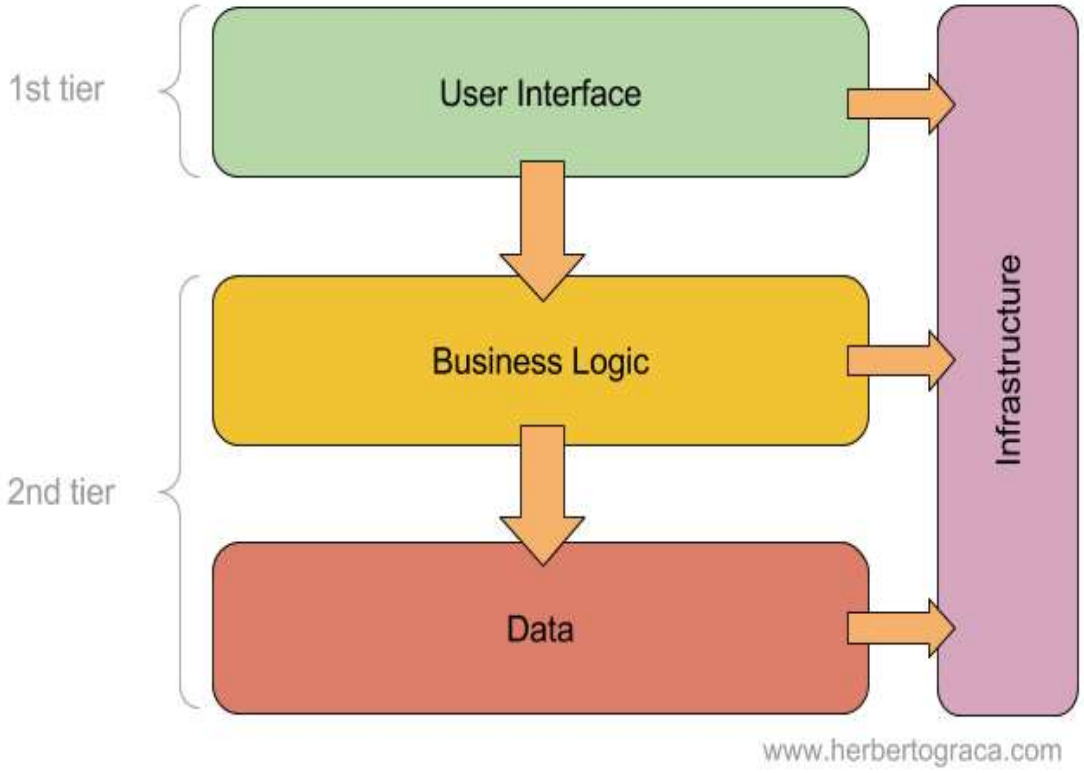
ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

Αρχιτεκτονικός Σχεδιασμός Λύσης

Μια από τις πιο διαδεδομένες αρχιτεκτονικές χτισίματος ενός web app είναι η αρχιτεκτονική ν-επιπέδων. Αυτού του τύπου η αρχιτεκτονική χρησιμοποιείται ευρέως από μεγάλο τμήμα του συνόλου των web εφαρμογών παγκοσμίως. Για εγκαταστάσεις μεγαλύτερων απαιτήσεων / κλίμακας τείνει να αντικατασταθεί από πιο Micro-oriented αρχιτεκτονικές, παρόλα αυτά παραμένει embedded ακόμα και σε αυτές ειδικά όταν αφορά εφαρμογές που αλληλοεπιδρούν με SaaS υπηρεσίες.

Λέγεται και multi-tier architecture διότι το λογισμικό κατασκευάζεται έτσι ώστε διαφορετικές λειτουργίες του να είναι υπό διαχείριση από διαφορετικές «ενότητες» η tiers της εφαρμογής. Ένας κλασσικός διαχωρισμός είναι:

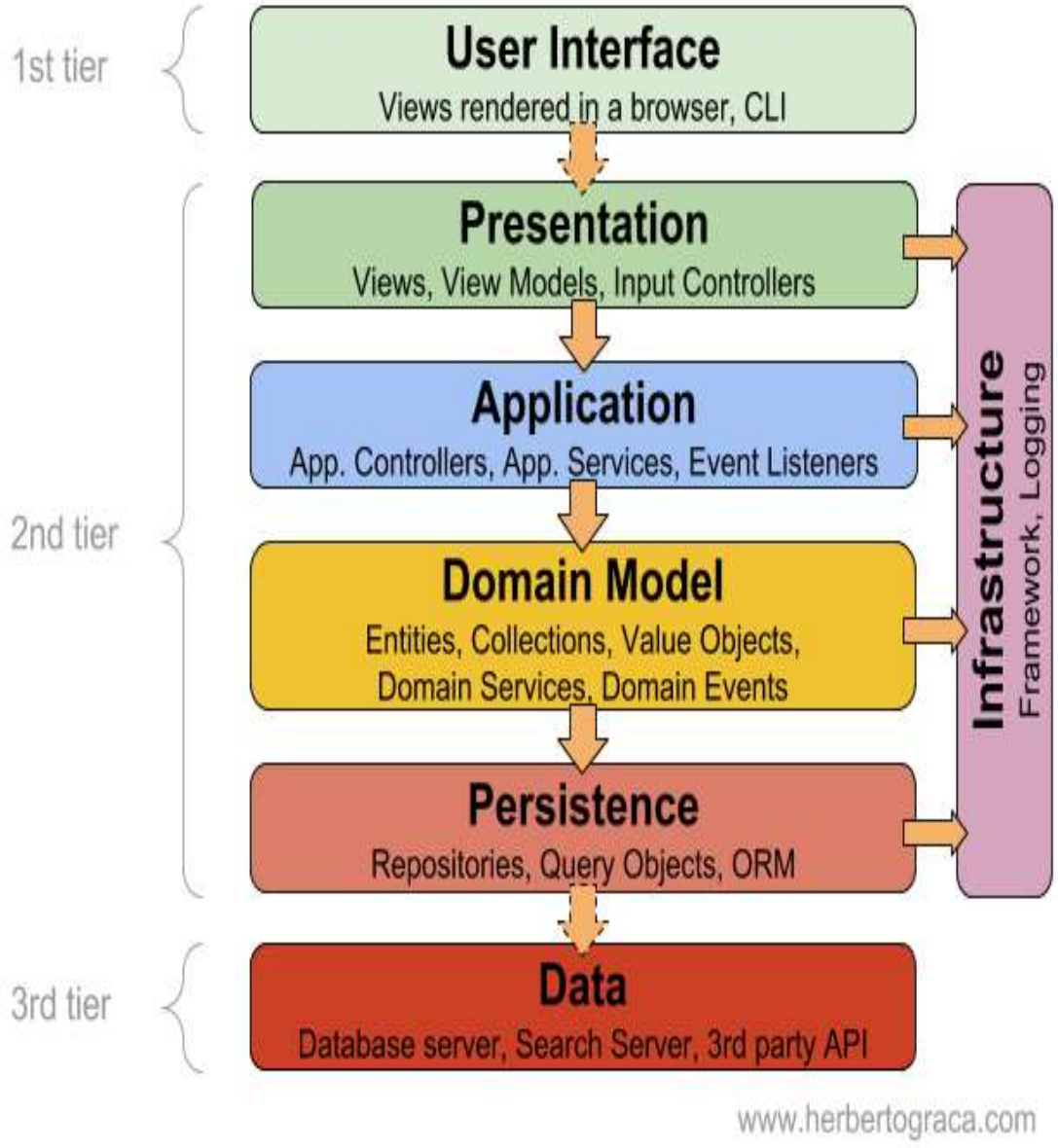
1. Presentation Layer
2. Data Management Layer
3. Presentation Layer



Εικόνα 3: Typical N-Tier 3 layer architecture

Αυτό σημαίνει ότι όλες αυτές οι διαφορετικές λειτουργικότητες είναι hosted σε διαφορετικές μηχανές / servers στην κλασσική περίπτωση, και στην δικιά μας περίπτωση σε containerized vm και εν συνέχεια σε azure Kubernetes pod. Λόγω της διάτμησης του όλου workload σε διαφορετικά tiers που εξυπηρετούνται από διαφορετικές μηχανές έχουμε προστασία ότι δεν γίνεται διαμοιρασμός στους πόρους που είναι αφιερωμένοι στο κάθε tier και έτσι πετυχαίνουμε να έχουμε υπηρεσία στο 100% των δυνατοτήτων του κάθε server. Επίσης αυτό μας βοηθάει αν έχουμε ξεχωριστή διαχείριση σε περίπτωση performance bottlenecks

(κυρίως παρατηρούνται σε βάσεις) και έχουμε την δυνατότητα να προβούμε σε δράσεις αύξησης του επιμέρους sizing των μηχανών για να εξυπηρετήσουμε τη ζήτηση.



Εικόνα 4: Typical N-tier architectory model featuring persistence and domain model

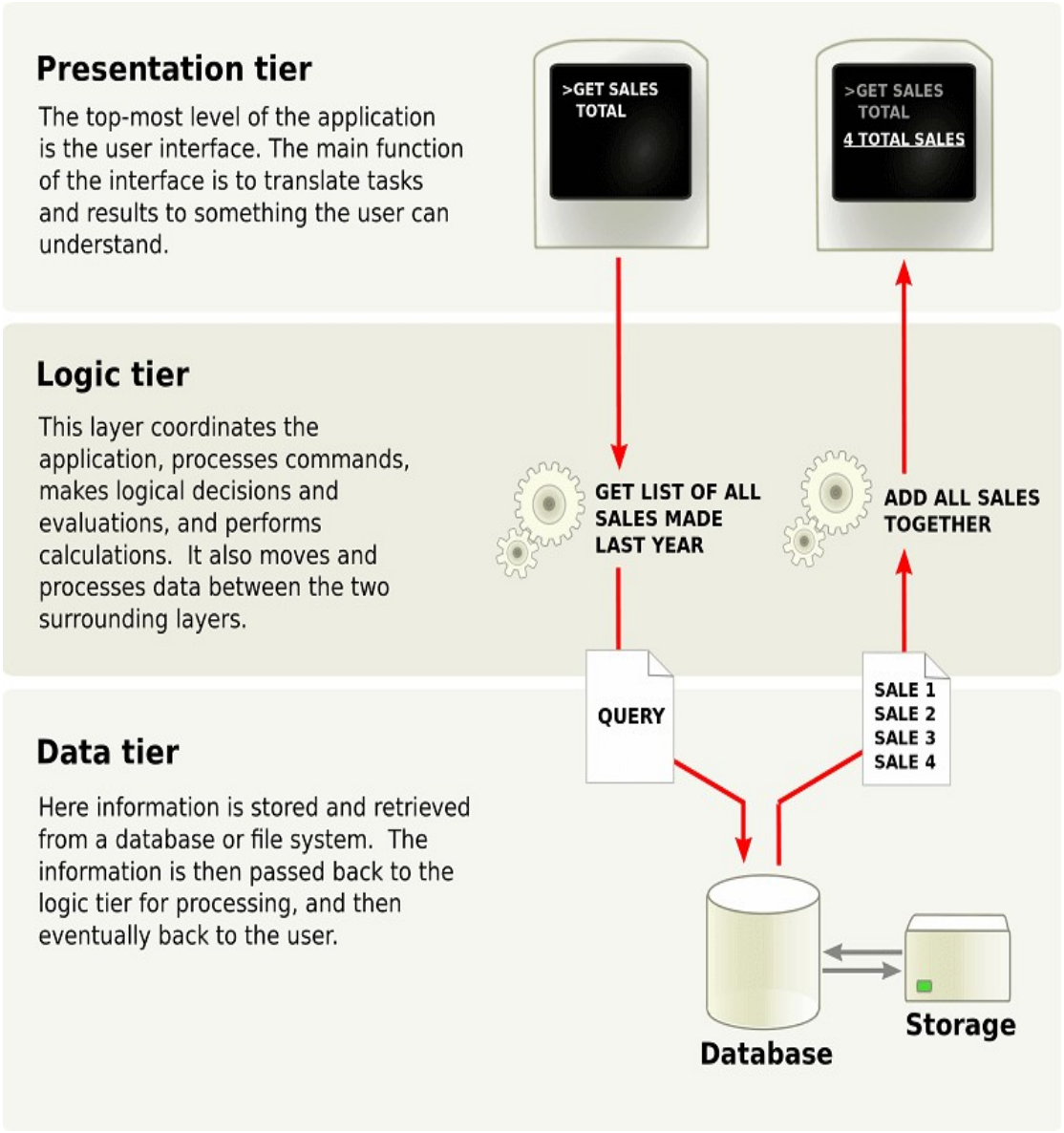
Ένα άλλο πολύ σημαντικό πλεονέκτημα είναι ότι έχουμε την δυνατότητα independent deploy του κάθε tier χωρίς να χρειάζεται να εγκαταστήσουμε/deploy όλη την εφαρμογή από την αρχή (όπως στην αρχιτεκτονική μονολιθικού τύπου). Στην περίπτωση της αρχιτεκτονικής N-tier μπορεί ένα tier να αντικατασταθεί πλήρως χωρίς να επηρεαστεί η λειτουργία των υπολοίπων. Για να εξασφαλιστεί το συγκεκριμένο είναι πολύ σημαντικό η όλη επικοινωνία να γίνετε με προσυμφωνημένα Interfaces/contracts έτσι ώστε αν εξασφαλίζετε το connectivity/modularity.

Αντίστοιχα στην περίπτωση προβλήματος σε κάποιο tier χρειάζεται εμπλοκή τεχνικών για debugging μόνο στο συγκεκριμένο tier, χωρίς να επηρεάζονται τα υπολοιπα. Βλέπουμε λοιπόν ότι μέσω της αρχιτεκτονικής αυτήν καθίσταται δυνατό να γίνουν όλες οι υπηρεσίες παραμετροποίησης/διόρθωσης/εγκατάστασης μιας εφαρμογής σε μονωμένα silo τα οποία παρέχουν την δυνατότητα ξεχωριστής διαχείρισης χωρίς να επηρεάζουν τα υπολοιπα, που είναι και η βασική διαφορά από την μονολιθική αρχιτεκτονική.

Υπάρχουν αρκετοί διαφορετικοί τύπου N-tier αρχιτεκτονικών αλλά συνήθως αυτές έχουνε η 3 η 4 ξεχωριστά tiers/layers ανάλογα με το που κατοικοεδρεύει το Business logic. Στην πράξη αυτό εξαρτάται και από την πολυπλοκότητα της εφαρμογής, πιο πολύπλοκες εφαρμογές τείνουν να έχουν περισσότερα tiers και μεγαλύτερη πολυπλοκότητα στη διαχείριση.

Μπορούμε επιγραμματικά να παραθέσουμε τα δυο μοντέλα παρακάτω:

1. Model 1
    a. Presentation Tier
        i. Responsible for accepting ready-to-be-presented data from Logic layer and render their view to the end user. It is highly recommended that this tier outsources all non-presentation related logic to previous tiers.
    b. Logic Tier
        i. Responsible for accepting the data from the lower tiers, manipulating them and bringing them to a form that can be used by the Presentation Tier to facilitate the end user's request. This is where all business logic lies in this kind of architecture.
    c. Data Tier
        i. Data Tier is responsible for interconnectivity with the various databases, as well as performing the necessary database operations to funnel data into the Logic Tier. The data tier traditionally is only concerned with the database and relevant queries.
2. Model 2
    a. Presentation Tier
        i. Responsible for accepting ready-to-be-presented data from Logic layer and render their view to the end user. It is highly recommended that this tier outsources all non-presentation related logic to previous tiers.
    b. Logic Tier
        i. Responsible for accepting the data from the lower tiers, manipulating them and bringing them to a form that can be used by the Presentation Tier to facilitate the end user's request. This is where all business logic lies in this kind of architecture.
    c. Persistence Tier
        i. Responsible for compiling database queries and performing the necessary manipulations on the data so they can be fed to the logic tier. It is the layer where the ORM resides and helps in decoupling the main application from the database.
    d. Data Tier
        i. This is where our physical database resides. This tier is not responsible for any queries, just the physical representation of the data, as well as inner relation db actions (indexing etc)



Εικόνα 5: Explanation for each tier of an N-tier application

Η διαφορετική φυσική τοποθεσία αυτών των επιπέδων είναι αυτό που διαφοροποιεί την αρχιτεκτονική ν-επιπέδων από άλλες αρχιτεκτονικές με παρεμφερή δομή (όπως είναι η MVC – Model View Controller, η οποία όμως μπορεί να αναφέρεται και σε μονολιθικού τύπου εφαρμογές. Σε μια αρχιτεκτονική ν-επιπέδων υπάρχει πάντα ένα logic layer το οποίο και διευθετεί όλη την επικοινωνία μεταξύ των διάφορων επιπέδων. Σε μια αντίστοιχη αρχιτεκτονική MVC αυτή η διεπαφη γίνεται σε τριγωνική μορφή, αντί να περάσει από το Logic-tier, το control layer είναι αυτό που έχει πρόσβαση στα μοντέλα καθώς και στα αντίστοιχα presentation views. Αντίστοιχα το Model layer είναι υπεύθυνο για την πρόσβαση στη βάση.

Οι δυο παραπάνω αρχιτεκτονικές δεν είναι αμοιβαία αποκλεισμένες και σε πολλές μοντέρνες εφαρμογές χρησιμοποιούνται ταυτόχρονα και το MVC και το N-tier. Είναι πολύ συνηθισμένο η όλη εφαρμογή να είναι στημένη σαν N-tier architecture και μόνο το Presentation layer να είναι Model View Controller (όπως είναι και η δικιά μας εργασία)

ΟΦΕΛΗ ΜΙΑΣ N-TIER ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ

Υπάρχουν πολλά οφέλη στην χρήση μιας n-tier αρχιτεκτονικής, αυτά σε γενικές γραμμές είναι:

1. Αυξημένες δυνατότητες διαχείρισης
    a. Κάθε επίπεδο μπορεί να έχει το δικό του διαχειριστικό, σε κάθε επίπεδο μπορεί να προστεθεί λειτουργικότητα χωρίς να επηρεαστούνε τα υπόλοιπα tiers
2. Ευελιξία
3. Επεκτασιμότητα
4. Αυξημένα επίπεδα ασφάλειας
    a. Το κάθε επίπεδο μπορεί να έχει διαφορετικό τρόπο εξασφάλισης και συνδυαστικά μπορούνε να επιτύχουνε πολύ υψηλά rating
    b. Μπορούν να χρησιμοποιηθούνε federated authentication tools σαν το Microsoft Entra / Amazon Directory services για να επιτύχουμε ακόμα μεγαλύτερη ασφάλεια.

Εν συντομία, με την αρχιτεκτονική N-tier μπορούμε να εντάξουμε νεες τεχνολογίες και να προσθέσουμε περισσότερα components χωρίς να χρειάζεται να ξαναγράψουμε ολόκληρη την εφαρμογή η να επανασχεδιάσουμε όλη την λύση, με βάση αυτή είναι ευκολότερο να κάνουμε scale την εφαρμογή και να την συντηρήσουμε. Παράλληλα, όσον αφορά την ασφάλεια παρέχεται η δυνατότητα να αποθηκεύσουμε confidential πληροφορία στο logic tier, απομακρύνοντας την από το presentation tier.

Άλλα οφέλη είναι:

1. Πιο αποτελεσματική υλοποίηση
    a. Η εφαρμογή αυτή είναι πολύ φιλική προς την υλοποίηση καθώς διαφορετικές ομάδες μπορούν να δουλέψουν σε διαφορετικά τμήματα της λύσης.
2. Εύκολη πρόσθεση νέας λειτουργικότητας
    a. Κατά την πρόσθεση νέας λειτουργικότητας, αυτή μπορεί να προστεθεί δεσμεύοντας μόνο το αντίστοιχο tier και όχι ολόκληρη την λύση.
3. Καλή δυνατότητα επαναχρησιμοποίησης
    a. Το κάθε tier της λύσης μπορεί να επαναχρησιμοποιηθεί δυνητικά στην υποστήριξη άλλων λύσεων

Η πιο διαδεδομένη παραλλαγή είναι η 3-tier application, όπου έχουμε 3 επίπεδα, το επίπεδο «παρουσίασης», το επίπεδο «Δεδομένων» και το αντίστοιχο επίπεδο λογικής της εφαρμογής. Ακολουθούνε συνοπτικές περίγραφες του κάθε επιπέδου:

1. Application-logic tier
    a. Εδώ γίνεται όλη η «σκέψη» της εφαρμογής, καθώς και όλο το validation. Στο επίπεδο αυτό παίρνονται η πλειοψηφία των αποφάσεων. Επίσης είναι υπεύθυνο για να γράψει/διαβάσει data στο data layer.
    b. Data-tier
        i. Εδώ γίνεται η αποθήκευση όλων των δεδομένων και είναι υπεύθυνο για όλα τα αντίστοιχα transaction με την βάση δεδομένων. Είναι Optimized να διατρέχει την βάση με πολύ καλό performance.
    c. Presentation-tier
        i. Είναι η διεπαφη με το χρήστη, και αποτελείται από όλες τις λειτουργικότητες εκείνες οι οποίες έχουνε άμεση δια-δράση με αυτών, από user information input μέχρι rendering γραφικών, αποτελεί των συνδετικό κρίκο μεταξύ του χρήστη και του logic-tier.

Υπάρχουν μοντέλα αρχιτεκτονικής n-tier που έχουν περισσότερα από τρία tiers. Παραδείγματα είναι εφαρμογές που έχουν αυτά τα tiers:

• Υπηρεσίες - όπως υπηρεσίες εκτύπωσης, καταλόγου ή βάσεις δεδομένων

• Επιχειρηματικός τομέας - το tier που θα φιλοξενεί αντικείμενα εξυπηρέτησης εφαρμογής όπως Java, DCOM, CORBA και άλλα.

• Επίπεδο παρουσίασης

• Επίπεδο πελάτη

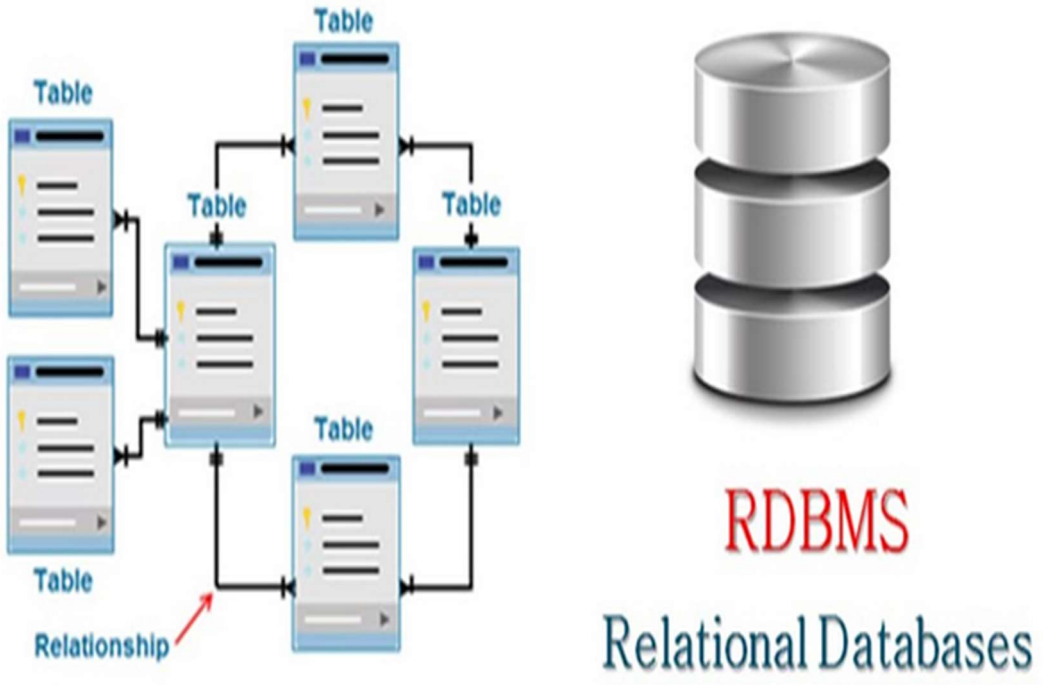
Ένα καλό παράδειγμα είναι όταν υπάρχει μια επιχειρησιακή αρχιτεκτονική που εξυπηρετεί υπηρεσίες. Το enterprise Service bus ή ESB θα υπάρχει ως ένα ξεχωριστό tier για να διευκολύνει την επικοινωνία του βασικού επιπέδου υπηρεσιών και του επιχειρησιακού τομέα. Σκέψεις για τη χρήση της αρχιτεκτονικής N-Tier για τις εφαρμογές. Επειδή λειτουργεί με πολλά tiers, πρέπει να εξασφαλιστεί ότι το εύρος ζώνης του δικτύου και το υλικό είναι γρήγορα.

Διαφορετικά, η απόδοση της εφαρμογής ενδέχεται να είναι αργή. Επίσης, αυτό θα σήμαινε ότι θα πρέπει να πληρώνονται περισσότερα για το δίκτυο, το υλικό και τη συντήρηση που απαιτείται για να διασφαλίσετε ότι έχετε καλύτερο εύρος ζώνης δικτύου.

Επίσης, είναι θεμιτό να χρησιμοποιήσουμε όσο το δυνατόν λιγότερα tiers. Το κάθε tier που προσθέτουμε στο λογισμικό ή στο έργο σημαίνει μια προστιθέμενη στρώση πολυπλοκότητας, περισσότερο υλικό για αγορά, καθώς και υψηλότερο κόστος συντήρησης και ανάπτυξης. Για να κάνετε τις εφαρμογές σας n-tier να έχουν νόημα, πρέπει να έχουν το ελάχιστο αριθμό tiers που χρειάζονται για να εξακολουθούν να απολαμβάνουν την επεκτασιμότητα, την ασφάλεια και τα άλλα οφέλη που φέρνει αυτή η αρχιτεκτονική.

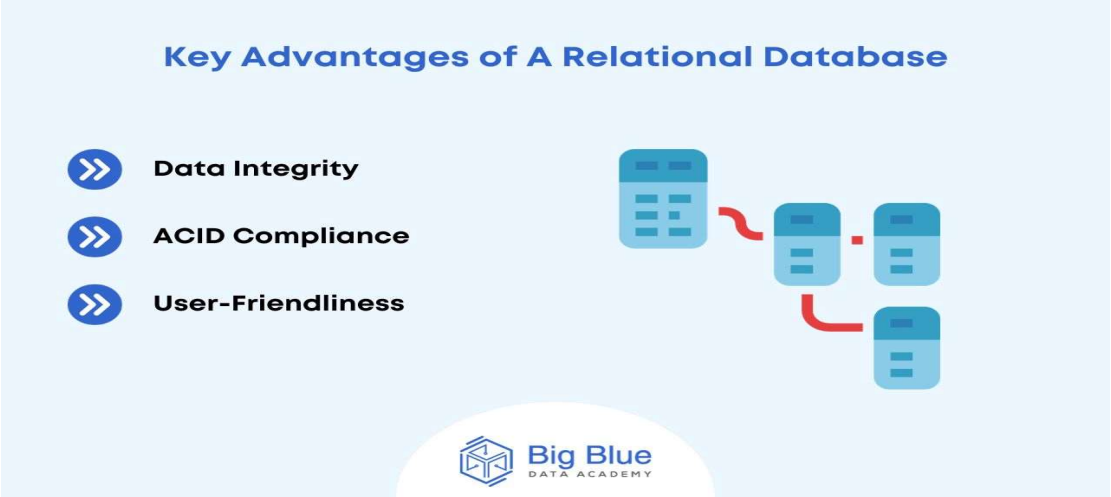
Σχεσιακές Βασεις Δεδομενων

Οι σχεσιακές βάσεις δεδομένων αποτελούν ένα θεμελιώδες στοιχείο της πληροφορικής και της επιστήμης των υπολογιστών. Αντιπροσωπεύουν μια δομή οργάνωσης δεδομένων όπου η πληροφορία οργανώνεται σε πίνακες που περιέχουν σειρές και στήλες. Η αξία των σχεσιακών βάσεων δεδομένων προέρχεται από τη δυνατότητά τους να διαχειρίζονται αποτελεσματικά μεγάλο όγκο πληροφοριών, να εξασφαλίζουν συνοχή και ακεραιότητα των δεδομένων, καθώς και να παρέχουν εύκολη πρόσβαση και ανάκτηση δεδομένων.



Εικόνα 6: Typical organization of a relational database

Κάθε σχεσιακή βάση δεδομένων αποτελείται από ένα σύνολο πινάκων. Κάθε πίνακας αποτελείται από σειρές και στήλες, όπου κάθε στήλη αντιπροσωπεύει έναν τύπο δεδομένων και κάθε σειρά αντιπροσωπεύει ένα εγγραφή. Το πρωτεύον κλειδί είναι ένας μοναδικός τρόπος για να αναγνωριστεί μια εγγραφή σε έναν πίνακα και να επιτραπεί η αναζήτηση και η αναφορά της.

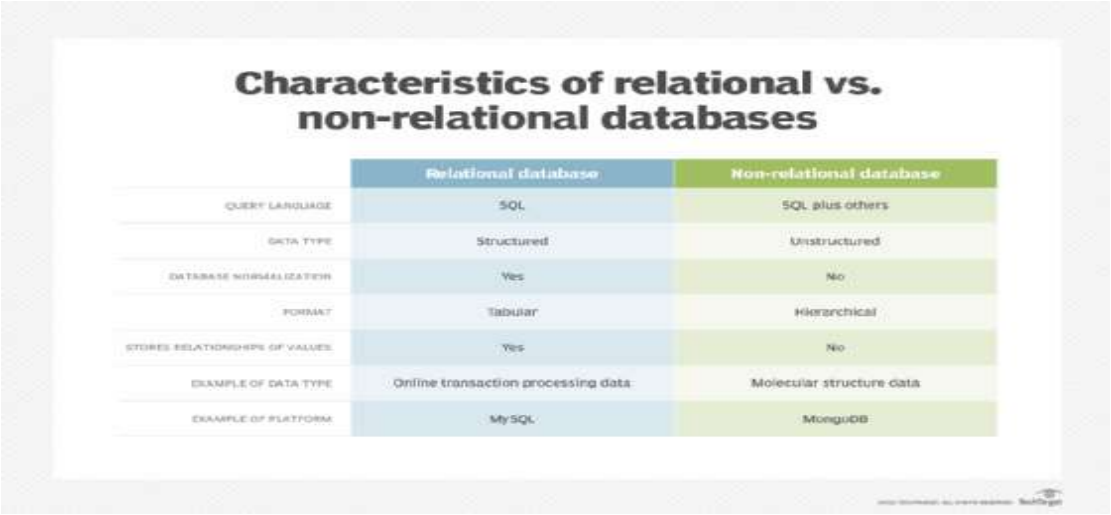


Εικόνα 7: Key advantages of a relational database

Οι σχεσιακές βάσεις δεδομένων χρησιμοποιούν τη γλώσσα SQL (Structured Query Language) για τη διαχείριση και την αλληλεπίδραση με τα δεδομένα. Η SQL παρέχει εντολές για τη δημιουργία, την τροποποίηση και τη διαγραφή δεδομένων, καθώς και για την επιλογή και την ενημέρωση των δεδομένων.

Η κανονικοποίηση είναι μια διαδικασία σχεδιασμού βάσης δεδομένων που στοχεύει στην ελαχιστοποίηση της επανάληψης δεδομένων και στη διατήρηση της συνέπειας των δεδομένων. Η κανονικοποίηση βοηθά στην αποφυγή των προβλημάτων που προκύπτουν από την αντίφαση των δεδομένων και βελτιστοποιεί την απόδοση της βάσης δεδομένων.

Οι σχεσιακές βάσεις δεδομένων υποστηρίζουν τη διατήρηση ακεραιότητας δεδομένων μέσω των διάφορων περιορισμών που μπορούν να οριστούν στους πίνακες, όπως ο περιορισμός της εισαγωγής τιμών, η επιβολή συνθηκών στην ενημέρωση και η δημιουργία συσχετίσεων μεταξύ των πινάκων.

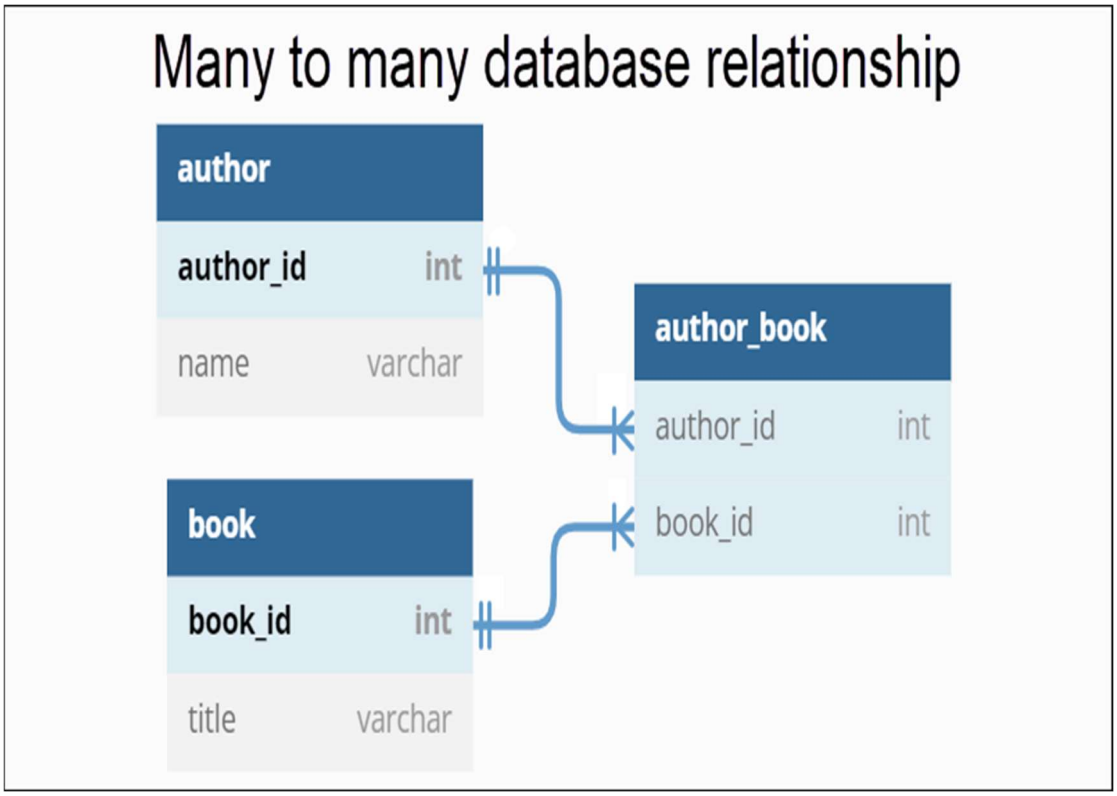


Εικόνα 8: Comparative analysis of relational versus non-relational databases

απαραίτητη για τη διατήρηση της ακεραιότητας και της εμπιστοσύνης στη βάση δεδομένων. Οι σχεσιακές βάσεις δεδομένων παρέχουν μηχανισμούς ασφάλειας όπως η διαχείριση των δικαιωμάτων πρόσβασης, ο ορισμός ρόλων χρηστών και η κρυπτογράφηση των δεδομένων.

Οι σχεσιακές βάσεις δεδομένων χρησιμοποιούν διάφορες τεχνικές βελτιστοποίησης της απόδοσης για την εκτέλεση των ερωτημάτων SQL. Αυτές περιλαμβάνουν τη χρήση δεικτών για την επιτάχυνση της αναζήτησης δεδομένων, την επιλογή κατάλληλων σχεδιαστικών προτύπων και τη βελτιστοποίηση των ερωτημάτων SQL μέσω της χρήσης εργαλείων εξαγωγής εκτέλεσης και στατιστικών.

Η ανάκτηση δεδομένων από μια σχεσιακή βάση δεδομένων μπορεί να γίνει με διάφορους τρόπους, συμπεριλαμβανομένης της χρήσης ερωτημάτων SQL, αποθηκευμένων διαδικασιών ή ακόμα και εφαρμογών που χρησιμοποιούν το API της βάσης δεδομένων.



Εικόνα 9: Typical many-to-many relationship

Η αντιγραφή ασφαλείας και η ανάκτηση είναι σημαντικές διαδικασίες για τη διασφάλιση της ακεραιότητας και της διαθεσιμότητας των δεδομένων. Οι σχεσιακές βάσεις δεδομένων παρέχουν μηχανισμούς αντιγραφής ασφαλείας και ανάκτησης, συμπεριλαμβανομένων των προγραμμάτων διαχείρισης βάσεων δεδομένων και εργαλείων αυτοματοποιημένης αντιγραφής ασφαλείας.

Τέλος, οι σχεσιακές βάσεις δεδομένων συνεργάζονται συχνά με άλλα συστήματα πληροφορικής όπως εφαρμογές λογισμικού και διαδικτυακές εφαρμογές. Οι διασυνδέσεις μεταξύ των συστημάτων επιτρέπουν την ανταλλαγή δεδομένων και την ολοκλήρωση των λειτουργιών, επιτρέποντας έτσι την ομαλή λειτουργία και την αύξηση της αποτελεσματικότητας των οργανισμών.

Συνοψίζοντας:

Οι σχεσιακές βάσεις δεδομένων οργανώνουν τα δεδομένα σε πίνακες με στήλες και σειρές.

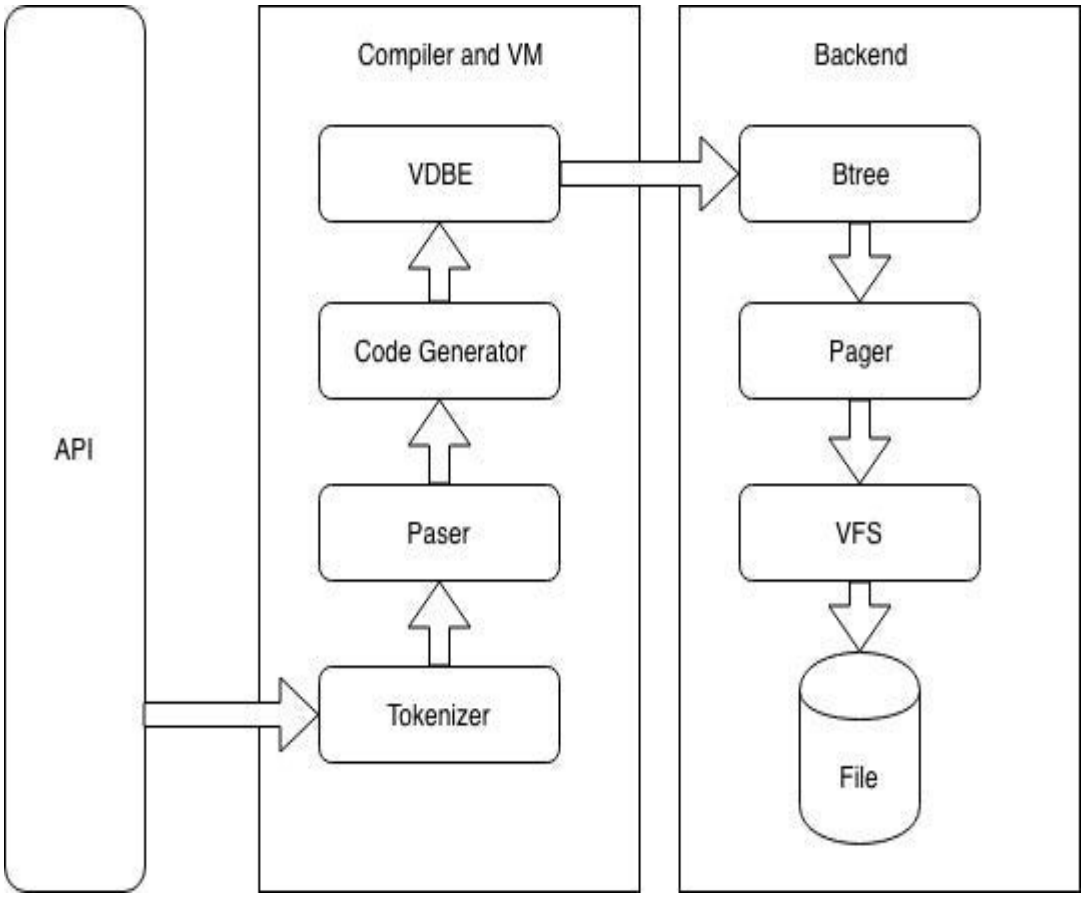
- Κάθε πίνακας περιλαμβάνει διακριτά στοιχεία δεδομένων, με κάθε στήλη να αντιστοιχεί σε έναν τύπο δεδομένων.
- Η SQL είναι η γλώσσα που χρησιμοποιείται για τη διαχείριση και την ανάκτηση δεδομένων από σχεσιακές βάσεις.
- Το πρωτεύον κλειδί αναγνωρίζει μοναδικά κάθε εγγραφή σε έναν πίνακα.
- Η κανονικοποίηση είναι η διαδικασία με την οποία εξαλείφονται οι πολλαπλές επαναλήψεις δεδομένων και διατηρείται η συνέπεια των δεδομένων.
- Οι σχεσιακές βάσεις υποστηρίζουν περιορισμούς δεδομένων για τη διατήρηση ακεραιότητας.
- Η ασφάλεια των δεδομένων περιλαμβάνει την οριοθέτηση δικαιωμάτων πρόσβασης για τους χρήστες.
- Οι σχεσιακές βάσεις είναι αποτελεσματικά στη διαχείριση μεγάλου όγκου πληροφοριών.
- Οι ευέλικτες δυνατότητες επεξεργασίας και ανάκτησης δεδομένων τις καθιστούν ιδανικές για ποικίλες εφαρμογές.
- Η σχεσιακή δομή παρέχει ευκολία στην ανάπτυξη και συντήρηση των βάσεων δεδομένων.

Για την σχεδίαση και την αποτύπωση σχέσεων σε μια βάση ακολουθούνται οι παρακάτω κανόνες:

- **Απεικόνιση Δομής:** Τα διαγράμματα οντοτήτων-σχέσεων (ER) απεικονίζουν τη δομή της βάσης δεδομένων, παρουσιάζοντας τις οντότητες, τις σχέσεις και τα γνωρίσματα.
- **Οντότητες**: Αντιπροσωπεύουν αντικείμενα ή έννοιες στον κόσμο της εφαρμογής και απεικονίζονται ως ορθογώνια στο διάγραμμα.
- **Γνωρίσματα Οντοτήτων:** Κάθε οντότητα έχει γνωρίσματα που περιγράφουν τις ιδιότητές της και απεικονίζονται ως πεδία στα ορθογώνια.
- **Κλειδιά:** Κάθε οντότητα έχει ένα ή περισσότερα κλειδιά που την αναγνωρίζουν μοναδικά, συμπεριλαμβανομένων των πρωτευόντων και των δευτερευόντων κλειδιών.
- **Σχέσεις:** Αντιπροσωπεύουν τις συνδέσεις μεταξύ δύο οντοτήτων και απεικονίζονται ως διακεκομμένες γραμμές.
- **Τύποι Σχέσεων:** Οι σχέσεις μπορούν να είναι μονόπλευρές ή διπλής κατεύθυνσης, με μονό ή πολλαπλά γνωρίσματα.
- **Συσχέτιση Οντοτήτων:** Οι οντότητες συσχετίζονται μέσω των σχέσεων, καθορίζοντας πώς συνδέονται μεταξύ τους.
- **Γνωρίσματα Σχέσεων:** Τα γνωρίσματα που αποτελούν το κλειδί μιας σχέσης περιγράφουν την κατάσταση της συσχέτισης μεταξύ των οντοτήτων.
- **Κανόνες Συσχέτισης**: Οι κανόνες συσχέτισης καθορίζουν τον τρόπο με τον οποίο οι οντότητες συνδέονται μεταξύ τους, όπως ονόματα και περιορισμοί συσχέτισης.
- **Απεικόνιση Περιορισμών:** Οι περιορισμοί συχνά απεικονίζονται στο διάγραμμα ER, προσθέτοντας πληροφορίες σχετικά με τους περιορισμούς ακεραιότητας και συνέπειας στη βάση δεδομένων.

SQLITE

Το SQLite αποτελεί ένα από τους πιο δημοφιλή συστήματα διαχείρισης βάσεων δεδομένων (RDBMS), που επικροτείται ευρέως για την ελαφρότητα, την μη αναγκη για extra dependencies και την αρχιτεκτονική χωρίς εξυπηρετητή/server. Είναι ένα ανοικτού κώδικα, μηδενικής διαμόρφωσης, transactional σύστημα βάσης δεδομένων που δεν απαιτεί ξεχωριστή διαδικασία server, καθιστώντας το ιδανική επιλογή για ενσωματωμένες εφαρμογές βάσεων δεδομένων, κινητές εφαρμογές και ιστότοπους μικρής έως μεσαίας κλίμακας.

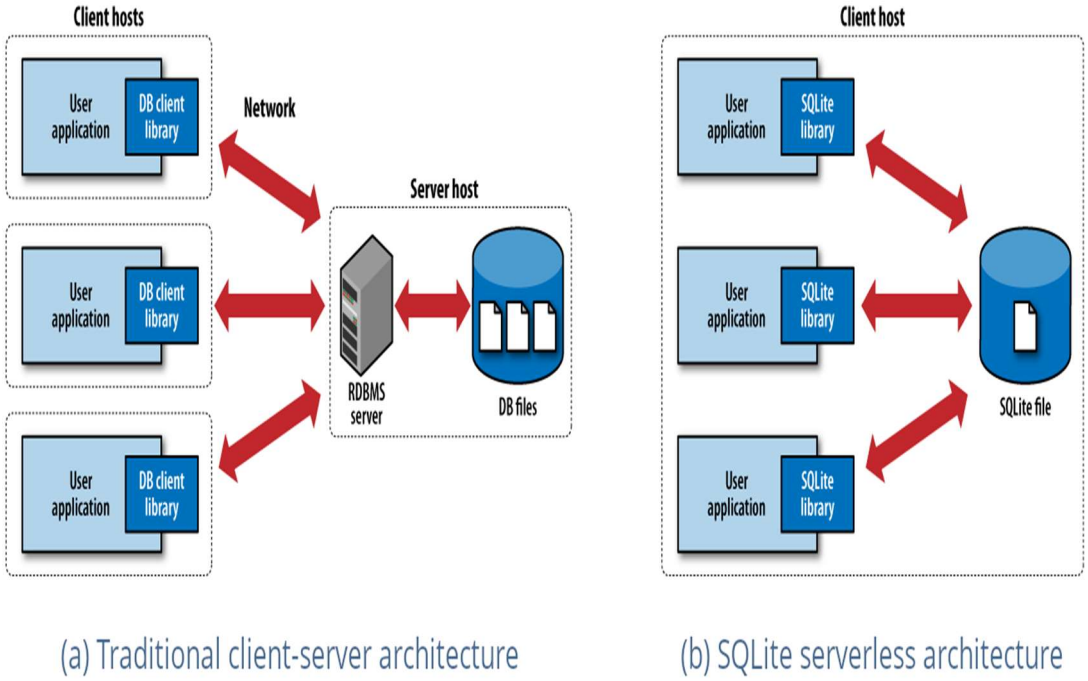


Εικόνα 10: Function diagram of a sqlite db

Βασικά Χαρακτηριστικά του SQLite:

- No external dependencies: Το SQLite λειτουργεί ως ένα μόνο, αυτό-περιεκτικό αρχείο, χωρίς εξωτερικές εξαρτήσεις ή αρχεία διαμόρφωσης που απαιτούνται. Αυτή η απλότητα διευκολύνει τις διαδικασίες ανάπτυξης και συντήρησης.
- Χωρίς Εξυπηρετητή: Αντίθετα με τις παραδοσιακές βάσεις δεδομένων πελάτη-εξυπηρετητή, το SQLite λειτουργεί χωρίς εξυπηρετητή, δηλαδή αποκτά πρόσβαση απευθείας στα αρχεία αποθήκευσής του χωρίς την ανάγκη για ξεχωριστή διαδικασία εξυπηρετητή. Αυτό μειώνει τις επιβαρύνσεις και ελαχιστοποιεί τις διαδικασίες εγκατάστασης.

- Μηδενική Διαμόρφωση: Το SQLite δεν απαιτεί καμία ρύθμιση ή διαχείριση. Μπορείτε απλά να συμπεριλάβετε τη βιβλιοθήκη SQLite στην εφαρμογή σας και να ξεκινήσετε να το χρησιμοποιείτε αμέσως.
- Διασυνδεσιμότητα: Το SQLite είναι υψηλά φορητό και συμβατό με διάφορα λειτουργικά συστήματα, συμπεριλαμβανομένων των Windows, macOS, Linux, iOS και Android. Αυτό εξασφαλίζει τη συνεχή ενσωμάτωση σε διαφορετικά περιβάλλοντα και πλατφόρμες.



Εικόνα 11: Sqlite serverless architecture

Πλεονεκτήματα της Χρήσης του SQLite:

- Low-resource solution: Το SQLite σχεδιάστηκε να είναι ελαφρύ, με ελάχιστη χρήση μνήμης και χώρου δίσκου. Αυτό το καθιστά εξαιρετική επιλογή για περιορισμένους πόρους περιβάλλοντα όπως κινητές συσκευές και ενσωματωμένα συστήματα.
- Υψηλή Απόδοση: Παρά το μικρό του μέγεθος, το SQLite προσφέρει εντυπωσιακή απόδοση, ιδιαίτερα για φορτία εργασιών ανάγνωσης. Η αποτελεσματική επεξεργασία ερωτημάτων και οι μηχανισμοί δεικτοδότησης συμβάλλουν στη γρήγορη ανάκτηση και επεξεργασία δεδομένων.
- Εύκολο στη Χρήση: Η απλότητα και η ευκολία χρήσης του SQLite το καθιστούν προσβάσιμο για προγραμματιστές όλων των επιπέδων. Η διεπαφή του βασίζεται στη SQL και η σχετική τεκμηρίωση διευκολύνει την ταχεία ανάπτυξη και το πρωτότυπο.
- Χαμηλές Διαδικασίες Συντήρησης: Με την απουσία διαδικασίας εξυπηρετητή για να διαχειριστεί, οι βάσεις δεδομένων SQLite απαιτούν ελάχιστη συντήρηση. Οι ενημερώσεις και οι αντίγραφα ασφαλείας είναι απλές και δεν υπάρχει ανάγκη για συνεχή παρακολούθηση ή βελτιστοποίηση απόδοσης.
- Ασφάλεια Δεδομένων: Το SQLite παρέχει ενσωματωμένη υποστήριξη για κρυπτογράφηση δεδομένων, επιτρέποντας στους προγραμματιστές να ασφαλίσουν τις ευαίσθητες πληροφορίες που αποθηκεύονται στη βάση δεδομένων. Επιπλέον, το

μοντέλο αποθήκευσης βασισμένο σε αρχεία απλοποιεί την απομόνωση και τον έλεγχο πρόσβασης στα δεδομένα.

- Καλες δυνατοτητες για scaling: Παρόλο που το SQLite σχεδιάστηκε κυρίως για εφαρμογές μικρής έως μεσαίας κλίμακας, μπορεί να διαχειριστεί σύνολα δεδομένων με μεγάλο μέγεθος αποτελεσματικά. Με κατάλληλο σχεδιασμό σχήματος και τεχνικές βελτιστοποίησης, οι βάσεις δεδομένων SQLite μπορούν να κλιμακωθούν για να ανταποκριθούν στις αυξημένες απαιτήσεις.
- Ευρεία Υιοθέτηση και Υποστήριξη από την Κοινότητα: Το SQLite διαθέτει μια μεγάλη και ενεργή κοινότητα χρηστών, με εκτενή τεκμηρίωση, εκπαιδευτικά βίντεο και διαδικτυακούς πόρους διαθέσιμους. Αυτή η δυναμική κοινότητα εξασφαλίζει τη συνεχή υποστήριξη, διορθώσεις σφαλμάτων και βελτιώσεις χαρακτηριστικών.
- Αξιοπιστία και Σταθερότητα: Το SQLite είναι γνωστό για την ανθεκτικότητά του και τη σταθερότητά του, με ένα αποδεδειγμένο ιστορικό σε περιβάλλοντα παραγωγής σε διάφορες βιομηχανίες. Η αυστηρή διαδικασία δοκιμής και οι συχνές ενημερώσεις εγγυώνται μια αξιόπιστη λύση βάσης δεδομένων.

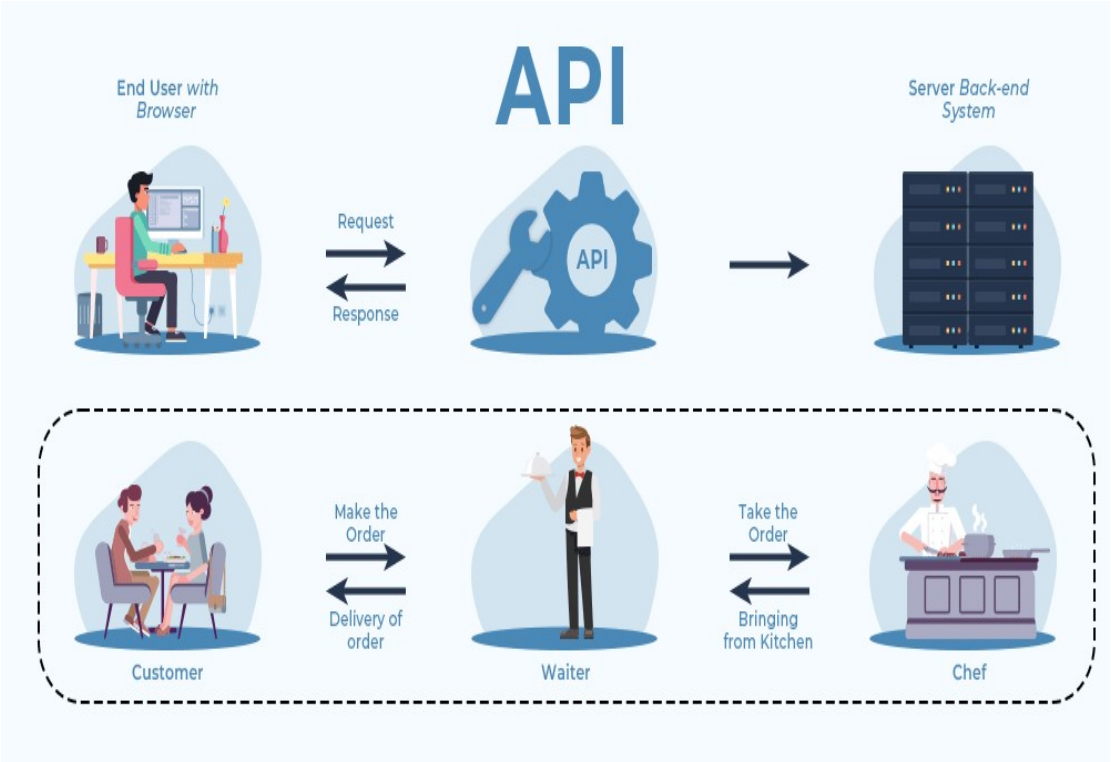
Συμπεράσματα:

Συνολικά, το SQLite προσφέρει μια ελκυστική συνδυασμό απλότητας, απόδοσης και αξιοπιστίας, κάνοντάς το προτιμώμενη επιλογή για μια ευρεία γκάμα εφαρμογών. Είτε δημιουργείτε μια κινητή εφαρμογή, διαχειρίζεστε συσκευές IoT ή πρωτοτυπείτε έναν ιστότοπο, το SQLite παρέχει την ευελιξία και την αποτελεσματικότητα που χρειάζεστε για να επιτύχετε. Αξιοποιώντας τη δύναμη του SQLite, οι προγραμματιστές μπορούν να επικεντρωθούν στην καινοτομία και στην παροχή εξαιρετικών εμπειριών χρηστών χωρίς να περιορίζονται από τις πολυπλοκότητες της διαχείρισης βάσεων δεδομένων.

Api Communications

Επικοινωνία API στην Πληροφορική:

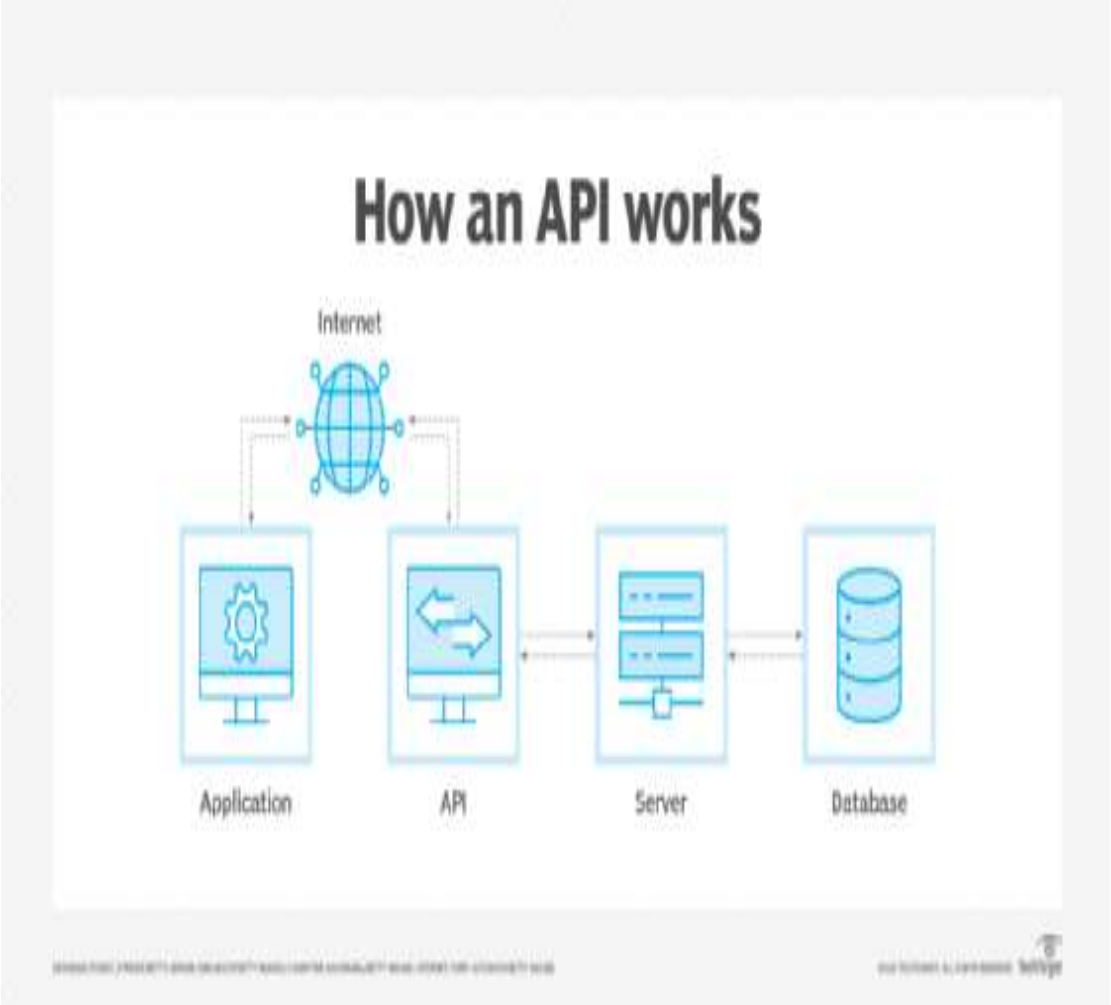
Η επικοινωνία μεταξύ εφαρμογών μέσω διαδικτυακών διεπαφών εφαρμογών (API) είναι βασική στη σύγχρονη πληροφορική. Οι API επιτρέπουν σε διάφορες εφαρμογές να ανταλλάσσουν δεδομένα με αξιοπιστία και αποτελεσματικότητα. Μία από τις δημοφιλέστερες κατηγορίες API είναι αυτές που παρέχουν υπηρεσίες κοινωνικών δικτύων όπως το Facebook, το Twitter και το Instagram, επιτρέποντας την πρόσβαση σε δεδομένα χρηστών και τη δημιουργία εφαρμογών που αλληλοεπιδρούν με τα δίκτυα αυτά. Ωστόσο, η αρχιτεκτονική REST (Representational State Transfer) έχει αποκτήσει μεγάλη δημοτικότητα στον χώρο των API, βασιζόμενη σε αρχές που προάγουν την απλότητα και την ευκολία επέκτασης. Αυτό επιτρέπει στις εφαρμογές να επικοινωνούν μεταξύ τους με αποτελεσματικό και εύκολο τρόπο.



Εικόνα 12: Typical Api usage cases

Χρήση του JSON:

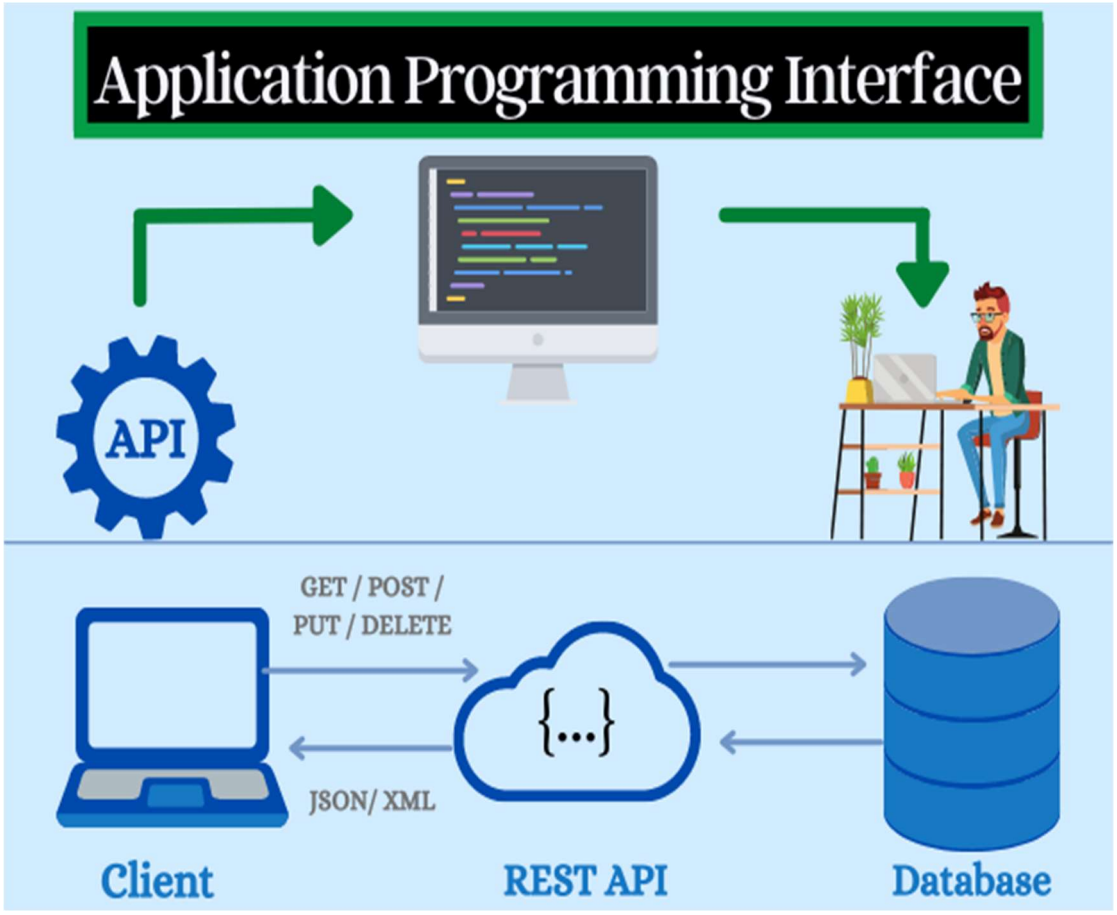
Η χρήση του JSON (JavaScript Object Notation) είναι διαδεδομένη στην ανταλλαγή δεδομένων μεταξύ εφαρμογών. Το JSON προσφέρει έναν εύκολο τρόπο παρουσίασης και ανάγνωσης των δεδομένων, βοηθώντας στη διευκόλυνση της επικοινωνίας. Είναι ένας ελαφρύς, ανθεκτικός και ευανάγνωστος τρόπος ανταλλαγής δεδομένων μεταξύ διαφορετικών εφαρμογών και συστημάτων. Χρησιμοποιείται ευρέως σε ιστοσελίδες, κινητά τηλέφωνα, διακομιστές και υπηρεσίες cloud. Το JSON είναι ανεξάρτητο από γλώσσες προγραμματισμού και εύκολο στην κατανόηση, κάτι που το καθιστά ιδανικό για την ανταλλαγή δεδομένων σε πολυδιάστατες εφαρμογές.



Εικόνα 13: Typical api function for integration

Χρήση του JWT:

Τα JWT (JSON Web Tokens) είναι ένας τρόπος τεκμηρίωσης που χρησιμοποιείται για την ασφαλή αυθεντικοποίηση και εξουσιοδότηση των χρηστών. Είναι ένας τύπος token που περιέχει πληροφορίες για τον χρήστη και την πρόσβασή του σε συγκεκριμένους πόρους. Τα JWT περιέχουν υπογεγραμμένες πληροφορίες που είναι ασφαλείς για την ανταλλαγή μεταξύ των διάφορων συστημάτων.



Εικόνα 14: Api methods in typical api functions (JSON/XML)

Ευχαριστώ για τη διευκρίνιση. Ας προσθέσουμε άλλες επτά βασικές πληροφορίες για να ολοκληρώσουμε τις 10 δηλώσεις:

Εφαρμογές του JSON:

Οι εφαρμογές που χρησιμοποιούν JSON ως μέσο ανταλλαγής δεδομένων είναι πολλές και ποικίλες. Από ιστοσελίδες και κινητά τηλέφωνα έως διακομιστές και υπηρεσίες cloud, το JSON χρησιμοποιείται ευρέως για τη μεταφορά δεδομένων. Η δομή του JSON επιτρέπει την ανταλλαγή πολύπλοκων δεδομένων με απλό και κατανοητό τρόπο.

Σημασία της Ασφάλειας στα API:

Η ασφάλεια αποτελεί κρίσιμο παράγοντα για τα API. Η προστασία των δεδομένων και η αποτροπή ανεπιθύμητων πρόσβασης είναι πρωταρχικές ανησυχίες. Η χρήση μηχανισμών όπως η κρυπτογράφηση και η αυθεντικοποίηση είναι ζωτικής σημασίας για τη διασφάλιση της ασφάλειας των δεδομένων σε επίπεδο API.

Ανάπτυξη Προσαρμοσμένων API:

Η δημιουργία προσαρμοσμένων API επιτρέπει τη δημιουργία εφαρμογών που ικανοποιούν συγκεκριμένες ανάγκες. Με την προσαρμογή των διεπαφών, οι προγραμματιστές μπορούν να δημιουργήσουν εφαρμογές που είναι αποδοτικές και αποτελεσματικές στην επίλυση συγκεκριμένων προβλημάτων.

Ενσωμάτωση με άλλες Τεχνολογίες:

Τα API συχνά ενσωματώνονται με άλλες τεχνολογίες, όπως οι βάσεις δεδομένων, οι υπηρεσίες cloud και τα συστήματα αυθεντικοποίησης. Η σωστή ενσωμάτωση και η συμβατότητα με άλλες τεχνολογίες είναι ζωτικής σημασίας για τη συνολική λειτουργία του συστήματος.

Επίλυση Προβλημάτων και Σφαλμάτων:

Η επίλυση προβλημάτων και σφαλμάτων είναι αναπόφευκτη στην ανάπτυξη και λειτουργία των API. Η ικανότητα ταχείας διάγνωσης και επίλυσης προβλημάτων είναι καθοριστική για τη διατήρηση της αξιοπιστίας των εφαρμογών.

Μελλοντικές Τάσεις στα API:

Με τη συνεχή εξέλιξη της τεχνολογίας, τα API αναμένεται να εξελίσσονται και να προσαρμόζονται σε νέες απαιτήσεις. Τάσεις όπως η μεγαλύτερη αυτοματοποίηση, η επέκταση σε έξυπνες συσκευές και η ανάπτυξη πιο αποδοτικών μοντέλων ασφάλειας αναμένονται να επηρεάσουν τη μελλοντική ανάπτυξη των API.

Χρήση του JWT (JSON Web Tokens):

Τα JSON Web Tokens (JWTs) είναι ένας τρόπος τεκμηρίωσης που χρησιμοποιείται για την ασφαλή αυθεντικοποίηση και εξουσιοδότηση των χρηστών. Αυτά τα tokens περιέχουν πληροφορίες για τον χρήστη και την πρόσβασή του σε συγκεκριμένους πόρους. Τα JWT περιλαμβάνουν υπογεγραμμένες πληροφορίες που είναι ασφαλείς για την ανταλλαγή μεταξύ των διαφόρων συστημάτων. Αυτό επιτρέπει την ασφαλή μεταφορά πληροφοριών μεταξύ διαφορετικών εφαρμογών και συστημάτων.

Δημοφιλή APIs:

Υπάρχουν πολλά δημοφιλή APIs που προσφέρουν διαφορετικές υπηρεσίες και δυνατότητες. Τα APIs κοινωνικών δικτύων όπως το Facebook, το Twitter και το Instagram επιτρέπουν την πρόσβαση σε δεδομένα χρηστών και τη δημιουργία εφαρμογών που αλληλοεπιδρούν με τα δίκτυα αυτά. Άλλα δημοφιλή APIs περιλαμβάνουν τα Google Maps API, το YouTube API και το PayPal API, που προσφέρουν δυνατότητες χαρτογράφησης, πρόσβαση σε βίντεο και διαχείριση πληρωμών αντίστοιχα.

Παρακάτω αναφέρονται τα πιο δημοφιλή api:

- **RESTful APIs:**
    - Τα RESTful APIs αντιπροσωπεύουν έναν δημοφιλή τύπο διεπαφής προγραμματισμού εφαρμογών που βασίζεται στην αρχιτεκτονική REST (Representational State Transfer). Ένα από τα βασικά χαρακτηριστικά τους είναι η αυτονομία και η ανεξαρτησία από την κατάσταση, καθώς κάθε αίτημα που στέλνεται περιέχει όλες τις πληροφορίες που απαιτούνται για την επεξεργασία του. Επιπλέον, τα RESTful APIs χρησιμοποιούν το πρωτόκολλο HTTP, προσφέροντας ένα ευέλικτο και εύκολο στη χρήση περιβάλλον για την ανάπτυξη web services. Η αρχιτεκτονική τους διευκολύνει την ανάπτυξη εφαρμογών που επικοινωνούν με άλλες εφαρμογές μέσω του διαδικτύου, επιτρέποντας την εύκολη ανταλλαγή δεδομένων μεταξύ διαφορετικών συστημάτων.
- **GraphQL APIs:**

- Το GraphQL είναι ένα ευέλικτο και ισχυρό πλαίσιο επικοινωνίας για τη μεταφορά δεδομένων από έναν εξυπηρετητή σε έναν πελάτη. Το βασικό χαρακτηριστικό του GraphQL είναι η δυνατότητα των πελατών να ορίζουν ακριβώς τα δεδομένα που επιθυμούν να λάβουν, ελαχιστοποιώντας τον όγκο των δεδομένων που μεταφέρονται με κάθε αίτημα. Αυτό το κάνει κατάλληλο για εφαρμογές που απαιτούν αποδοτική μεταφορά δεδομένων, όπως οι εφαρμογές κινητών τηλεφώνων ή οι εφαρμογές web. Επιπλέον, ο τρόπος με τον οποίο λειτουργεί το GraphQL επιτρέπει την εύκολη επέκταση και ανάπτυξη των API χωρίς την ανάγκη για συνεχείς αλλαγές στη δομή των δεδομένων.

- **SOAP APIs:**
  - Τα SOAP APIs είναι ένας παλαιότερος τύπος διεπαφής προγραμματισμού εφαρμογών που χρησιμοποιείται κυρίως για τη δημιουργία επιχειρηματικών εφαρμογών. Αν και μπορεί να θεωρηθεί πιο περίπλοκο σε σύγκριση με άλλους τύπους API όπως τα RESTful APIs, το SOAP προσφέρει μια πλούσια σειρά λειτουργιών και επιλογών για τη δημιουργία πολύπλοκων εφαρμογών. Επιπλέον, το SOAP προσφέρει υψηλό επίπεδο ασφάλειας και αξιοπιστίας, καθιστώντας το κατάλληλο για εφαρμογές που απαιτούν αξιόπιστη μεταφορά δεδομένων και ευαισθησία στην ασφάλεια δεδομένων.
- **Webhooks:**
  - Τα Webhooks είναι μηχανισμοί που επιτρέπουν στις εφαρμογές να επικοινωνούν αυτόματα και να αντιδρούν όταν συμβαίνουν συγκεκριμένα γεγονότα. Ενώ ένα API χρειάζεται συνήθως να κληθεί από μια εφαρμογή για να λάβει δεδομένα, ένα Webhook αναμένει σε ένα συγκεκριμένο URL και ενεργοποιείται αυτόματα όταν συμβεί ένα γεγονός. Αυτό το καθιστά ιδιαίτερα χρήσιμο για εφαρμογές που χρειάζονται άμεσες αντιδράσεις σε εξωτερικά γεγονότα, όπως οι εφαρμογές κοινωνικών μέσων ή οι εφαρμογές παρακολούθησης.

Αρχιτεκτονική REST (Representational State Transfer):

Η αρχιτεκτονική REST έχει αποκτήσει μεγάλη δημοτικότητα στον χώρο των API. Βασίζεται σε αρχές που προάγουν την απλότητα και την ευκολία επέκτασης. Αυτό επιτρέπει στις εφαρμογές να επικοινωνούν μεταξύ τους με αποτελεσματικό και εύκολο τρόπο. Τα APIs που βασίζονται στην αρχιτεκτονική REST χρησιμοποιούν τους βασικούς μεθόδους HTTP, όπως GET, POST, PUT και DELETE, για να διαχειριστούν τις αιτήσεις και τις απαντήσεις. Η αρχιτεκτονική REST επιτρέπει τη δημιουργία ευέλικτων και εύκολα επεκτάσιμων συστημάτων.

Ανάπτυξη και Διαχείριση των Συνολικών API:

Η ανάπτυξη και η διαχείριση των συνολικών API απαιτεί συστηματική προσέγγιση και εξειδικευμένα εργαλεία. Η χρήση διαφόρων πλατφορμών και εργαλείων διαχείρισης API μπορεί να βοηθήσει στη διαχείριση των διαφορετικών API που χρησιμοποιούνται σε ένα έργο ή μια επιχείρηση.

Προστασία των Δεδομένων Χρηστών:

Η προστασία των δεδομένων των χρηστών αποτελεί κεντρικό ζήτημα στην ανάπτυξη API. Η χρήση προηγμένων τεχνικών κρυπτογράφησης και διαδικασιών αυθεντικοποίησης είναι απαραίτητη για τη διασφάλιση της εμπιστευτικότητας και της ιδιωτικότητας των προσωπικών δεδομένων.

Συμμόρφωση με Κανονιστικά Πλαίσια:

Η συμμόρφωση με κανονιστικά πλαίσια ασφαλείας και προστασίας δεδομένων, όπως ο Γενικός Κανονισμός για την Προστασία των Δεδομένων (GDPR), απαιτεί προσεκτικό σχεδιασμό και υλοποίηση των API. Η τήρηση των νομικών απαιτήσεων είναι ζωτικής σημασίας για την αποφυγή πιθανών προστίμων και επιπλέον οικονομικών απωλειών.

Docker & docker_compose

Εισαγωγή στο Docker:

Το Docker είναι μια πλατφόρμα λογισμικού που επιτρέπει την εύκολη δημιουργία, την παράδοση και την εκτέλεση εφαρμογών χρησιμοποιώντας τεχνολογίες ελαφρού virtualization. Με το Docker, μπορείτε να εκτελέσετε εφαρμογές σε οποιοδήποτε περιβάλλον χωρίς να χρειάζεται να ανησυχείτε για τις διαφορές στη ρύθμιση ή τις εξαρτήσεις.

Πλεονεκτήματα του Docker:

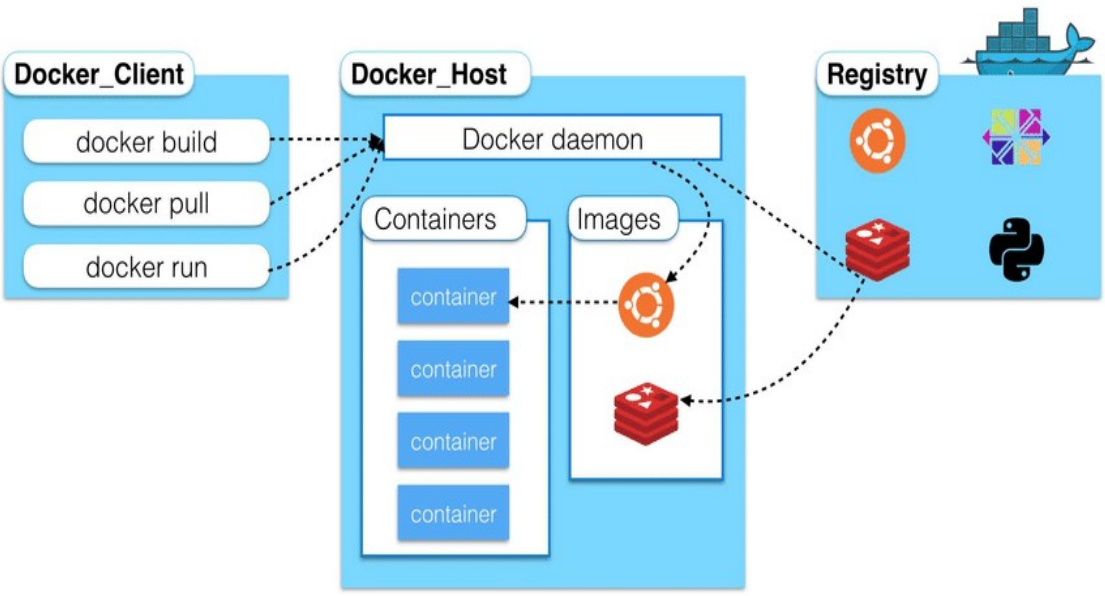
Ένα από τα κύρια πλεονεκτήματα του Docker είναι η απομόνωση των εφαρμογών σε ελαφριές εικόνες, οι οποίες είναι φορητές και ανεξάρτητες από το περιβάλλον εκτέλεσης.

Διαδικασία Εργασίας με το Docker:

Η διαδικασία εργασίας με το Docker περιλαμβάνει τη δημιουργία των Dockerfile, των αρχείων που περιγράφουν το περιβάλλον εκτέλεσης της εφαρμογής, και την κατασκευή των εικόνων Docker.

Διαχείριση Εικόνων Docker:

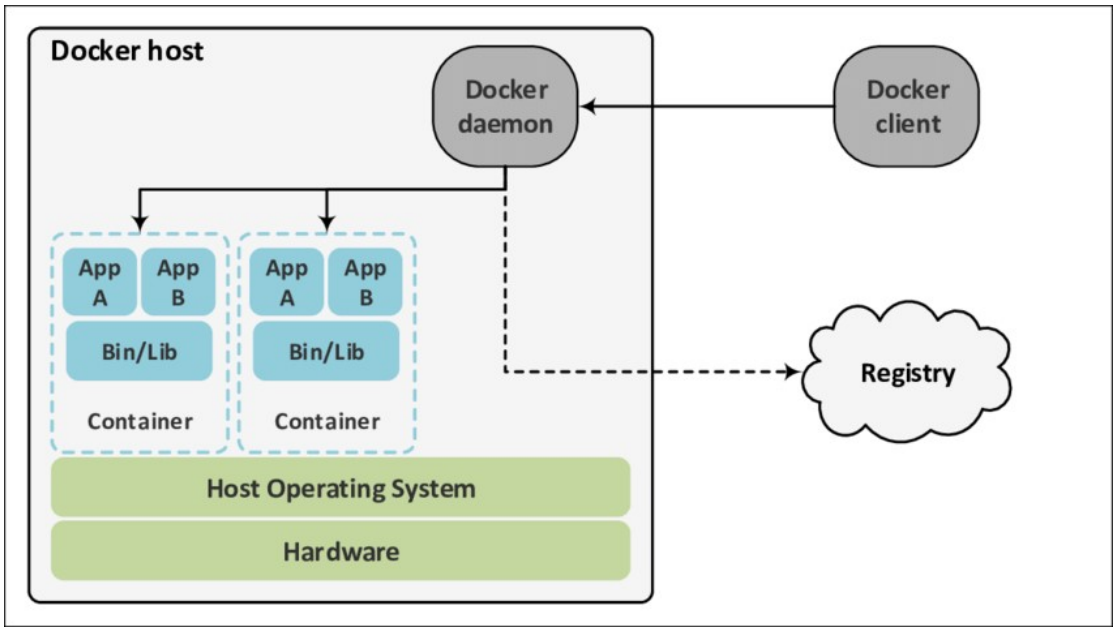
Οι εικόνες Docker μπορούν να αναπτυχθούν τοπικά ή να αποθηκευτούν σε κοινόχρηστα αποθετήρια, όπως το Docker Hub. Αυτό επιτρέπει την εύκολη ανταλλαγή και την επαναχρησιμοποίηση εικόνων.



Εικόνα 15: Basic docker functionality

Εισαγωγή στο Docker Compose:

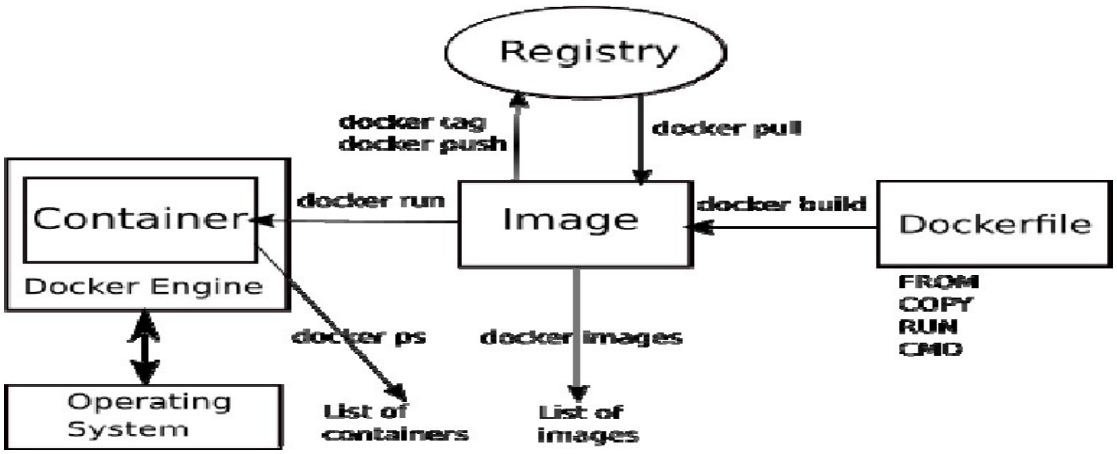
Το Docker Compose είναι ένα εργαλείο που επιτρέπει τη διαχείριση πολλαπλών εφαρμογών με χρήση ενός αρχείου διαμόρφωσης YAML. Με το Docker Compose, μπορείτε να ορίσετε τις ρυθμίσεις του δικτύου, τους όγκους δεδομένων και άλλες παραμέτρους για τις εφαρμογές σας.



Εικόνα 16: Basic docker compose functionality

Χρήση του Docker Compose:

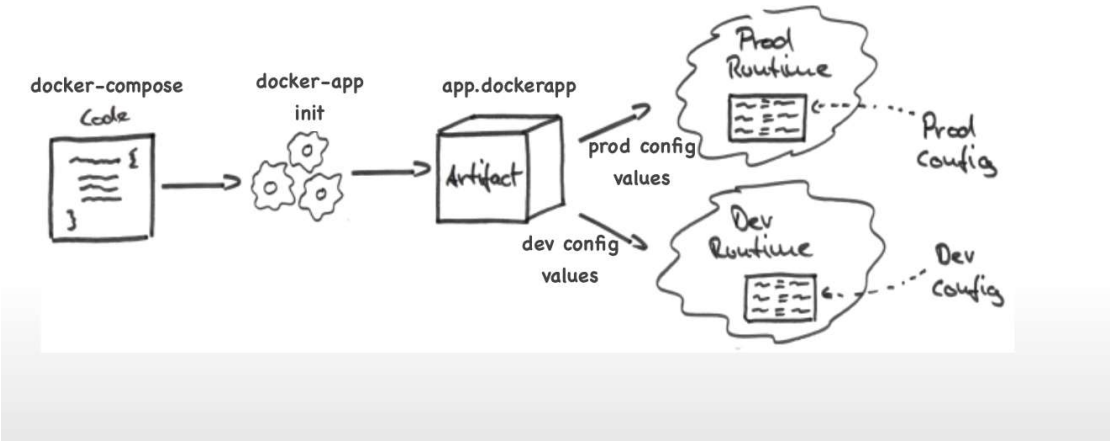
Η χρήση του Docker Compose απλοποιεί τη διαχείριση πολλαπλών εφαρμογών, καθώς μπορείτε να ξεκινήσετε, να σταματήσετε και να διαμορφώσετε ολόκληρο το περιβάλλον ανάπτυξής σας με ένα απλό αρχείο διαμόρφωσης.



Εικόνα 17: Docker containers and images, docker registry

Διαχείριση Πολυεπίπεδων Εφαρμογών:

Ένα από τα βασικά πλεονεκτήματα του Docker Compose είναι η δυνατότητα διαχείρισης πολυεπίπεδων εφαρμογών. Μπορείτε να ορίσετε διάφορα τμήματα της εφαρμογής σας (όπως οι υπηρεσίες βάσεων δεδομένων, οι εξυπηρετητές εφαρμογών και οι διακομιστές φόρτωσης) σε ένα αρχείο docker-compose.yml.



Εικόνα 18: Docker compose for managing multi tier apps

Επεκτασιμότητα του Περιβάλλοντος Ανάπτυξης:

Ένα από τα κύρια χαρακτηριστικά του Docker Compose είναι η επεκτασιμότητά του. Μπορείτε να προσθέσετε νέες υπηρεσίες και να προσαρμόσετε το περιβάλλον ανάπτυξής σας με ελάχιστη προσπάθεια, επιτρέποντάς σας να ανταπεξέλθετε στις ανάγκες της εφαρμογής σας χωρίς να χρειάζεται να αναδιαμορφώσετε ολόκληρο το περιβάλλον.

Αυτοματοποίηση Διαδικασιών:

Οι δυνατότητες αυτοματοποίησης του Docker Compose επιτρέπουν τη δημιουργία αυτόματων διαδικασιών για τη διαχείριση και την ανάπτυξη των εφαρμογών σας. Μπορείτε να ορίσετε σενάρια εκτέλεσης που ανταποκρίνονται στις ανάγκες του κάθε περιβάλλοντος ανάπτυξης σας, εξοικονομώντας έτσι χρόνο και πόρους.

Συνεργασία και Ανάπτυξη Ομάδας:

Ο Docker Compose διευκολύνει τη συνεργασία και την ανάπτυξη ομάδας, καθώς επιτρέπει στους προγραμματιστές να ορίζουν και να μοιράζονται το περιβάλλον ανάπτυξής τους με άλλα μέλη της ομάδας. Αυτό διευκολύνει την ανταλλαγή γνώσεων και επιτρέπει την ομαδική ανάπτυξη εφαρμογών με αποτελεσματικό τρόπο.

ΑΠΑΙΤΗΣΗ (Business Requirement)

Η απαίτηση (Business requirement) σε ένα καθεστώς επιχείρησης αποκρυσταλλώνεται στο έγγραφο BRD (Business Requirement Document). Μέσα στο document αυτό γίνεται η πλήρης εντύπωση της εμπορικής απαίτησης, των παραδοτέων, των διάφορων κριτήριων αποδοχής καθώς και των KPIs με βάση τα οποία αυτά θα βαθμολογηθούν. Είναι η βάση εκκίνησης κάθε industrial-strength έργου πληροφορικής και το βασικό input για της ομάδες των solution architects και των business analysts που πρέπει να μεταγλωττίσουν την εμπορική απαίτηση σε τεχνική προδιαγραφή.

Για το συγκεκριμένο έργο ο σκοπός ήταν να επιτύχουμε:

1. **Δημιουργία ενός μοντέρνου framework για ένα health-oriented site**
    a. Επιλογή n-tier αρχιτεκτονικής
    b. Επιλογή sqlite για portability
    c. Επιλογή jwt για serverless auth
    d. Επιλογή docker / docker-compose για CI/CD
2. **Δημιουργία λειτουργικότητας για υποστήριξη Upload/download ιατρικών εγγράφων**
    a. Επιλογή api endpoint (upload/download) για το integration με το front-end
    b. Επιλογή αποθήκευσης των εγγράφων στον server
    c. Επιλογή διατήρησης της συσχέτισης χρήστη/uploaded εγγραφών σε πίνακα στη βάση.
3. **Δημιουργία λειτουργικότητας για την καταγραφή ιατρικών επισκέψεων σε νοσοκομεία**
    a. Δημιουργία του πίνακα hospitals / visitations που κρατάνε την συσχέτιση επισκέψεων του ασθενή σε αντίστοιχα ιδρύματα.
    b. Δημιουργία relational σχέσης με τον πίνακα των users
    c. Παροχή δυνατότητας στον χρήστη για insert/update καθώς και αποθήκευση του κόστους της κάθε επίσκεψης.
4. **Δημιουργία λειτουργικότητας για την καταγραφή ιατρικών επισκέψεων σε κλινικές**
    a. Δημιουργία του πίνακα clinics / visitations που κρατάνε την συσχέτιση επισκέψεων του ασθενή σε αντίστοιχα ιδρύματα.
    b. Δημιουργία relational σχέσης με τον πίνακα των users
    c. Παροχή δυνατότητας στον χρήστη για insert/update καθώς και αποθήκευση του κόστους της κάθε επίσκεψης.
5. **Δημιουργία λειτουργικότητας για την καταγραφή ιατρικών επισκέψεων σε προσωπικούς ιατρούς**
    a. Δημιουργία του πίνακα doctors / visitations που κρατάνε την συσχέτιση επισκέψεων του ασθενή σε αντίστοιχα ιδρύματα.
    b. Δημιουργία relational σχέσης με τον πίνακα των users
    c. Παροχή δυνατότητας στον χρήστη για insert/update καθώς και αποθήκευση του κόστους της κάθε επίσκεψης.
6. **Δημιουργία λειτουργικότητας για την παρακολούθηση συνταγών από τον χρήστη**
7. **Δημιουργία λειτουργικότητας για αυτόματες ειδοποιήσεις του χρήστη σε θέματα υγείας.**
8. **Δημιουργία λειτουργικότητας για συσχετισμένα τηλεφωνά/email που θα παραλαμβάνουνε τις ειδοποιήσεις του χρήστη.**
9. **Δημιουργία λειτουργικότητας για υπολογισμό / πρόταση δίαιτας με βάση μια θερμιδική αξία και ένα επίπεδο άσκησης.**
10. **Δημιουργία λειτουργικότητας για υπολογισμό δείκτη μάζας σώματος δεδομένων παραμέτρων του χρήστη.**
11. **Προσωποποιημένο account management και συναφείς λειτουργικότητες.**

Επίσης, η εφαρμογή μας είχε της παρακάτω τεχνικές απαιτήσεις που έχουν να κάνουν με τον τρόπο που είναι στημένη η εφαρμογή, τον τρόπο που γίνεται deploy καθώς και monitor. Συγκεκριμένα:

1. Υλοποίηση λύσης σε N-tier αρχιτεκτονική με ξεχωριστά backend / frontend / logic layer.
2. Υλοποίηση λύσης με διεπαφη με SaaS υπηρεσία αποστολής ειδοποιήσεων (sms/email) – Sendgrid
3. Υλοποίηση λύσης με container-driven pipeline (docker + docker compose)
4. Υλοποίηση λύσης με docker-compose και 1 docker / component so that components are individually deployable.
5. Η εφαρμογή θα πρέπει να υποστηρίζει διαχείριση χρηστών, auditing, logging και να έχει υλοποιηθεί με μοντέρνο τρόπο authentication (jwt token)

ΑΝΑΛΥΣΗ (Business Analysis)

Κατά την ανάλυση της λύσης κληθήκαμε να πάρουμε κάποιες αποφάσεις σχετικά με το πως θα στηθεί το όλο προϊόν και όσον αφορά τις υποδομές, και όσον αφορά την υλοποίηση άλλα και όσον αφορά το deploy pipeline.

Όσον αφορά τις υποδομές αποφασίσαμε να μην στηθεί μια λύση με γνώμονα κάποιο συγκεκριμένο λειτουργικό, αλλά να κινηθούμε agnostic, έτσι επιλέχθηκε η λύση του docker και αντίστοιχα η λύση μας έχει δοκιμαστεί σε docker desktop (UX/Windows).

Όσον αφορά την δυνατότητα να υπάρχει ανεξάρτητο deploy των components, καθώς και η δυνατότητα να μπορεί να γίνει ανεξάρτητα debug κάθε component αποφασίστηκε να στηθούν όλα τα διαφορετικά tiers της εφαρμογής σε ξεχωριστά docker files, τα οποία θα συντονίζονται από docker-compose.

Όσον αφορά την δυνατότητα για scalabiity-in-cloud υπάρχει η δυνατότητα να ενταχθεί το docker-compose yaml σε Kubernetes cluster που να έχει δικό του scaling plan ανάλογα με το προφίλ της ζήτησης (horizontal scaling/ vertical scaling).

Όσον αφορά την δυνατότητα για high-availability-in-cloud υπάρχει η δυνατότητα να ενταχθεί το docker-compose σε Kubernetes-cluster με high-availability schema, η εναλλακτικά να γίνουν deploy τα dockerfiles σε 2 η περισσότερα dedicated VM που να βρίσκονται κάτω από έναν Load balancer (either layer 4 ie Azure Load Balancer for IP based load balancing or Application Gateway for a layer 7 balancing)

Όσον αφορά το presentation layer αποφασίστηκε η χρήση της Vue / Vite για την διαχείριση του front-end, η οποία έχει στηθεί κατά την MVC αρχιτεκτονική με ξεχωριστά routes για κάθε λειτουργικότητα, περισσότερες πληροφορίες μπορούμε να δούμε στο κεφάλαιο του devops.

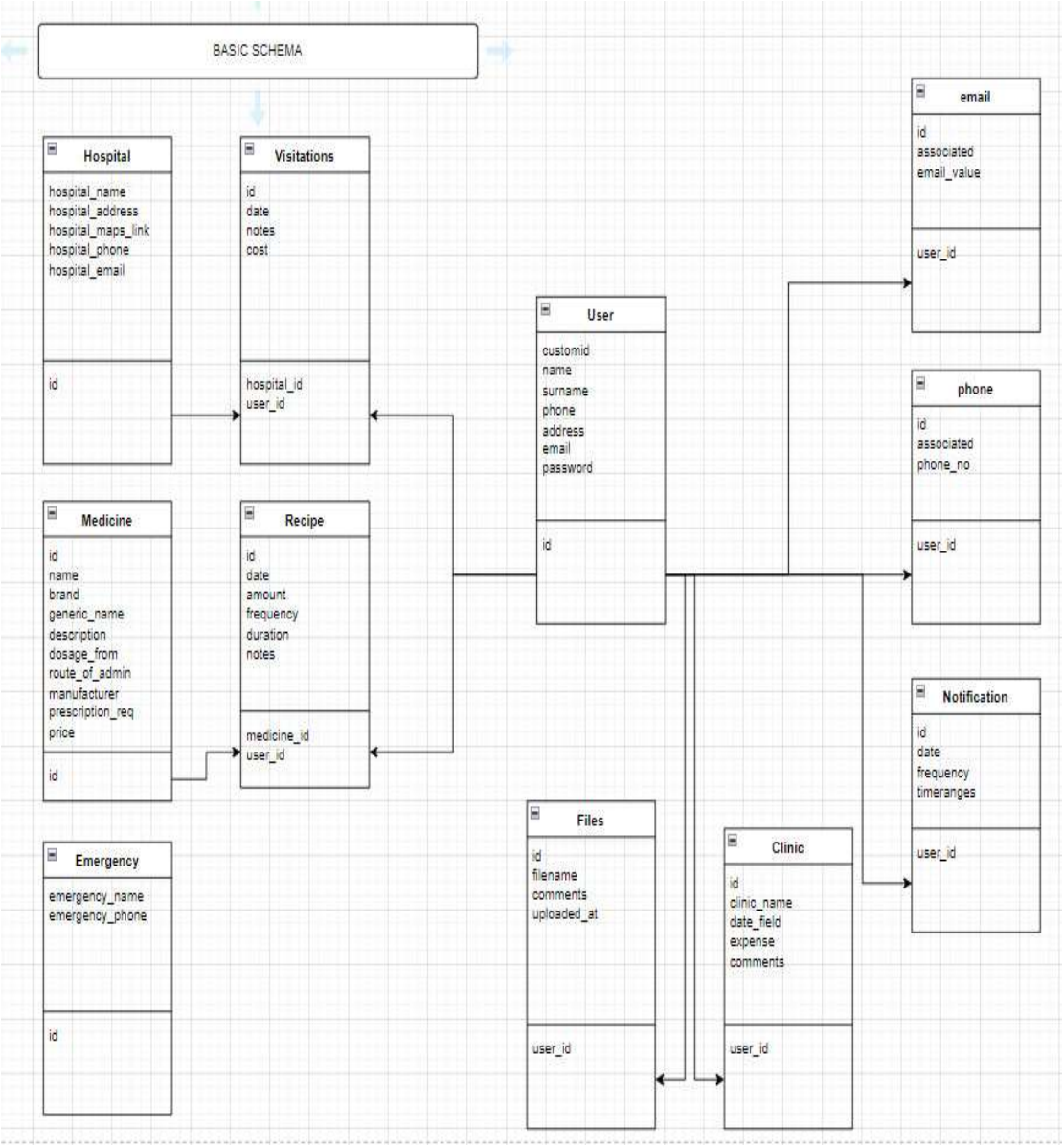
Όσον αφορά το Logic / Persistence layer αποφασίστηκε η χρήση της Python / Flask έτσι ώστε να έχουμε ένα extensible framework που να μπορεί να εξυπηρετήσει όλες τις δυνητικές ανάγκες που μπορεί να δημιουργηθούν. Η αρχιτεκτονική είναι και εδώ MVC και στο σύνολο N-tier of MVCs.

Όσον αφορά το data layer αποφασίστηκε η χρήση της sqlite έναντι κάποια server-based βάσης δεδομένων καθώς θέλουμε η εφαρμογή να είναι extensible for mobile data, ειδικά σε περιπτώσεις που αφορά πίνακες οι οποίοι αφορούν την ίδια την εφαρμογή και όχι κάποια δεδομένα χρήστη.

Όσον αφορά το σχήμα της βάσης, επιλέχθηκε ως άξονας ο πίνακας των users, και χτίστηκε κανονικοποιημενα η βάση τριγύρω από αυτόν με γνώμονα να μην υπάρχουν διπλοεγγραφες δεδομένων.

ΣΧΕΔΙΑΣΗ (Architectural Design)

Database schema



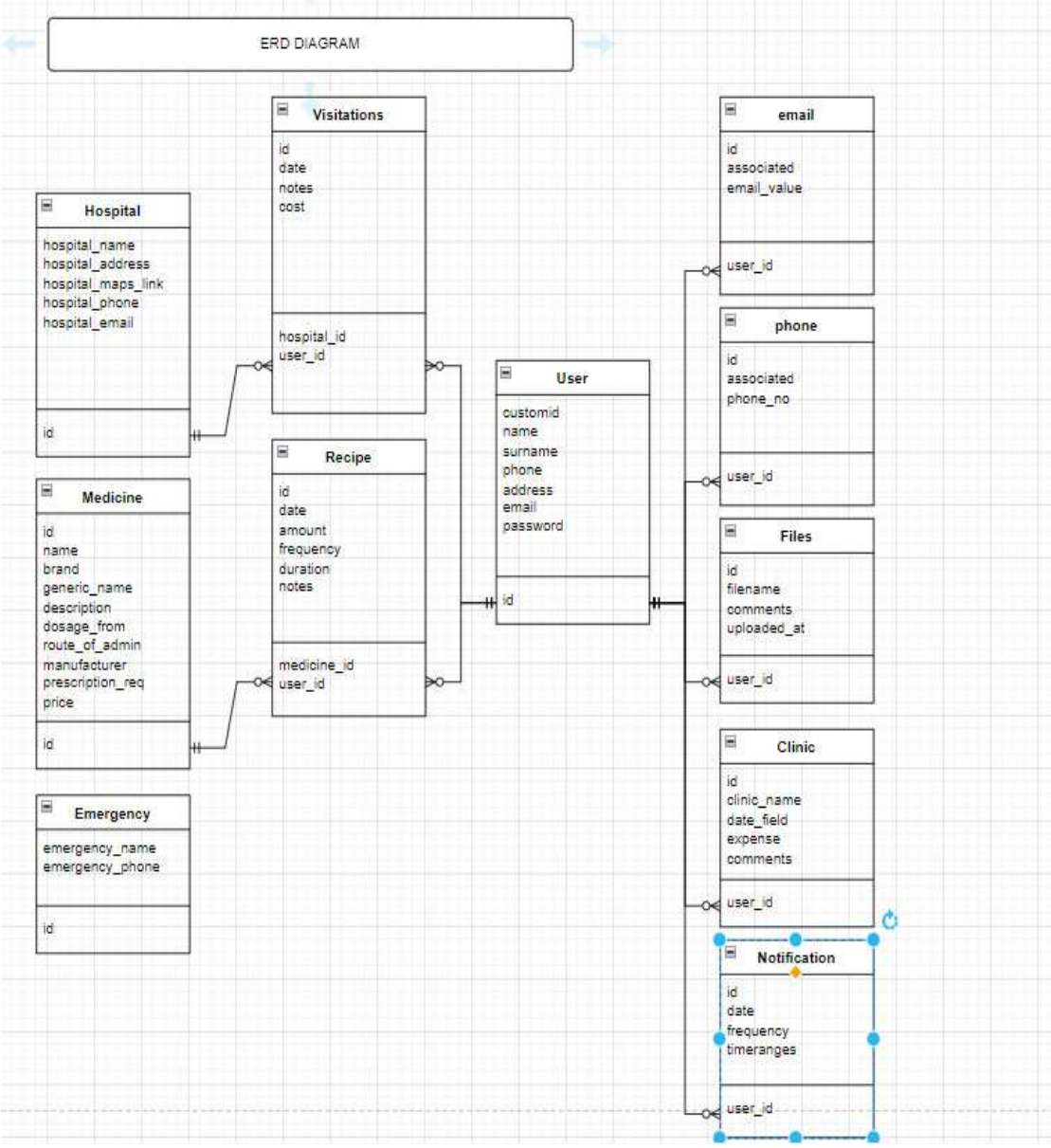
Εικόνα 19: Basic database schema

Στο βασικό schema της βάσης έχουμε τον πίνακα των users σαν κεντρικό άξονα οργάνωσης ο οποίος διασυνδέεται με σχεσιακές σχέσεις με άλλες οντότητες που χρησιμοποιούμε στην εφαρμογή μας. Στην πλειοψηφία των περιπτώσεων η διασύνδεση γίνεται με foreign key το user_id αλλά έχουμε και πίνακες πολλαπλών συσχετίσεων που γίνονται join με πολλαπλά Keys.

Επιλέχθηκε η σχεδίαση σε σχεσιακή βάση δεδομένων καθώς χρειαζόμαστε γρήγορη πρόσβαση στα δεδομένα, και υψηλό performance στα συγκεκριμένα Joins. Οι σχεσιακές βάσεις από σχεδιασμού τους έχουν πολύ υψηλή απόδοση στην συσχέτιση δεδομένων και με τα αντίστοιχα indexing σε tables μπορούν να πιάσουν πολύ καλές αποδόσεις.

Επίσης, επιβάλλουνε ένα σύνολο κανόνων οι οποίοι εξασφαλίζουν την καλή υγεία των δεδομένων μας καθώς και αντίστοιχα μας προστατεύουνε από διπλοκαταχωρησεις δεδομένων, κακή χρήση των data types και λάθη του data entry.

Database ERD diagram



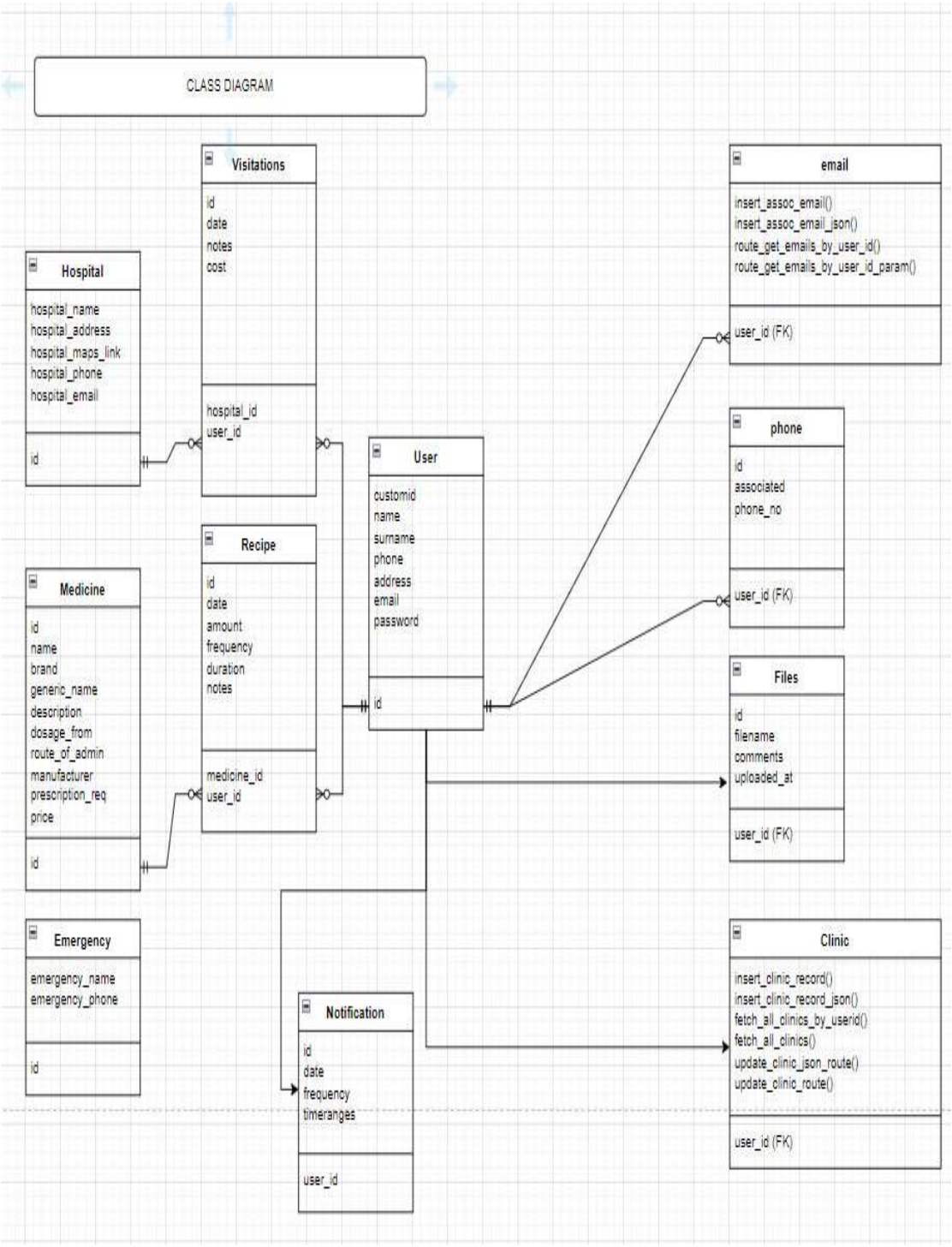
Εικόνα 20: Database Entity-Relation Diagram

Βλέπουμε λοιπόν στο διάγραμμα ERD ότι έχουμε αναπαραστήσει ένα σύνολο σχέσεων όπως αυτές περιγράφονται παρακάτω:

1. Users -> Notifications (One-to-Many) – Join on FK(user_id)
2. Users -> Clinics (One-to-Many) – Join on FK(user_id)
3. Users -> Files (One-to-Many) – Join on FK(user_id)
4. Users -> Phones (One-to-Many) – Join on FK(user_id)

5. Users -> Email (One-to-Many) – Join on FK(user_id)
6. Users -> Visitations (One-to-Many) – Join on FK(user_id)
7. Hospital -> Visitations (One-to-Many) – Join on FK(hospital_id)
8. Users -> Hospitals (Many-to-Many) – Join on FK(hospital_id, user_id)
9. Users -> Recipe (One-to-Many) – Join on FK (user_id)
10. Medicine -> Recipe (One-to-Many) – Join on FK (medicine_id)
11. Recipe -> Users (Many-to-Many) – Join on FK (medicine_id, me

Database Class Diagram



Εικόνα 21: Database class diagram

Στο παραπάνω διάγραμμα βλέπουμε ενδεικτικά κάποιες από τις μεθόδους που γίνονται expose από κάθε entity της βάσης μας στο επίπεδο του κωδικα-μοντελου πια. Οπού έχουν παραληφθεί είναι γιατί είναι πολλά για να χωρέσουν στο σχήμα. Ενδεικτικά αναφέρουμε για το entity Email:

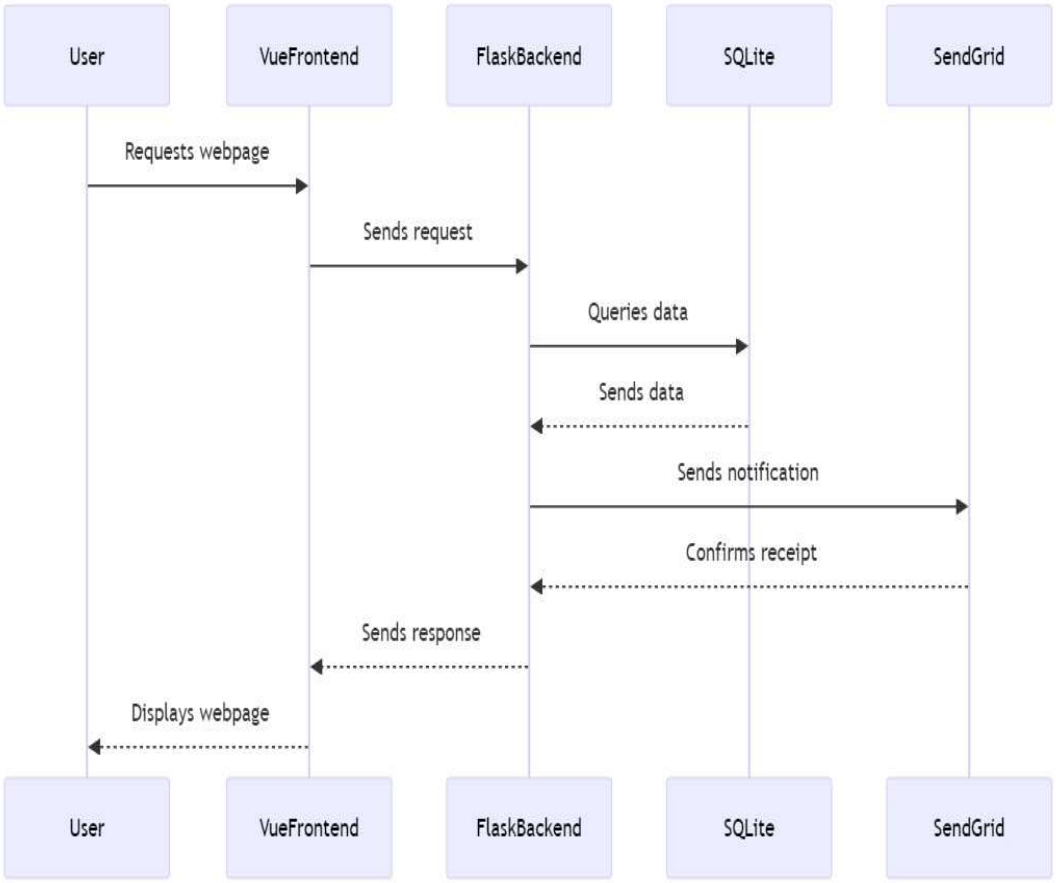
1. Insert_assoc_email()

2. Insert_assoc_email_json()
3. Get_emails_by_userid()
4. Get_emails_by_userid_param()

Για το entity Clinic:

1. Insert_clinic_record()
2. Insert_clinic_record_json()
3. Fetch_all_clinics_by_userid()
4. Fetch_all_clinics()
5. update_clinic_json_route()
6. update_clinic_route()

Data Flow Sequence Diagram



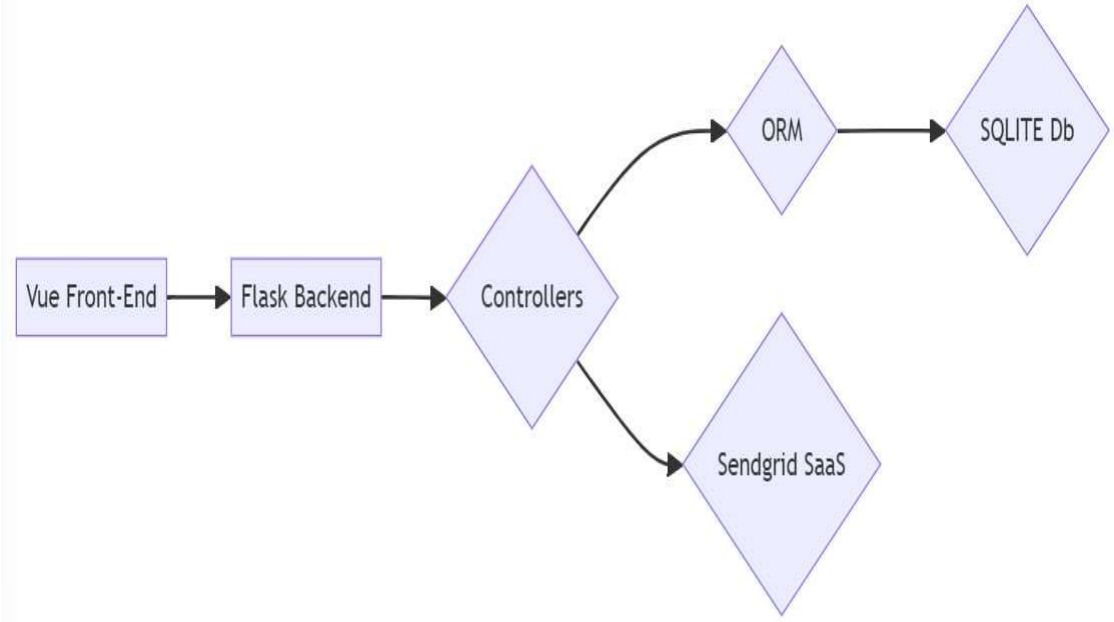
Εικόνα 22: Data Sequence Diagram

Στο παραπάνω sequence diagram βλέπουμε ένα ενδεικτικό flow της εφαρμογής μας με περιγραφή όλων των αντίστοιχων διεπαφών:

1. Αρχικά, ένας χρήστης της εφαρμογής αιτείται πρόσβαση στη σελίδα της εφαρμογής

2. Το αίτημα το παραλαμβάνει το Vue Front-end το οποίο την προωθεί στον αντίστοιχο front-end controller.
3. Ο front-end controller προωθεί το αίτημα στο επόμενο tier που είναι το FlaskBackend.
4. Το FlaskBackend παραλαμβάνει το αίτημα και επιλεγεί με βάση τους controllers του ποιο controller θα ενεργοποιήσει, στην συνέχεια προωθώντας το αίτημα στο datalayer.
5. Το datalayer θα επιλέξει το κατάλληλο database query για να φέρει την σωστή πληροφορία και στην κατάλληλη μορφή ετσι ώστε να μπορεί να την χρησιμοποιήσει το business layer.
6. Σε περίπτωση που χρειάζονται notifications τότε από το business layer φεύγει αίτημα προς την SaaS platform του Sendgrid με καταλληλά φορμαρισμένο mail template και στέλνει το email.
7. Σε συνέχεια η απάντηση προωθείται από το data layer προς το FlaskBackend και στην συνέχεια προς το VueFrontEnd που είναι υπεύθυνο για το πως θα κάνει render την πληροφορία στο front-end.

Solution Base Architecture



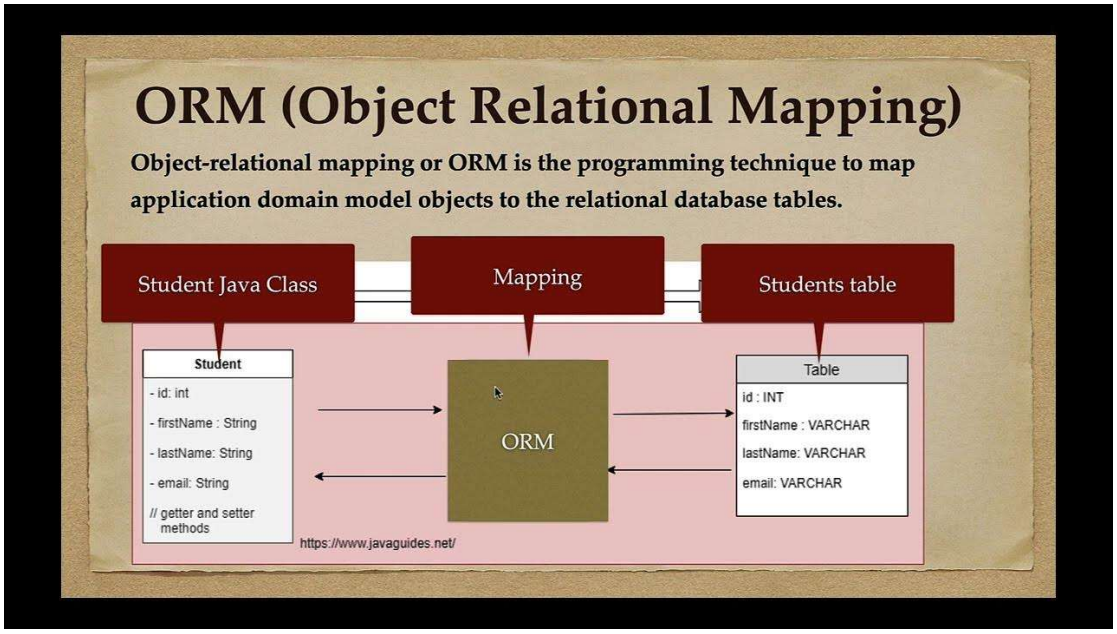
Εικόνα 23: Solution Base Backend Architecture Diagram

Στο παραπάνω σχήμα βλέπουμε ένα συνοπτικό high-level διάγραμμα της εφαρμογής μας. Ακολουθεί επεξήγηση για κάθε ένα component παρακάτω:

1. Vue Front-End
    a. Controllers: Responsible for parsing the information retrieved from the data layer and presenting them in the right front-end template to be rendered.
2. Flask Back-End
    a. Controllers: Responsible for accepting a new request and passing it to the right segment of the datalayer for it to be processed. When the processing is done the response is returned to the front-end. The communication happens via API call initiated from the front-end.
    b. SaaS Services
    c. ORM
3. Database (Sqlite)
    a. We chose the usage of sqlite for simplicity as well as ease-of-deployment reasons, it is also easier to port to mobile and also create a demo because no database servers need to be provisioned.

Object-relational mappers

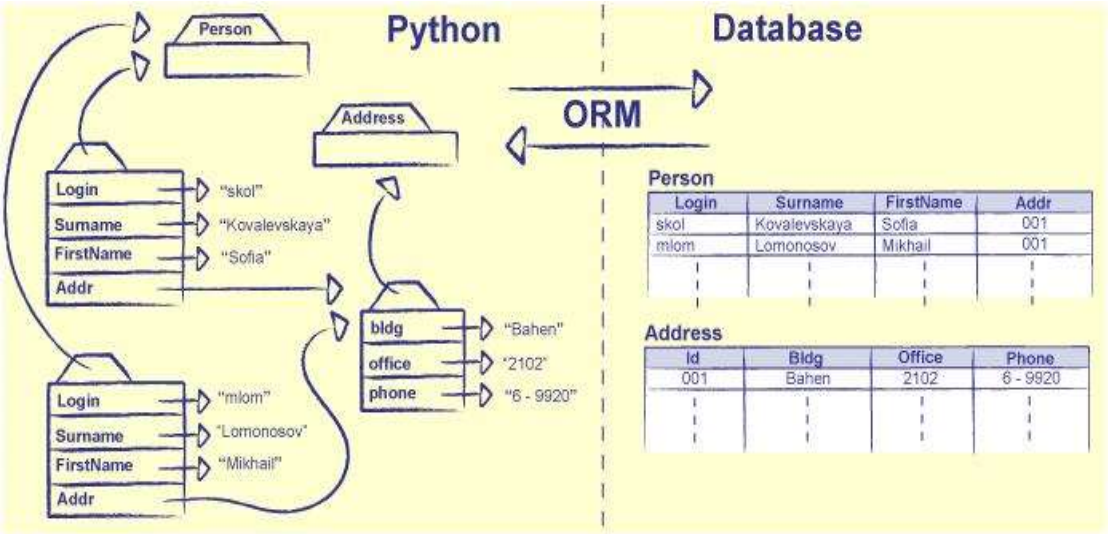
Οι Object Relational Mappers (ORMs) γεφυρώνουν το χάσμα μεταξύ γλωσσών προγραμματισμού αντικειμενοστραφών όπως Python, Java ή C# και σχεσιακών βάσεων δεδομένων όπως MySQL, PostgreSQL ή SQLite. Βασικά παρέχουν ένα επίπεδο αφαίρεσης, επιτρέποντας στους προγραμματιστές να αλληλεπιδρούν με τις βάσεις δεδομένων χρησιμοποιώντας αντικείμενα αντί για SQL ερωτήματα απευθείας.



Εικόνα 24: Object relational mapping schematic explanation

Τα ORMs λειτουργούν με το να αντιστοιχίζουν πίνακες βάσεων δεδομένων σε κλάσεις και γραμμές πινάκων σε αντικείμενα, διευκολύνοντας την ομαλή επικοινωνία μεταξύ της εφαρμογής και της βάσης δεδομένων. Αυτό σημαίνει ότι οι προγραμματιστές μπορούν να διαχειρίζονται δεδομένα χρησιμοποιώντας γνωστά αντικειμενοστραφή παραδείγματα όπως η κληρονομιά, η ενθάρρυνση, και ο πολυμορφισμός, αντί να ασχολούνται με την πολύπλοκη σύνταξη SQL.

Δημοφιλή πλαίσια ORMs περιλαμβάνουν το SQLAlchemy για την Python, το Hibernate για την Java, το Entity Framework για το .NET, και το Django ORM για το πλαίσιο Django. Κάθε ένα από αυτά τα πλαίσια προσφέρει το δικό του σύνολο χαρακτηριστικών και πλεονεκτήματα, προσαρμοσμένα σε διαφορετικές γλώσσες προγραμματισμού και προτιμήσεις του προγραμματιστή.

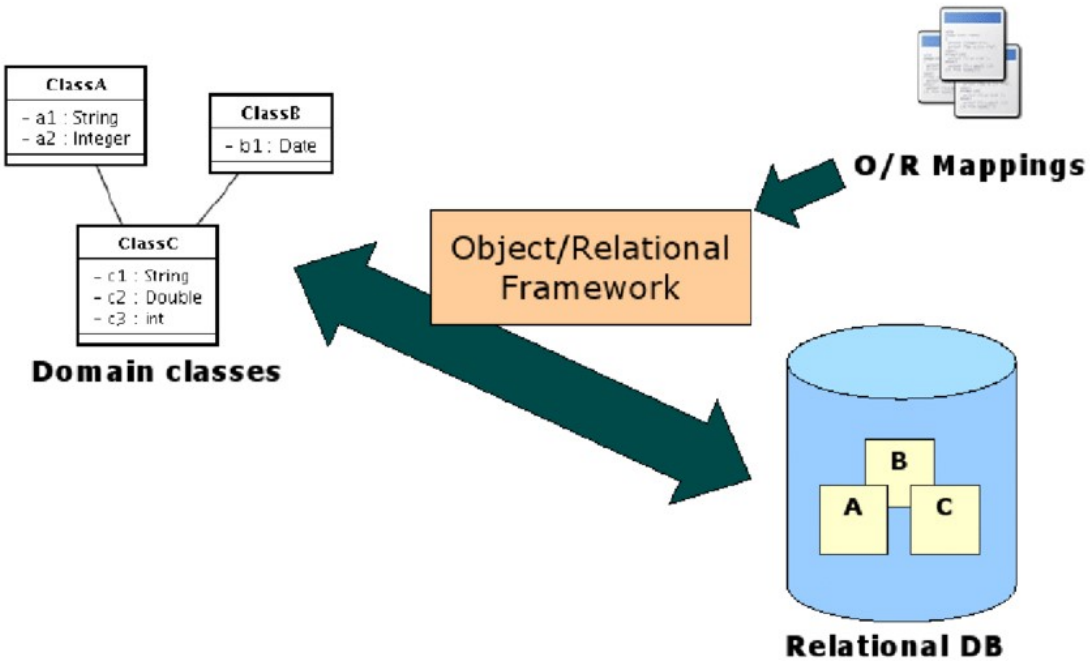


Εικόνα 25: Interfacing with the actual database and models

Ένα από τα κύρια οφέλη της χρήσης του ORM είναι η αυξημένη παραγωγικότητα και η μείωση του χρόνου ανάπτυξης. Οι προγραμματιστές μπορούν να επικεντρωθούν περισσότερο στην επιχειρηματική λογική και λιγότερο στην επανάληψη της εγγραφής SQL ερωτημάτων, με αποτέλεσμα πιο καθαρό, πιο συντηρήσιμο κώδικα.

Τα ORMs παρέχουν επίσης ένα επίπεδο ανεξαρτησίας βάσης δεδομένων, επιτρέποντας στους προγραμματιστές να μεταβαίνουν μεταξύ διαφορετικών συστημάτων βάσεων δεδομένων χωρίς να χρειάζεται να επαναγράφουν μεγάλα τμήματα του κώδικά τους. Αυτή η ευελιξία μπορεί να είναι ιδιαίτερα χρήσιμη σε περιπτώσεις όπου οι απαιτήσεις κλιμακωσιμότητας ή απόδοσης αλλάζουν με τον καιρό.

Ένα άλλο πλεονέκτημα του ORM είναι η βελτίωση της ασφάλειας. Τα πλαίσια ORM συχνά περιλαμβάνουν ενσωματωμένα μηχανισμούς για την πρόληψη επιθέσεων εισβολής SQL και άλλων κοινών ευπάθειών ασφάλειας, μειώνοντας τον κίνδυνο διαρροής δεδομένων και μη εξουσιοδοτημένης πρόσβασης.



Εικόνα 26: Object relational abstraction of actual database

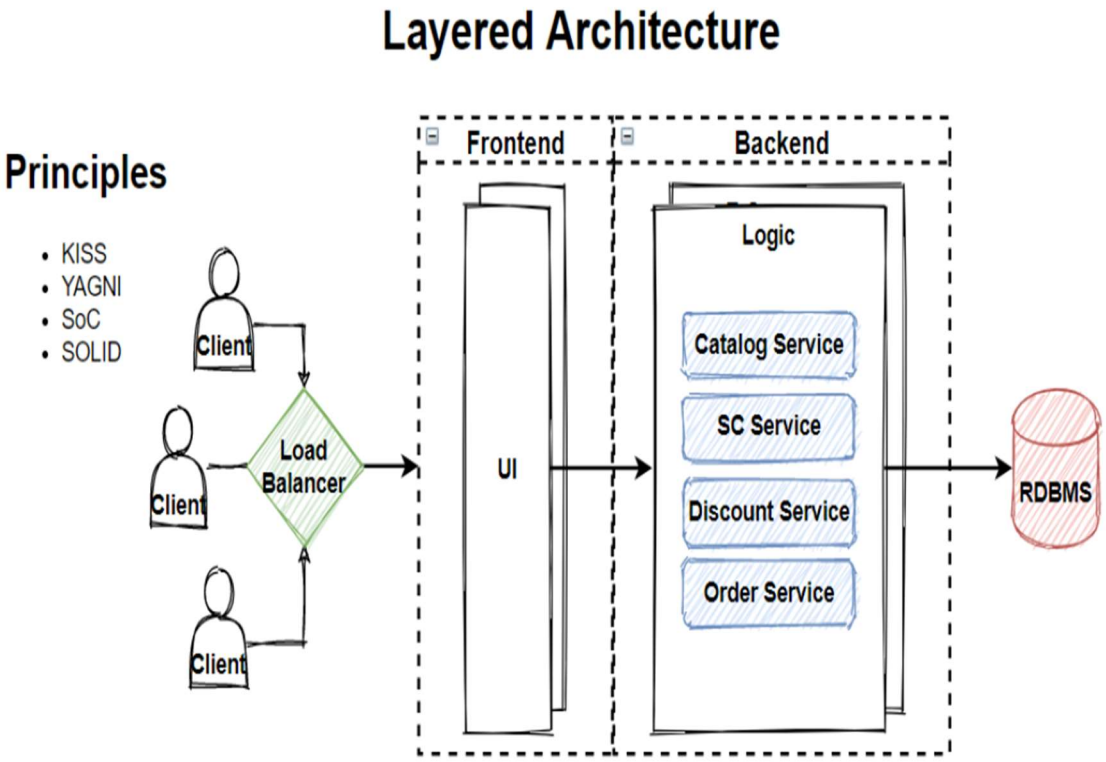
Ωστόσο, τα ORMs δεν είναι απαλλαγμένα από μειονεκτήματα. Ένα κοινό πρόβλημα είναι ο επιπλέον φόρτος απόδοσης. Τα SQL ερωτήματα που δημιουργούνται από το ORM μπορεί να μην είναι πάντα τόσο αποδοτικά όσο τα χειροποίητα SQL, οδηγώντας σε πιθανά σημεία φραγής απόδοσης, ειδικά σε εφαρμογές υψηλής κυκλοφορίας.

Επιπλέον, τα ORMs μπορεί μερικές φορές να αφαιρούν υπερβολικά πολλή από την υποκείμενη λειτουργικότητα της βάσης δεδομένων, κάνοντας δύσκολο τον βελτιστοποίηση των ερωτημάτων ή την αξιοποίηση προηγμένων χαρακτηριστικών της βάσης δεδομένων. Οι προγραμματιστές μπορεί να βρεθούν περιορισμένοι από τις δυνατότητες του πλαισίου ORM που χρησιμοποιούν.

Ένα άλλο μειονέκτημα του ORM είναι η καμπύλη μάθησης που συνδέεται με την κατάκτηση ενός νέου πλαισίου. Ενώ τα ORMs μπορούν να απλοποιήσουν τις διαδράσεις με τη βάση δεδομένων, εισάγουν επιπλέον πολυπλοκότητα, ειδικά για προγραμματιστές που δεν είναι εξοικειωμένοι με τα έννοια αντικειμενοστραφούς αντιστοίχισης.

Συνοψίζοντας, ενώ τα ORMs προσφέρουν πολλά οφέλη όπως η αύξηση της παραγωγικότητας, η ανεξαρτησία της βάσης δεδομένων και η βελτίωση της ασφάλειας, έρχονται και με το δικό τους σύνολο προκλήσεων, συμπεριλαμβανομένου του επιπλέον φόρτου απόδοσης και της καμπύλης μάθησης. Τελικά, η απόφαση να χρησιμοποιηθεί ένα ORM εξαρτάται από παράγοντες όπως οι απαιτήσεις του έργου, η εμπειρογνωμοσύνη του προγραμματιστή και οι παράγοντες απόδοσης.

ΣΧΕΔΙΑΣΗ (Code Design)



Εικόνα 27: Layered architecture principles

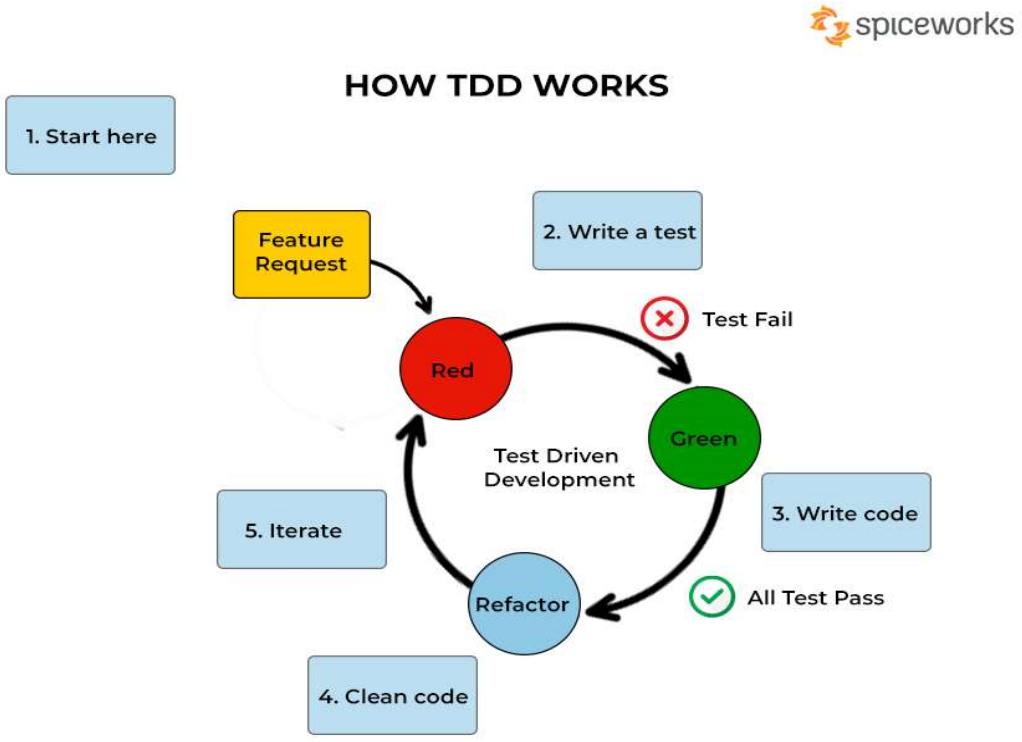
1. Modular architecture
    a. Η modular αρχιτεκτονικη επιτρέπει τη διαίρεση του κώδικα σε μικρότερα, ανεξάρτητα τμήματα που ονομάζονται modules. Αυτό διευκολύνει τη συντήρηση και την επαναχρησιμοποίηση κώδικα, καθώς κάθε μονάδα είναι ανεξάρτητη και εστιάζει σε μια συγκεκριμένη λειτουργικότητα.
2. Dependency injection
    a. Το Dependency injection είναι μια τεχνική σχεδιασμού που χρησιμοποιείται για τη μείωση της συνδεσιμότητας μεταξύ διαφορετικών συστατικών του συστήματος. Με τη χρήση της ενσωμάτωσης εξαρτήσεων, οι κλάσεις ή οι συναρτήσεις δεν είναι συνδεδεμένες άμεσα με τις υπηρεσίες ή τα αντικείμενα που χρησιμοποιούν, αλλά περνούνται ως παράμετροι.
3. Separate logic tier
    a. Η διάχυση της λογικής σε ξεχωριστά επίπεδα βελτιώνει τη διαχείριση του κώδικα και την αναγνωσιμότητα. Κάθε επίπεδο εστιάζει σε μια συγκεκριμένη λειτουργικότητα, όπως η προσπέλαση δεδομένων, η επιχειρηματική λογική και η παρουσίαση.
4. Separate logic tier for SaaS services
    a. Για τις υπηρεσίες λογισμικού ως υπηρεσία (SaaS), είναι κρίσιμο να έχουμε ξεχωριστά επίπεδα λογικής για τη διαχείριση των χρηστών, των συνδρομητών και της ασφάλειας. Αυτό επιτρέπει την εύκολη κλιμάκωση και την απομακρυσμένη διαχείριση των υπηρεσιών, ενώ εξασφαλίζει την αποτελεσματική απόδοση και ασφάλεια των δεδομένων των πελατών.
5. Logging + Error control
    a. Η καταγραφή συμβάντων και ο έλεγχος σφαλμάτων είναι κρίσιμοι για την ασφάλεια και τη σταθερότητα του συστήματος. Η καταγραφή συμβάντων

καταγράφει τις ενέργειες των χρηστών και τις αλλαγές στα δεδομένα, προσφέροντας αναλυτική πληροφορία για πιθανά προβλήματα ή ασυνήθιστες ενέργειες. Ο έλεγχος σφαλμάτων επιτρέπει την ανίχνευση, την καταγραφή και την αντιμετώπιση σφαλμάτων και εξαιρέσεων, βελτιώνοντας έτσι την αποδοτικότητα της ανάπτυξης και τη συνολική αξιοπιστία του συστήματος.

6. Enumerators
    a. Οι enumerators είναι λογικές συσχετίσεις για default τιμές που επαναχρησιμοποιούνται συχνά στο επίπεδο του κώδικα, χρησιμοποιούνται για να υπάρχει ένα ενιαίο reference σε αυτές τις συλλογες από σταθερες από όλα τα components του κωδικα. Κλασσικες περιπτωσεις χρησης enumerators είναι για http status codes, http headers, model types etc.
7. Test cases
    a. Για κάθε λειτουργικοτητα πρεπει να υπαρχει και το αντιστοιχο test, η λυση εχει χτιστει με βαση το TDD (Test Driven Development).

TDD (Test-Driven Development)

Η Ανάπτυξη Με Τεστ (Test-Driven Development - TDD) είναι μια πρακτική ανάπτυξης λογισμικού που θέτει τα τεστ στο επίκεντρο της διαδικασίας ανάπτυξης. Κατά την TDD, οι προγραμματιστές γράφουν τα τεστ πριν την υλοποίηση του κώδικα, καθιστώντας τα τεστ τον οδηγό για τον σχεδιασμό και την υλοποίηση της λειτουργικότητας.



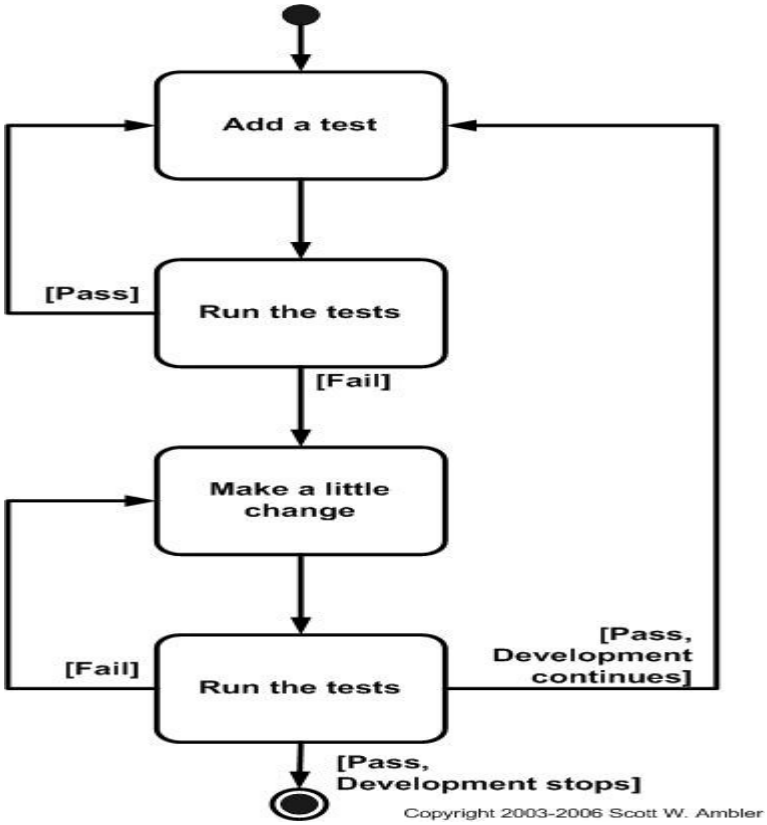
Εικόνα 28: Test driven development

Η διαδικασία ανάπτυξης με τη μέθοδο TDD ακολουθεί συνήθως τρεις κύκλους: Κόκκινο, Πράσινο, Μπλε. Αρχικά, γράφονται τα τεστ που αποτυγχάνουν (κόκκινος κύκλος), αντιπροσωπεύοντας τις αναμενόμενες λειτουργίες του κώδικα. Έπειτα, ο κώδικας υλοποιείται για να κάνει τα τεστ να περάσουν (πράσινος κύκλος). Τέλος, ο κώδικας βελτιώνεται και

αναδιοργανώνεται χωρίς να αλλάζει η λειτουργικότητά του, ενισχύοντας την κατανόηση και τη συντηρησιμότητα του (μπλες κύκλοι).

Μια από τις κύριες αρχές της TDD είναι η συνεχής επανάληψη των κύκλων Κόκκινο, Πράσινο, Μπλε. Αυτός ο κύκλος επιτρέπει στους προγραμματιστές να επικεντρωθούν σε μικρά κομμάτια λειτουργικότητας και να διατηρήσουν τον κώδικα πάντα λειτουργικό.

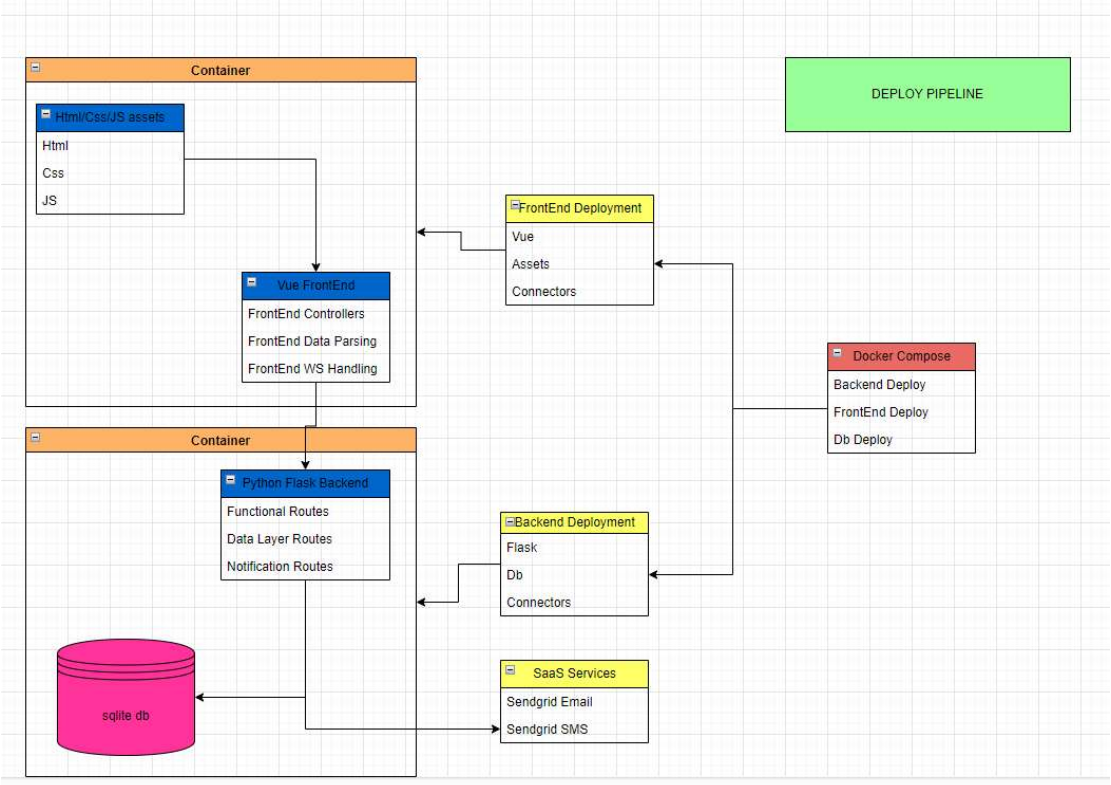
Με τη χρήση της TDD, οι προγραμματιστές μπορούν να επιτύχουν υψηλό επίπεδο κάλυψης κώδικα με τεστ, βελτιώνοντας έτσι την ποιότητα του λογισμικού και μειώνοντας τον κίνδυνο εμφάνισης σφαλμάτων. Ένα άλλο πλεονέκτημα της TDD είναι η αυτοματοποιημένη διαδικασία δοκιμής, η οποία επιτρέπει στους προγραμματιστές να εκτελούν τα τεστ εύκολα και γρήγορα κατά τη διάρκεια της ανάπτυξης και της συντήρησης του κώδικα.



Εικόνα 29: Test driven development flow chart

Ωστόσο, η εφαρμογή της TDD απαιτεί προσοχή και πρακτική από τους προγραμματιστές. Είναι σημαντικό να γράφονται καλά σχεδιασμένα και ορθά τεστ που να καλύπτουν όλες τις πιθανές περιπτώσεις. Επίσης, οι προγραμματιστές πρέπει να είναι ικανοί να διαχειριστούν την πολυπλοκότητα και τις αλλαγές που μπορεί να επιφέρει η στενή σχέση μεταξύ κώδικα και τεστ.

DEPLOY PIPELINE



Εικόνα 30: Schematic of deploy pipeline for solution

Το Docker είναι μια πλατφόρμα ανοιχτού κώδικα που παρέχει μια τεχνολογία λογισμικού που επιτρέπει τη δημιουργία, την ανάπτυξη και την εκτέλεση εφαρμογών μέσα σε περιβάλλοντα που ονομάζονται "containers" ή "images". Αυτό το σύστημα παρέχει έναν τρόπο να τρέξετε εφαρμογές σε απομονωμένα περιβάλλοντα, τα οποία μπορούν να εκτελέσουν τις απαιτούμενες βιβλιοθήκες ή τα dependencies τους χωρίς να επηρεάζουν το σύνολο του συστήματος.

Χρήση του Docker

Το Docker επιτρέπει στους προγραμματιστές και τους architects να δημιουργούν, να τροποποιούν και να μοιράζονται εφαρμογές χρησιμοποιώντας containers. Αυτό το περιβάλλον containerization παρέχει μια σειρά από οφέλη για τους χρήστες.

Υπερ της Χρήσης του Docker

- **Απομόνωση**: Τα containers προσφέρουν απομόνωση για τις εφαρμογές. Κάθε εφαρμογή τρέχει σε ένα διαφορετικό περιβάλλον, χωρίς να επηρεάζει άλλες εφαρμογές ή το σύστημα ως σύνολο.

- **Ελαφρύτητα**: Οι containers είναι images που μπορούν να ξεκινήσουν γρήγορα και να καταλήξουν σε λιγότερη χρήση πόρων συγκριτικά με τους παραδοσιακούς κώδικες ανάπτυξης εφαρμογών σε συγκεκριμένα IDLE
- **Ευελιξία:** Το Docker παρέχει ευελιξία στην ανάπτυξη και την εκτέλεση εφαρμογών, καθώς οι containers είναι φορητοί και εύκολο να μεταφερθούν από ένα περιβάλλον σε ένα άλλο, ανεξάρτητα από το λειτουργικό σύστημα ή την υποδομή.
- **Self-sufficiency**: Οι containers μπορούν να δημιουργηθούν, να ενταχθούν και να καταστραφούν αυτόματα, επιτρέποντας τη διαδικασία του develpoment και το delivery των processes.

Παραδείγματα Χρήσης

Για να κατανοήσουμε καλύτερα τη χρήση του Docker, ας δούμε μερικά παραδείγματα εφαρμογών:

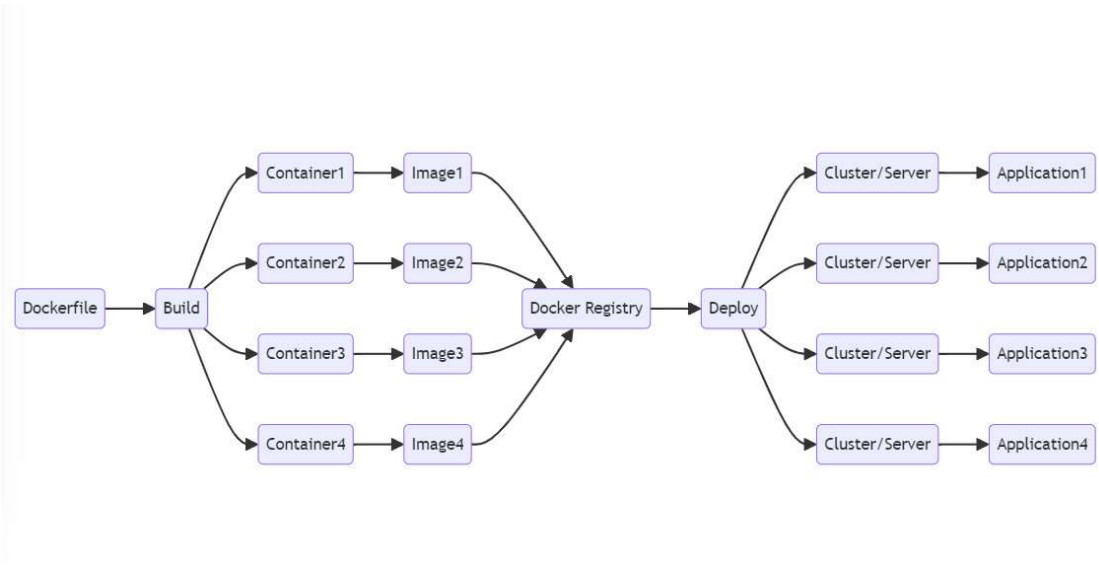
- **Ανάπτυξη Εφαρμογών**: Ένας προγραμματιστής μπορεί να χρησιμοποιήσει το Docker για να δημιουργήσει ένα container που περιλαμβάνει όλα τα εργαλεία και τα dependencies που απαιτούνται για την ανάπτυξη της εφαρμογής του, χωρίς να χρειάζεται να διαμορφώσει το περιβάλλον ανάπτυξης σε κάθε μηχάνημα ξεχωριστά.
- **Δοκιμές Εφαρμογών:** Μια ομάδα δοκιμών μπορεί να χρησιμοποιήσει το Docker για να δημιουργήσει ένα περιβάλλον container που περιλαμβάνει όλα τα στοιχεία που απαιτούνται για την εκτέλεση των δοκιμών, εξαλείφοντας τυχόν προβλήματα σχετικά με τα dependencies του λογισμικού.
- **Παραγωγικό Περιβάλλον:** Ένας διαχειριστής συστήματος μπορεί να χρησιμοποιήσει το Docker για να δημιουργήσει containers που περιλαμβάνουν εφαρμογές ή υπηρεσίες και να τα εκτελέσει στο παραγωγικό περιβάλλον, εξασφαλίζοντας συνέπεια και αξιοπιστία.
- **Κατανεμημένα Συστήματα:** Μια εφαρμογή που απαιτεί κατανεμημένη αρχιτεκτονική μπορεί να εκμεταλλευτεί το Docker για τη δημιουργία διαφορετικών services σε containers, τα οποία μπορούν να επικοινωνήσουν μεταξύ τους μέσω δικτύου.

Συμπεράσματα

Συνολικά, το Docker προσφέρει ένα πανίσχυρο εργαλείο για τη δημιουργία, ανάπτυξη και εκτέλεση εφαρμογών. Με την απομόνωση, την ελαφρότητα και την ευελιξία που παρέχει, το Docker καθιστά την ανάπτυξη λογισμικού πιο αποτελεσματική και τη διαχείριση των εφαρμογών πιο αποτελεσματική.

Με την συνεχή ανάπτυξη και την κοινότητα πίσω από το Docker, αναμένεται να δούμε ακόμα περισσότερες καινοτομίες και βελτιώσεις στο μέλλον, καθιστώντας το Docker ακόμα πιο ανεκτίμητο για την ανάπτυξη λογισμικού και τη διαχείριση των υποδομών.

Solution Docker Compose Schema

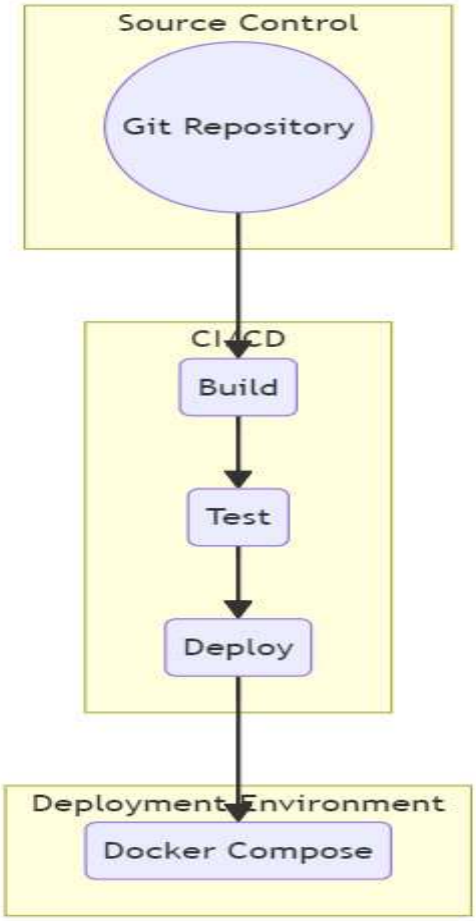


Εικόνα 31: Docker compose schema

Στο παραπάνω σχήμα βλέπουμε ένα τυπικό δείγμα λειτουργίας ενός containerized application με βάση τα docker, docker-compose. Ένα dockerfile μπορεί να περιέχει πολλαπλά containers, τα οποία γίνονται όλα Build on runtime. Το docker είναι os level virtualization οπότε μας επιτρέπει να τρέξουμε πολλούς containers στον ίδιο host και με δυνατότητα να έχουμε πολλά διαφορετικά περιβάλλοντα (windows / linux).

Τα images ανεβαίνουν στο docker registry και κατά το deploy τα κανει pull apo το docker registry, deploy στα αντίστοιχα περιβάλλοντα, ανοίγει τις αντίστοιχες δικτυακές πόρτες και τρέχει τις εφαρμογές. Οι εφαρμογές μπορούν να τρέξουν είτε στον ίδιο server cluster η σε διαφορετικούς.

Solution Deploy Pipeline



Εικόνα 32: Basic CI/CD flow

Στην παραπάνω εικόνα βλέπουμε ένα τυπικό δείγμα CI/CD pipeline και όπως βλέπουμε απαρτίζεται από τα ακόλουθα components:

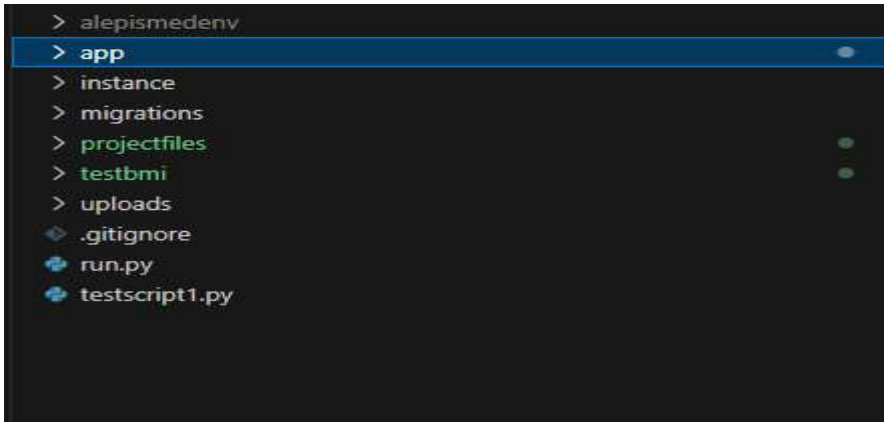
1. Source Control (Continuous integration): Συνήθως χρησιμοποιούνται git/github η τα αντίστοιχα managed cloud services όπως πχ Azure DevOps.
2. Deployment (Continuous Deployment): Κατά την διαδικασία αυτή γίνονται Build τα αντίστοιχα images της εφαρμογής, τρέχουνε συγκεκριμένα test cases και στην συνέχεια γίνονται deploy.
3. Deployment Environment: Την όλη οργάνωση του deploy την αναλαμβάνει το docker compose που χτίζει όλα τα αντίστοιχα περιβάλλοντα, ανοίγει δικτυακές επικοινωνίες και προσβάσεις, τρέχει τα απαραίτητα preparation + test scripts κτλ.

ΥΛΟΠΟΙΗΣΗ

Σε αυτήν την ενότητα θα γίνει μια περιγραφή των συστατικών της λύσης, αναφορικά με τον κώδικα και τα διαφορά frameworks που χρησιμοποιήθηκαν. Θα γίνει επίσης αναφορά στην όλη δομή του backend και του front-end καθώς και όλων αντίστοιχα των components.

BACKEND

Η γενική δομή του backend μας φαίνεται παρακάτω:

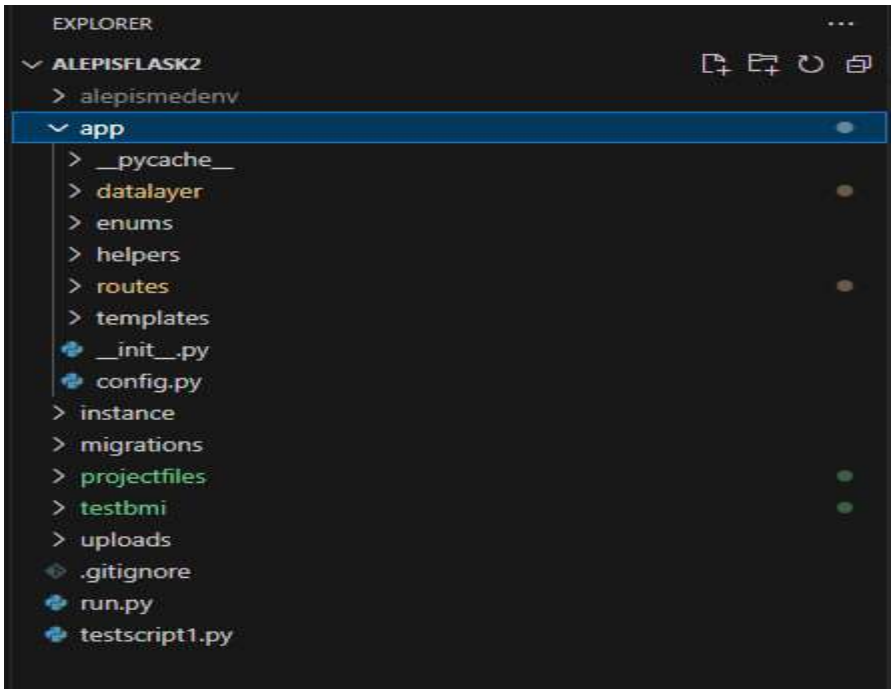


Εικόνα 33: Basic back-end code structure

1. Alepismedenv
    a. Όλα τα σχετικά modules και πακέτα που απαιτεί η λύση μας για να είναι λειτουργική.
2. App
    a. Το κύριο directory με όλα τα files όπου περιέχονται το datalayer και τα μοντέλα μας καθώς και οι δηλώσεις των apis.
3. Instance
    a. Το directory που κρατεί την τρέχουσα κοπιά της βάσης (και παρελθοντικές), και χρησιμοποιείται by default από το flask για να αποθηκεύει local databases.
4. Migrations
    a. Το directory που κρατεί όλη την πληροφορία για την διαχείριση των Migrations της βάσης κατά την προσθήκη/αλλαγή στο μοντέλο, καθώς και για όλες τις αλλαγές στα πεδία.
5. Projectfiles
    a. Το directory που κρατάει διαφορά υποστηρικτικά αρχεία και διαγράμματα για την εφαρμογή.
6. Testbmi
    a. Μια σουίτα για να γίνετε Manual testing των external endpoints που χρησιμοποιούμε και των SaaS services.
7. Uploads
    a. Το directory που κρατάει όλα τα uploaded αρχεία και αντίστοιχα χρησιμοποιείται ως το repo από το οποίο οι χρήστες τραβάνε όλα τα downloads.
8. Gitignore

a. Βασικό αρχείο για εξαίρεση αρχείων/directories από το version tracking.

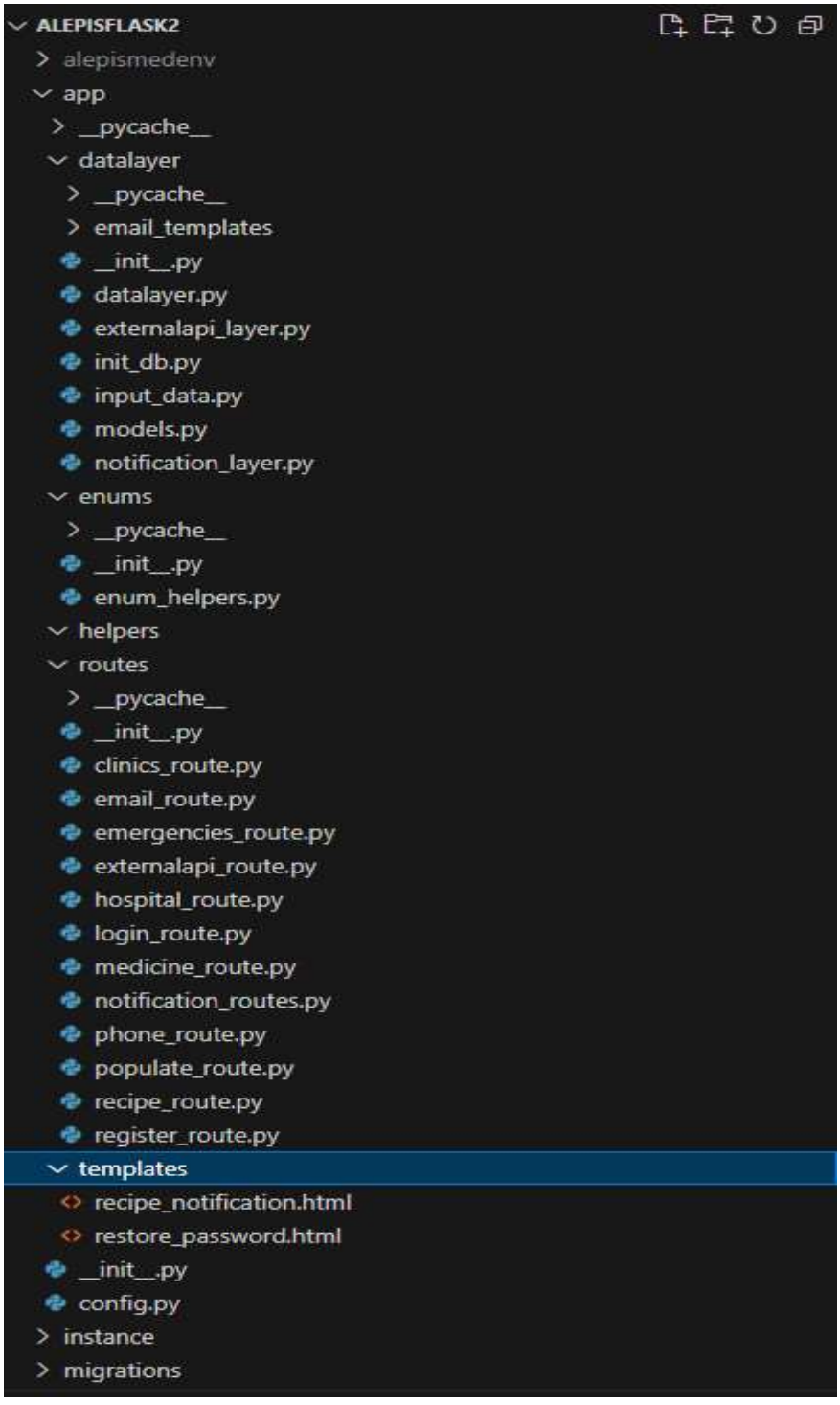
9. run.py
    a. Βασικό αρχείο που ξεκινά τον flask server.



Εικόνα 34: Backend application structure

Στο παραπάνω διάγραμμα βλέπουμε την βασική δομή της εφαρμογής, η οποία ακολουθεί το N-tier architecture paradigm και αποτελείται επιγραμματικά από:

- Datalayer
    i. All actions related to db connectivity and data manipulation.
- Enums
    i. Enumerators to hold predefined set of static values (ie http status codes)
- Helpers
    i. Contains various helper functions to assist the app
- Routes
    i. This is our logic tier, the routes act as an interface between the backend and the frontend.
- Templates
    i. This is where some of our html templates are stored, mostly relating to backend functionalities (ie the mail bodies of notification emails)
- __init__.py
    i. This is where our application factory is housed along with others parameters that need to be loaded on application run.
- Config.py
    i. These are various static parameters related to the database.



Εικόνα 35: Expanded application structure

```python
# app/datalayer/models.py
from datetime import datetime
from flask_sqlalchemy import SQLAlchemy
import hashlib

db = SQLAlchemy()

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    customid = db.Column(db.String(20), unique=True, nullable=False)
    name = db.Column(db.String(100), nullable=False)
    surname = db.Column(db.String(100), nullable=False)
    phone = db.Column(db.String(15), nullable=False)
    address = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    emails = db.relationship('Email', backref='user', lazy=True)
    phones = db.relationship('Phone', backref='user', lazy=True)

    def __init__(self, name, surname, phone, address, email, password):
        self.name = name
        self.surname = surname
        self.phone = phone
        self.address = address
        self.email = email
        self.customid = self.generate_custom_id(email)
        self.password = password

    def generate_custom_id(self, email):
        # Using hashlib to generate a hash from the email
        hashed_email = hashlib.md5(email.encode()).hexdigest()
        return hashed_email[:20]  # Using the first 20 characters of the hash as customid

class Email(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    associated = db.Column(db.String(50))
    email_value = db.Column(db.String(120), nullable=False)  # Added email_value field
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
```

Εικόνα 36: Database models (Code sample)

Στην παραπάνω φωτογραφία βλέπουμε ενδεικτικά την δομή του backend μας όσον αφορά τα μοντέλα δεδομένων, κάθε πίνακας της βάσης αναπαρίσταται σε ένα μοντέλο και έχουμε την δυνατότητα καθώς είναι μοντελοποιημενο σε κλασεις να φτιάξουμε custom μεθόδους που να εξυπηρετούν τις ανάγκες μας.

Datalayer

PopulateDb + Preview methods:

```python
import os
import sys
from flask import jsonify, send_from_directory, current_app
#from app import app  # Import the Flask app instance
from werkzeug.utils import secure_filename

sys.path.insert(0, '.')

from flask import jsonify, request
from app.datalayer.models import db, User, Email, Phone, Hospital, Emergency, Clinic, Files, Medicine, Visitations, Recipe
from app.datalayer.input_data import medicine_data_in
#from app.config import Config

def populate_db():
    # Add some records to the User table
    # user1 = User(customid='user1_customid', phone='123456789', address='Address1', email='user1@example.com')
    # user2 = User(customid='user2_customid', phone='987654321', address='Address2', email='user2@example.com')

    # db.session.add_all([user1, user2])
    # db.session.commit()

    # # Add associated records to the Email and Phone tables
    # email1 = Email(customid='email1_customid', associated='Email Associated 1', user=user1)
    # email2 = Email(customid='email2_customid', associated='Email Associated 2', user=user2)

    # phone1 = Phone(customid='phone1_customid', phoneno='123-456-7890', associated='Phone Associated 1', user=user1)
    # phone2 = Phone(customid='phone2_customid', phoneno='987-654-3210', associated='Phone Associated 2', user=user2)

    # db.session.add_all([email1, email2, phone1, phone2])
    # db.session.commit()

    # return jsonify({'status': '1', 'message': 'Database populated successfully'})

    #Add some records to the User table
    user1 = User(name='John', surname='Doe', phone='123456789', address='Address1', email='user1@example.com', password='samp
    user2 = User(name='Jane', surname='Doe', phone='987654321', address='Address2', email='user2@example.com', password='samp

    db.session.add_all([user1, user2])
    db.session.commit()
```

Εικόνα 37: Populate/Preview methods for debug and testing

Σύνολο από class methods στο datalayer που αποσκοπούν στο να εισάγουν ένα σύνολο από records και συσχετιζόμενα records σε όλους τους βασικούς πίνακες έτσι ώστε να μπορεί να δρομολογηθεί όλο το σύνολο των λειτουργιών που απαιτούνται από το demo.

Ουσιαστικά εισάγει με την σωστή σειρά και με τα σωστά id records αφού ελέγξει για διπλοεγγραφές, δεν επιτρέπει διπλό run καθώς χτυπάει σε primary key constraints.

Εξυπηρετεί τα ακόλουθα:

1. PopulateDb
2. PopulateEmergencies
3. PopulateHospitals
4. PopulateClinics
5. PopulateAssociatedEmails
6. PopulateAssociatedPhones

Join / PreviewJoin methods:

```python
def preview_join(user_id_param=None):
    # Perform a join query to fetch data from users, emails, and phones
    # result = db.session.query(User, Email, Phone). \
    #     join(Email, User.id == Email.user_id). \
    #     join(Phone, User.id == Phone.user_id).all()
    #     #join(Phone, User.id == Phone.user_id).all()

    # if user_id_param is not None:
    #     print("User id param: {0}".format(user_id_param))
    #     query = query.filter(User.id == user_id_param)  # Filter by user_id if provided

    # result = query.all()

    # # Assemble the result in JSON format
    # preview_data = []
    # for user, email, phone in result:
    #     preview_data.append({
    #         'user_id': user.id,
    #         'name': user.name,
    #         'surname': user.surname,
    #         'email': email.email_value,
    #         'phone': phone.phoneno,
    #         'email_associated': email.associated,
    #         'phone_associated': phone.associated
    #     })

    # # Return the result as JSON
    # return jsonify({'preview_data': preview_data})

   # Perform a join query to fetch data from users, emails, and phones
    query = db.session.query(User, Email, Phone). \
        join(Email, User.id == Email.user_id). \
        join(Phone, User.id == Phone.user_id)

    if user_id_param is not None:
        query = query.filter(User.id == user_id_param)  # Filter by user_id if provided

        result = query.all()
```

Εικόνα 38: Join/Preview methods for foreign keys

Σύνολο μεθόδων που εξυπηρετούν το testing επιστρέφοντας συγκεκριμένα Joins πινάκων που επιτρέπουν την λειτουργία του demo, χρήση για testing και παρουσίαση.

Η λειτουργία του βασίζεται σε κλήση στις classes του ORM και σε συνέχεια μετατροπή σε sql κώδικα που εκτελεί inner joins πάνω στην βάση και επιστρέφει τα αντίστοιχα data.

Παρέχει δυνατότητα για λειτουργία επιστροφής όλης της συσχετιζόμενης πληροφορίας για έναν συγκεκριμένο χρήστη που χρησιμοποιείται ευρέως από το front end για να κάνει Populate όλες τις αντίστοιχες σελίδες του site.

```python
def get_emails_by_user_id(user_id):
    # Retrieve all email records for a specific user ID
    emails = Email.query.filter_by(user_id=user_id).all()

    # Assemble the result in JSON format
    email_data = []
    for email in emails:
        email_data.append({
            'email_id': email.id,
            'user_id': email.user_id,
            'associated': email.associated,
            'email_value': email.email_value
        })

    # Return the result as JSON
    return {'email_data': email_data}

def preview_rows_phone():
    # Retrieve all rows from the Phone table
    phones = Phone.query.all()

    # Assemble the result in JSON format
    phone_data = []
    for phone in phones:
        phone_data.append({
            'phone_id': phone.id,
            'user_id': phone.user_id,
            'associated': phone.associated,
            'phoneno': phone.phoneno
        })

    # Return the result as JSON
    return jsonify({'phone_data': phone_data})
```

Εικόνα 39: Model calling on ORM

Στην παραπάνω εικόνα βλέπουμε την υλοποίηση δυο μεθόδων:

1. Get_emails_by_user_id
2. Preview_rows_phone

Και οι δυο μέθοδοι χρησιμοποιούνε το orm (SqlAlchemy) για να μπορούνε να καλέσουνε στα αντίστοιχα μοντέλα των πινάκων της βάσης και να κάνουν abstract όλο το data layer μέσα στο orm. Εκεί οι αντίστοιχες sql εντολές δημιουργούνται και εφαρμόζονται στη βάση. Με την επιστροφή του αποτελέσματος κάνουμε επαναληπτικό βρόγχο πάνω στο αποτέλεσμα για να εφαρμόσουμε το custom formatting που θα χρησιμοποιηθεί από το front-end.

Users table Implementation

```python
def get_user_info_by_id(user_id):
    # Perform a join query to fetch data from users, emails, and phones
    result = db.session.query(User, Email, Phone). \
        join(Email, User.id == Email.user_id). \
        join(Phone, User.id == Phone.user_id). \
        filter(User.id == user_id).all()

    # Assemble the result in a centralized JSON format
    user_info = {}
    emails = set()
    phones = set()

    for user, email, phone in result:
        user_info['user_id'] = user.id
        user_info['user_name'] = user.name
        user_info['user_surname'] = user.surname

        # Collect associated emails and phones
        emails.add(email.email_value)
        phones.add(phone.phoneno)

    # Convert sets to lists for JSON serialization
    user_info['associated_emails'] = list(emails)
    user_info['associated_phones'] = list(phones)

    # Return the result as JSON
    return jsonify(user_info)
```

Εικόνα 40: Basic functionality for user data

Στην παραπάνω εικόνα βλέπουμε την υλοποίηση μιας μεθόδου:

1. Get_user_info_by_id

Και οι δυο μέθοδοι χρησιμοποιούνε το orm (SqlAlchemy) για να μπορούνε να καλέσουνε στα αντίστοιχα μοντέλα των πινάκων της βάσης και να κάνουν abstract όλο το data layer μέσα στο orm. Εκεί οι αντίστοιχες sql εντολές δημιουργούνται και εφαρμόζονται στη βάση. Με την επιστροφή του αποτελέσματος κάνουμε επαναληπτικό βρόγχο πάνω στο αποτέλεσμα για να εφαρμόσουμε το custom formatting που θα χρησιμοποιηθεί από το front-end.

Sample Static Table Implementation

```python
def fetch_all_hospitals():
    # Fetch all hospitals from the database
    hospitals = Hospital.query.all()

    # Assemble the result in JSON formataaa
    hospitals_data = []
    for hospital in hospitals:
        hospitals_data.append({
            'id': hospital.id,
            'hospital_name': hospital.hospital_name,
            'hospital_address': hospital.hospital_address,
            'hospital_maps_link': hospital.hospital_maps_link,
            'hospital_phone': hospital.hospital_phone,
            'hospital_email': hospital.hospital_email
        })

    return jsonify({'hospitals': hospitals_data})
```

Εικόνα 41: Static table implementation

Στην παραπάνω φωτογραφία βλέπουμε ενδεικτικά την μέθοδο fetch_all_hospitals(), αυτή η μέθοδος χρησιμοποιείται από το αντίστοιχο route/endpoint (FetchHospitals) και την καλεί το frontend για να φέρει όλη της αντιστοιχία πληροφορία που χρειάζεται η σελίδα /hospitals.

Η υλοποίηση ακολουθεί την ίδια λογική, με χρήση του SqlAlchemy έχει πρόσβαση στα δεδομένα της βάσης και μετα τα επιστρέφει ανάλογα με το format που απαιτεί το front end.

Sample Table Implementation

```python
def insert_clinic_record(user_id, clinic_name, date_field, expense, comments):
    # Insert a new clinic record
    new_clinic_record = Clinic(user_id=user_id, clinic_name=clinic_name, date_field=date_field, expense
    db.session.add(new_clinic_record)
    db.session.commit()

    return jsonify({'status': '1', 'message': 'New clinic record inserted successfully'})

def fetch_all_clinics_by_user_id(user_id):
    # Fetch all clinic records for a specific user ID
    clinics = Clinic.query.filter_by(user_id=user_id).all()

    # Assemble the result in JSON format
    clinics_data = []
    for clinic in clinics:
        clinics_data.append({
            'clinic_id': clinic.id,
            'clinic_name': clinic.clinic_name,
            'date_field': clinic.date_field,
            'expense': clinic.expense,
            'comments': clinic.comments,
            'user_id': clinic.user_id
        })

    return jsonify({'clinics': clinics_data})


def fetch_all_clinics():
    # Fetch all clinic records
    clinics = Clinic.query.all()

    # Assemble the result in JSON format
    clinics_data = []
    for clinic in clinics:
        clinics_data.append({
            'clinic_id': clinic.id,
            'clinic_name': clinic.clinic_name
```

Εικόνα 42: Sample methods to access clinics table

Στην παραπάνω εικόνα βλέπουμε την αντίστοιχη υλοποίηση μεθόδων που αλληλοεπιδρούν με τον πίνακα clinics και αντίστοιχα υποστηρίζουν την λειτουργικότητα της σελίδας clinics. Η υλοποίηση και εδώ είναι με τη χρήση του orm και αντίστοιχο formatting όπου απαιτείται για την διευκόλυνση του front-end.

Notification Layer

notification_layer.py × recipe_notification.html ...\email_templates recipe_notification.html ...\templates restore_password.html ...\email_

app > datalayer > notification_layer.py > ...

```python
def send_email_flask_mail(subject, recipients, body):
    # msg = Message(subject, recipients=recipients, body=body)
    # mail.send(msg)

    # Create the Message object
    msg = Message(subject=subject, recipients=[recipients], body=body)

    # Send the email using Flask-Mail
    mail.send(msg)

def send_email_sendgrid(subject, recipients, body):
    message = SendGridMail(
        from_email=current_app.config['MAIL_DEFAULT_SENDER'],
        to_emails=recipients,
        subject=subject,
        html_content=body
    )
    try:
        sg = SendGridAPIClient(current_app.config['SENDGRID_API_KEY'])
        sg.send(message)
        return True
    except Exception as e:
        print(str(e))
        return False


def send_email_flask_recipe(recipients, hour_in):
    # Render the HTML template with the provided hour_in value
    html_content = render_template('recipe_notification.html', hour_in=hour_in)

    # template_path = os.path.join('email_templates', 'recipe_notification.html')
    # html_content = render_template(template_path, hour_in=hour_in)

    # Send the email using Flask-Mail
    mail.send_message(subject='Missed Recipe Dose Notification', html=html_content, recipients=[recipients])

def send_email_flask_restorepass(recipients, link_in):
    # TODO: add to postman
    # Render the HTML template with the provided link_in value
    html_content = render_template('restore_password.html', link_in=link_in)
    # Send the email using Flask-Mail
    mail.send_message(subject='Reset Password Link', html=html_content, recipients=[recipients])

```

Εικόνα 43: Notification layer

Στην παραπάνω εικόνα βλέπουμε χαρακτηριστικές μεθόδους που περιέχονται στις classes που υποστηρίζουν τα notifications. Η διαχείριση των email και sms γίνεται με την χρήση πλατφόρμας SaaS (Sendgrid) και της αντίστοιχης βιβλιοθήκης του flask -> FlaskMail.

External Api Utilization

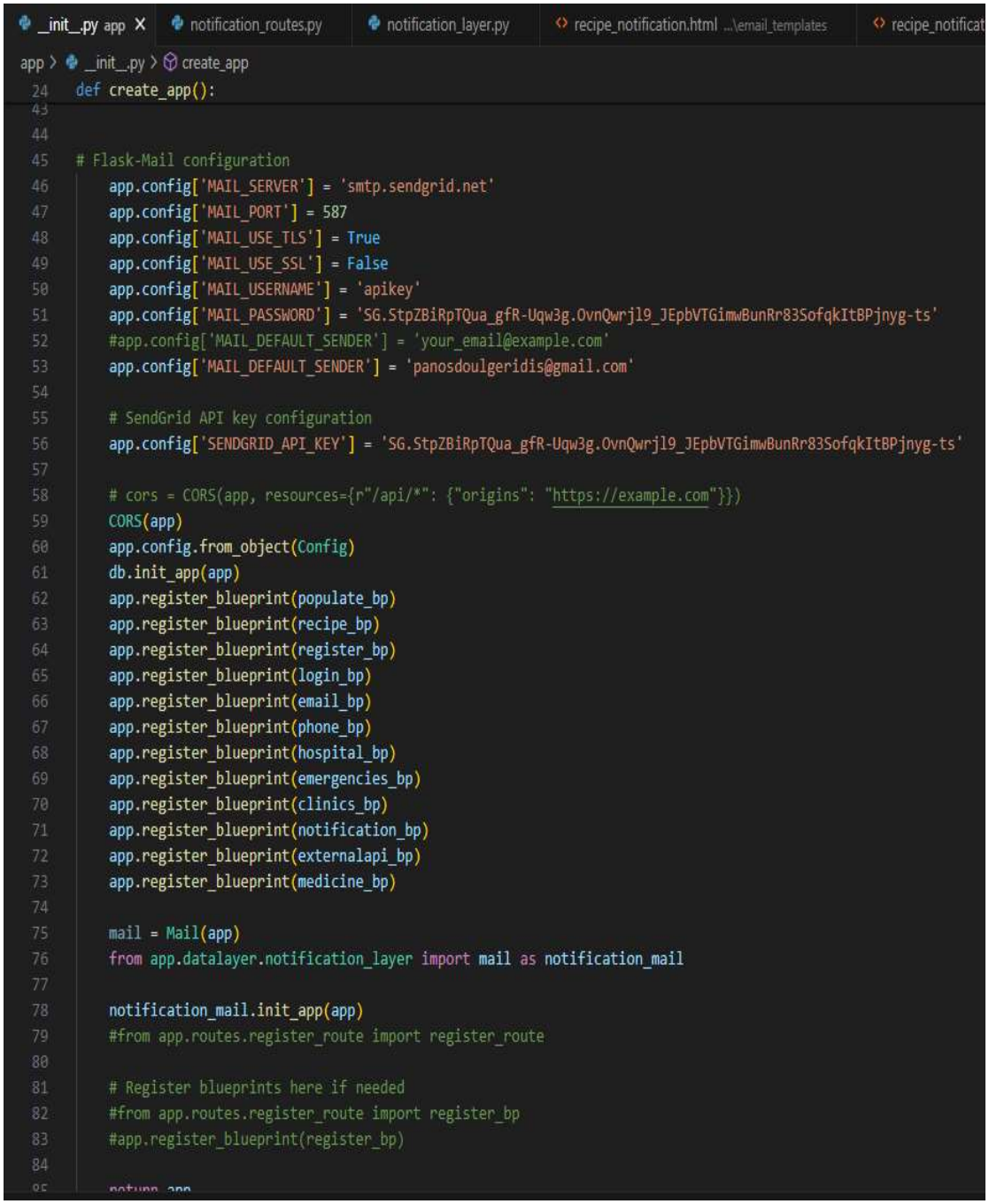
routes > externalapi_route.py > calc_ideal_weight_json

```python
@externalapi_bp.route('/calc-body-fat-form', methods=['POST'])
def calc_body_fat_form():
    # Extract data from form-data
    age = request.form.get('age')
    gender = request.form.get('gender')
    weight = request.form.get('weight')
    height = request.form.get('height')
    neck = request.form.get('neck')
    waist = request.form.get('waist')
    hip = request.form.get('hip')

    # Call the calc_body_fat function
    result =externalapi_layer.calc_body_fat(age, gender, weight, height, neck, waist, hip)

    return jsonify(result)


@externalapi_bp.route('/calc-body-fat-json', methods=['POST'])
def calc_body_fat_json():
    data = request.get_json()
    age = data.get('age')
    gender = data.get('gender')
    weight = data.get('weight')
    height = data.get('height')
    neck = data.get('neck')
    waist = data.get('waist')
    hip = data.get('hip')

    # Call the calc_body_fat function
    result = externalapi_layer.calc_body_fat(age, gender, weight, height, neck, waist, hip)

    return jsonify(result)
```

Εικόνα 44: External API layer

Στην παραπάνω εικόνα βλέπουμε 2 χαρακτηριστικές μεθόδους που χρησιμοποιούν το external api layer για να προσπελάσουν λειτουργικότητες εντός της εφαρμογής και υπολογιστικές σουίτες και εν συνέχεια να υποστηρίξουν τις αντίστοιχες λειτουργικότητες στο frontend.

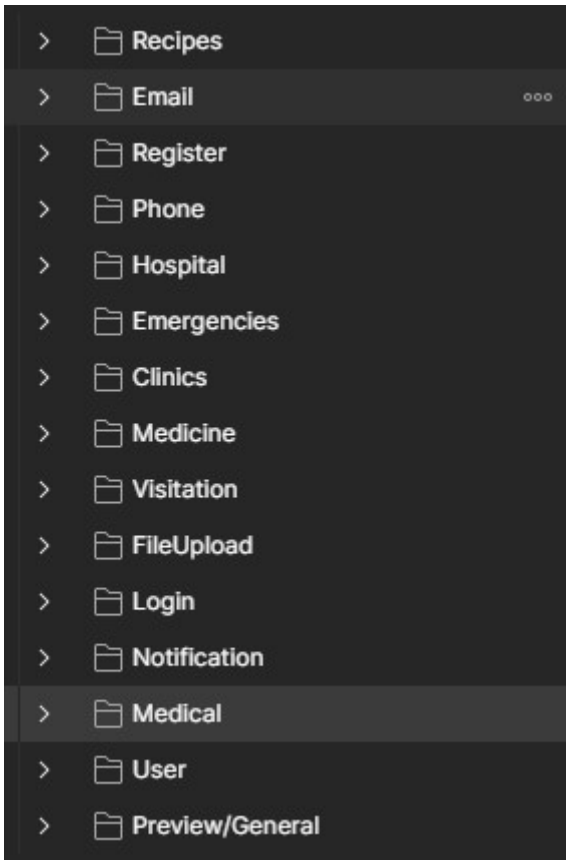
Application Factory



Εικόνα 45: Application factory

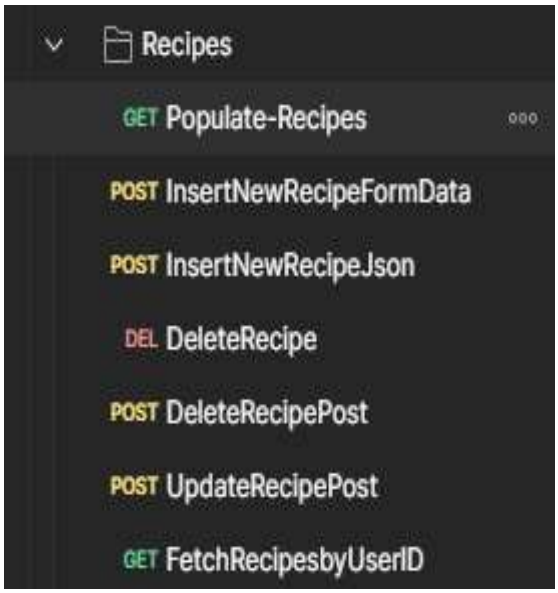
Στην παραπάνω εικόνα βλέπουμε την υλοποίηση αντίστοιχα του application factory σε flask, ο ρόλος του είναι να αρχικοποιησει όλες τις παραμέτρους λειτουργίας της εφαρμογής μας και δρα σαν ένα κέντρο ελέγχου της εφαρμογής, βλέπουμε αντίστοιχα που ορίζονται τα routes ανά θεματική ενότητα καθως και αντίστοιχα η διασύνδεση με τις κλασσεις που διαχειρίζονται τα Notifications και άλλες λειτουργικότητες.

Exposed Apis



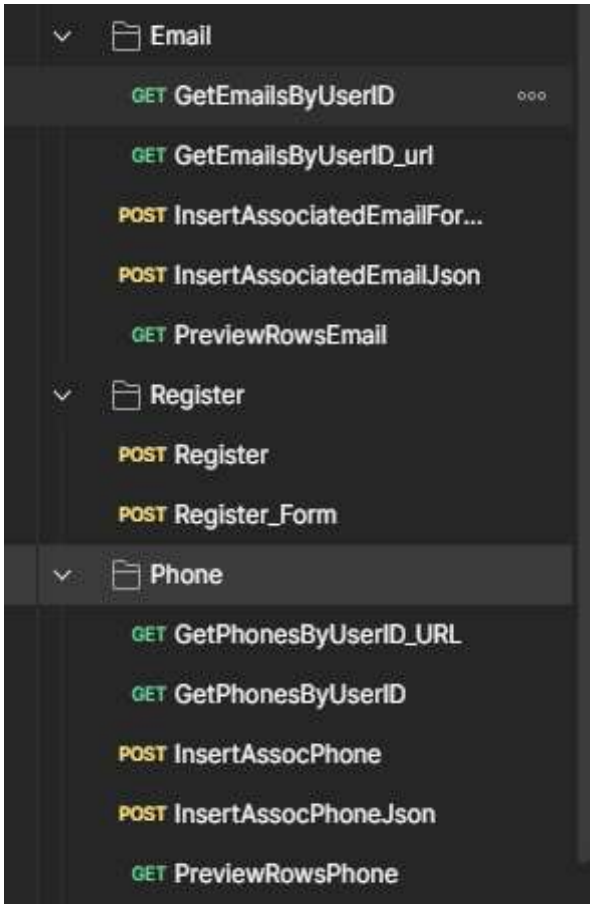
Εικόνα 46: Exposed APIs (All categories)

1. **Recipes:** All available apis that interact with the Recipes db table.
2. **Email:** All available apis that interact with the Email db table.
3. **Register:** Register methods and validations.
4. **Phone:** All available apis that interact with the phone db table.
5. **Hospital:** All available apis that interact with the hospital db table.
6. **Emergencies:** All available apis that interact with emergencies db table.
7. **Clinics:** All available apis that interact with the clinics db table.
8. **Medicine:** All available apis that interact with the medicine db table.
9. **Visitation:** Visitation methods to support base website functionality.
10. **FileUpload:** Upload/Download methods to support base site functionality.
11. **Login:** Login methods and validations to support base site functionality.
12. **Notification:** Notification methods and validations to support base site functionality as long as supporting services.
13. **Medical:** Hosts all related apis that serve medical information as well as medical calculations.
14. **User:** All available apis that interact with User db table.
15. **Preview/General:** Preview/Generic methods to support testing and development.



Εικόνα 47: Exposed Recipes Methods

16. **PopulateRecipes:** Εισάγει ένα σετ από records στον πίνακα των συνταγών για χρήση από testing suites.
17. **InsertNewRecipeFormData:** Εισάγει ένα νέο recipe για ένα συγκεκριμένο χρήστη, βασικό δεδομένο για την λειτουργικότητα των ειδοποιήσεων δοσοληψίας φαρμάκων, χρήση με form data για το frontend.
18. **InsertNewRecipeJson:** Εισάγει ένα νέο recipe για ένα συγκεκριμένο χρήστη, βασικό δεδομένο για την λειτουργικότητα των ειδοποιήσεων δοσοληψίας φαρμάκων, χρήση με json για exposed apis σε 3rd parties.
19. **DeleteRecipe:** Διαγραφή μιας συνταγής για ένα συγκεκριμένο χρήστη, τρέχει αυτόματα με την λήξη της διάρκειας μιας συνταγής. Χρήση από το frontend για manual ενημέρωση του schedule.
20. **DeleteRecipePost:** Διαγραφή μιας συνταγής για ένα συγκεκριμένο χρήστη, τρέχει αυτόματα με την λήξη της διάρκειας μιας συνταγής. Χρήση από το frontend για manual ενημέρωση του schedule, χρήση για exposed apis to 3rd parties.
21. **UpdateRecipePost:** Ενημέρωση μιας συνταγής, δυνατότητα ενημέρωσης όλων των σχετικών παραμέτρων μιας συνταγής, χρήση από το frontend.
22. **FetchRecipesByUserID:** Επιστρέφει όλες τις συσχετιζόμενες ενεργές συνταγές ενός συγκεκριμένου χρήστη, οι συνταγές που έχουν λήξει τίθενται σε ανενεργό status. (Δεν σβήνονται ποτέ)

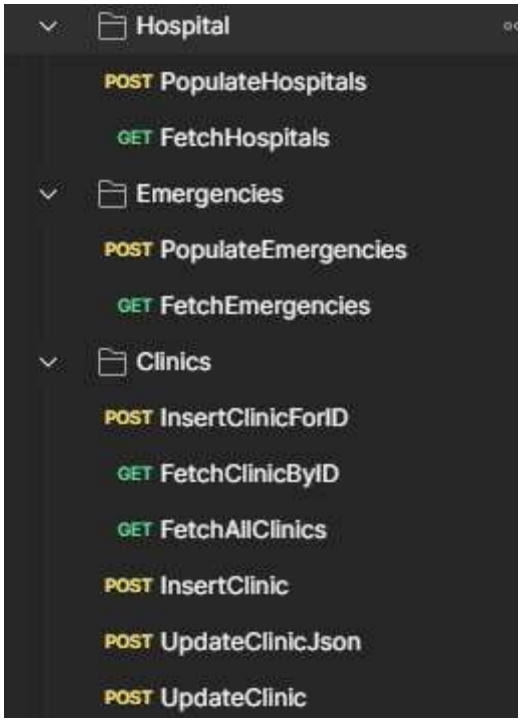


Εικόνα 48: Exposed Email/Register/Phone methods

23. **GetEmailsByUserID:** Επιστρέφει όλα τα συσχετιζόμενα email για έναν συγκεκριμένο χρήστη, λειτουργεί με inner join μεταξύ του πίνακα email/users.
24. **GetEmailsByUserID_url:** Επιστρέφει όλα τα συσχετιζόμενα email για έναν συγκεκριμένο χρήστη, λειτουργεί με inner join μεταξύ του πίνακα email/users, λειτουργία με παραμέτρους στο url, για χρήση από redirect.
25. **InsertAssociatedEmailForID:** Εισάγει ένα νέο συσχετιζόμενα email για έναν συγκεκριμένο χρήστη, βασική λειτουργικότητα για την ενημέρωση συγγενικών προσώπων.
26. **InsertAssociatedEmailJson:** Εισάγει ένα νέο συσχετιζόμενα email για έναν συγκεκριμένο χρήστη, βασική λειτουργικότητα για την ενημέρωση συγγενικών προσώπων, λειτουργία με json για χρήση ως exposed api.
27. **PreviewRowsEmail:** Μέθοδος για να φέρνει κάποια ενδεικτικά records από τον πίνακα των emails, χρήση για testing scenarios.
28. **Register:** Βασική μέθοδος που εγγράφει έναν νέο χρήστη στον βασικό πίνακα των users, περιέχει ένα σετ από validations που εξασφαλίζουν την σωστή μορφοποίηση των δεδομένων και αποτρέπουν τις διπλοεγγραφές. Χρήση με json που αποσκοπεί σαν exposed api σε 3rd parties.
29. **Register_form:** Βασική μέθοδος που εγγράφει έναν νέο χρήστη στον βασικό πίνακα των users, περιέχει ένα σετ από validations που εξασφαλίζουν την σωστή μορφοποίηση των δεδομένων και αποτρέπουν τις διπλοεγγραφές. Χρήση με form data για την λειτουργία του front-end.
30. **GetPhonesByUserID_URL:** Επιστρέφει όλα τα συσχετιζόμενα τηλέφωνα για ένα συγκεκριμένο user id και αποσκοπεί στις λειτουργικότητες των αυτομάτων ενημερώσεων καθώς και να φέρνει την αντίστοιχη πληροφορία για το frontend.
31. **GetPhonesByUserID:** Επιστρέφει όλα τα συσχετιζόμενα τηλέφωνα για ένα συγκεκριμένο user id και αποσκοπεί στις λειτουργικότητες των αυτομάτων

ενημερώσεων καθώς και να φέρνει την αντίστοιχη πληροφορία για το frontend, χρήση με json που αποσκοπεί σε exposed api σε 3rd parties.

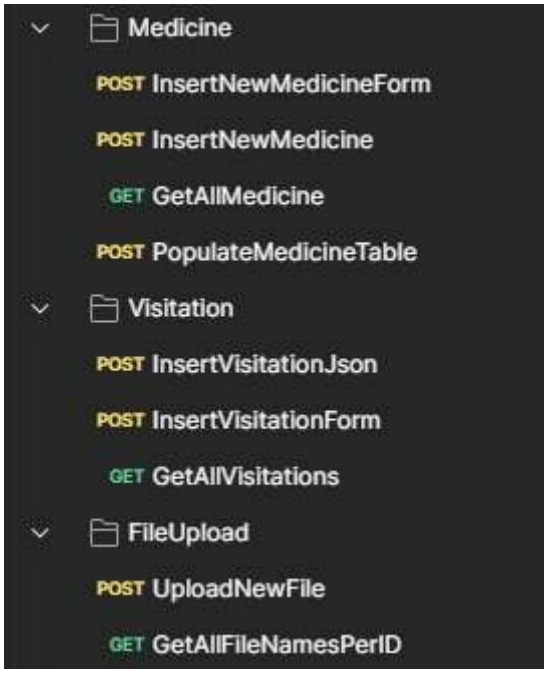
32. **InsertAssocPhone:** Εισάγει ένα νέο τηλέφωνο για ένα συγκεκριμένο χρήστη, βασική λειτουργικότητα για το module των Notifications, λειτουργία με form data για χρήση από το frontend.
33. **InsertAssocPhoneJson:** Εισάγει ένα νέο τηλέφωνο για ένα συγκεκριμένο χρήστη, βασική λειτουργικότητα για το module των Notifications, λειτουργία με json για χρήση από 3rd parties.
34. **PreviewRowsPhone:** Μέθοδος η οποία επιστρέφει ένα δείγμα από τον πίνακα των τηλέφωνων για χρήση από τα testing suites.



Εικόνα 49: Hospital / Clinics apis

35. **PopulateHospitals:** Γεμίζει τον πίνακα των hospitals με ένα baseline ενδεικτικών records που χρησιμοποιείται για testing.
36. **FetchHospitals:** Επιστρέφει όλη την λίστα από τον πίνακα των νοσοκομείων, χρήση για Population των αντίστοιχων dropdowns.
37. **PopulateEmergencies:** Γεμίζει τον πίνακα των emergencies με ένα baseline ενδεικτικών records που χρησιμοποιείται για testing.
38. **FetchEmergencies:** Επιστρέφει όλη την λίστα από τον πίνακα των επειγόντων, χρήση για Population των αντίστοιχων dropdowns.
39. **InsertClinicForID:** Εισάγει ένα νέο record στον πίνακα με τις κλινικές για συγκεκριμένο χρήστη / user id.
40. **FetchClinicByID:** Επιστρέφει το record από τον πίνακα με τις κλινικές για συγκεκριμένο χρήστη / user id.
41. **FetchAllClinics:** Επιστρέφει όλες τις κλινικές για όλα τα user_id, χρήση για να γίνει populate ο πίνακας των κλινικών με όσες το δυνατόν περισσότερες κλινικές.
42. **InsertClinic:** Εισάγει μια νέα κλινική agnostic από user_id, η κλινική αυτή γίνεται διαθέσιμη σε όλους τους χρήστες, χρήση για populate dropdowns στο frontend.

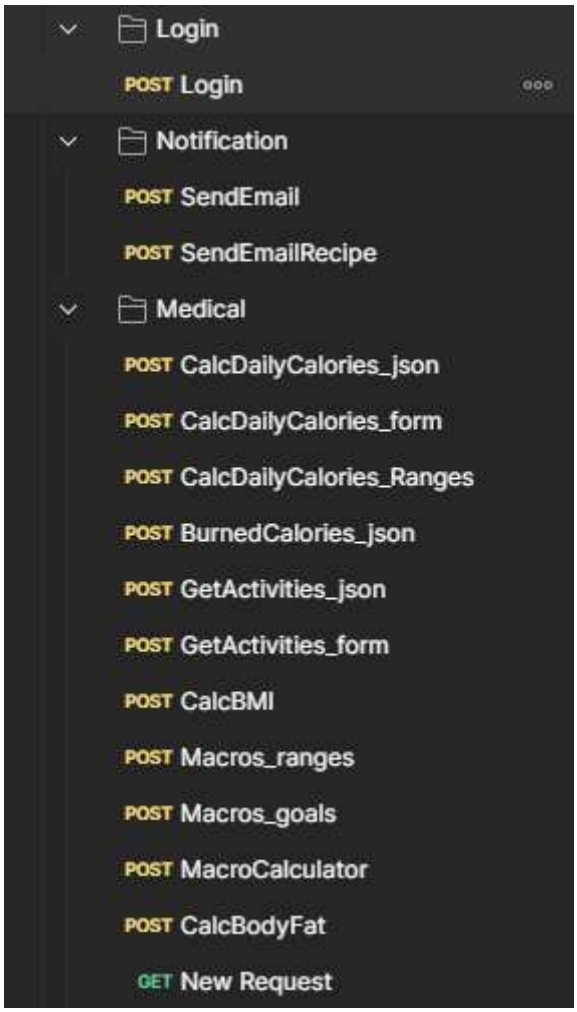
43. **UpdateClinicJson:** Ενημερώνει την πληροφορία μιας συγκεκριμένης κλινικής, δυνατότητα για ενημέρωση ολων των αντιστοιχων πεδιων του πινακα, χρηση με Json payload.
44. **UpdateClinic:** Ενημερωνει την πληροφορια μιας συγκεκριμενης κλινικης, δυνατοτητα για ενημερωση ολων των αντιστοιχων πεδιων του πινακα, χρηση με form data.



Εικόνα 50: Medicine / Visitation / FileUpload APIs

1. **InsertNewMedicineForm:** Εισάγει ένα νέο record στον πίνακα των φαρμάκων με βάση τα form data.
2. **InsertNewMedicine:** Εισάγει ένα νέο record στον πίνακα των φαρμάκων με βάση json data format.
3. **GetAllMedicine:** Φέρνει όλη την αποθηκευμένη πληροφορία στον πίνακα των φαρμάκων / medicine.
4. **PopulateMedicineTable:** Βασικό api που εισάγει πολλαπλά records στον πίνακα των φαρμάκων για χρήση από debug.
5. **InsertVisitationJson:** Εισάγει ένα νέο record στον πίνακα των visitation με βάση json data format.
6. **InsertVisitationForm:** Εισάγει ένα νέο record στον πίνακα των visitation με βάση form data.
7. **GetAllVisitations:** Φέρνει όλες τις αποθηκευμένες καταχωρήσεις στον πίνακα των visitation για όλους τους χρήστες.
8. **GetVisitationsByUser**: Φέρνει όλες τις αποθηκευμένες καταχωρήσεις στον πίνακα των visitation για συγκεκριμένο χρήστη.
9. **GetVisitationsByHospital:** Φέρνει όλες τις αποθηκευμένες καταχωρήσεις στον πίνακα των visitation για όλα τα νοσοκομεία.
10. **UploadNewFile:** Βασικό api για το upload ενός αρχείου στο server storage και αντίστοιχα στον πίνακα διατήρησης της συσχέτισης αρχείου με user_id.
11. **GetAllFileNamesPerID:** Επιστρέφει όλα τα αντιστοιχία αρχεία που έχει κάνει upload ένας χρήστης, λειτουργεί με inner join στον πίνακα των users, χρήση από το front-end.

12. **GetFileFromID**: Με βάση ένα συγκεκριμένο file id κάνει download το αντίστοιχο αρχείο στο localhost του χρήστη.



Εικόνα 51: Login / Notification / General Medical apis

1. **Login:** Βασική λειτουργικότητα Login, περιλαμβάνει έλεγχο ύπαρξης του χρήστη και στην συνέχεια authorization via json web token. Οι ρολόι που υπάρχουν στην εφαρμογή είναι users, associated_users και admins.
2. **SendEmail:** Api που περιέχει την βασική λειτουργικότητα αποστολής email και αναφέρεται σε ειδοποιήσεις.
3. **SendEmailRecipe:** Api που περιέχει την βασική λειτουργικότητα αποστολής email και αναφέρεται σε ειδοποιήσεις που αναφέρονται στις συνταγές.
4. **CalcDailyCalories_json:** Βασική διεπαφη με την SaaS υπηρεσία για τον υπολογισμό καθημερινών θερμίδων σε σχέση με τους τύπους δίαιτας.
5. **CalcDailyCalories_form**: Βασική διεπαφη με την SaaS υπηρεσία για τον υπολογισμό καθημερινών θερμίδων σε σχέση με τους τύπους δίαιτας, δουλεύει με form data.
6. **CalcDailyCalories_Ranges:** Βασική διεπαφη για να γίνουν Populate τα dropdown διαγράμματα για τον υπολογισμό των θερμίδων.
7. **BurnedCalories:** Βασική διεπαφη για υπολογισμό των καμένων θερμίδων ανά τύπο άσκησης / δραστηριότητας.

8. **GetActivities_json:** Βασική διεπαφη για το populate των dropdown lists για τα activities.
9. **GetActivities_Form:** Βασική διεπαφη για το populate των dropdown lists για τα activities, δεδομένα τύπου φόρμας.
10. **CalcBMI:** Βασική διεπαφη για τον υπολογισμό δείκτη μάζας σώματος με Input βιομετρικά δεδομένα.
11. **Macros_ranges:** Βασική διεπαφη για το Population των dropdown που αφορούν τροφές.
12. **Macros_goals:** Βασική διεπαφη για το Population των dropdown που αφορούν στόχους δίαιτας.
13. **MacroCalculator:** Βασική διεπαφη για τον υπολογισμό αντίστοιχης δίαιτας με βάση άσκηση και τροφές.
14. **CalcBodyFat:** Βασική διεπαφη για τον υπολογισμό δείκτη μάζας σώματος.

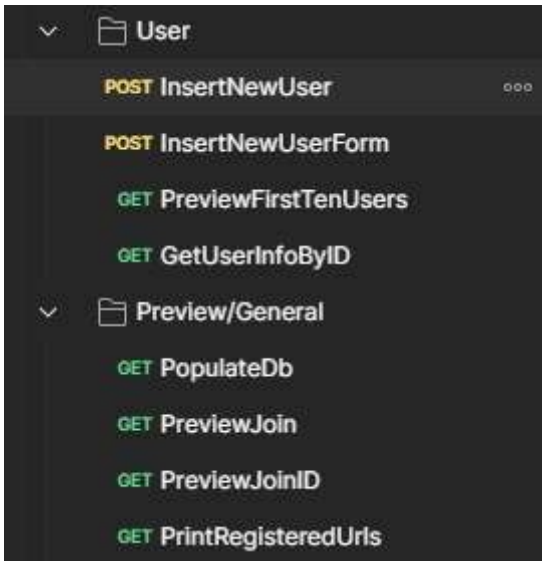


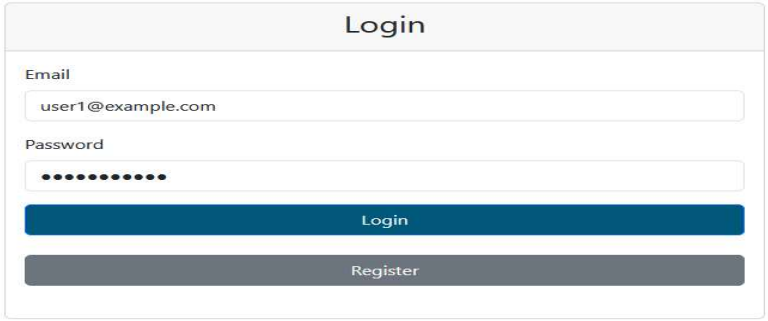
Figure 52- Api methods for User, Preview functionality

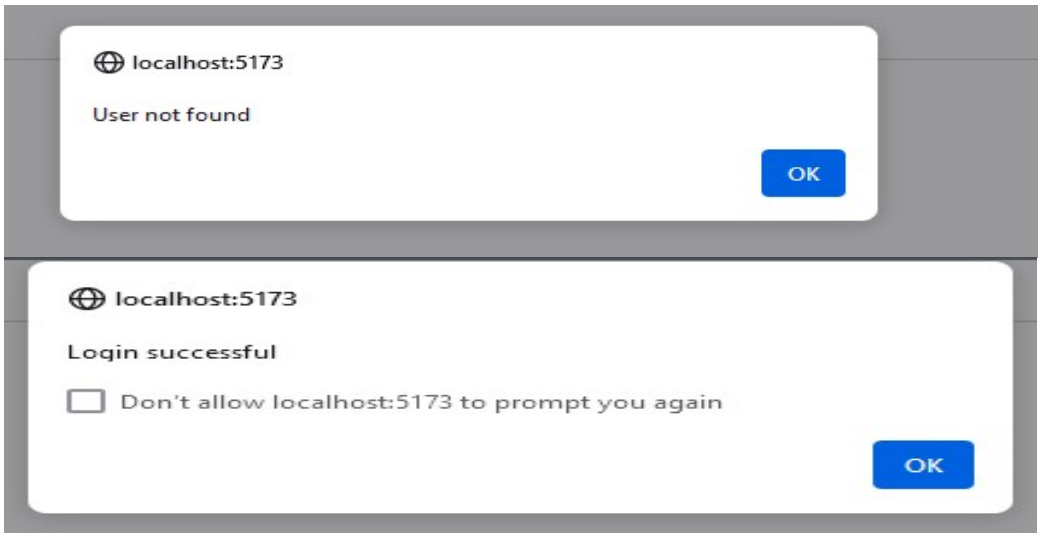
Στην παραπάνω εικόνα βλέπουμε τις αντίστοιχες μεθόδους που περιλαμβάνουν τις λειτουργικότητες σχετικά με τους χρήστες καθώς και διάφορες λειτουργικότητες που βοηθούν στο testing/debugging και preview. Επιγραμματικά:

1. **InsertNewUser:** Βάζει ένα νέο χρήστη μετα από ένα σύνολο validation μέσα στον πίνακα των users
2. **InsertNewUserForm:** Βάζει ένα νέο χρήστη μετα από ένα σύνολο από validation μέσα στον πίνακα των users, χρησιμοποιώντας form data.
3. **PreviewFirstTenUsers:** Επιστρέφει ενδεικτικά τις πρώτες 10 εγγραφές στον πίνακα των users, χρήση για debugging/testing
4. **GetUserInfoByID:** Επιστρέφει όλη την σχετική πληροφορία για ένα χρήστη με Input των user id.
5. **PopulateDb:** Βάζει μια σειρά από διασυνδεδεμένα records σε όλους τους βασικούς πίνακες της βάσης με σκοπό να υπάρχουνε data για testing.
6. **PreviewJoin:** Περιέχει ενδεικτικά Join πάνω στους πιο βασικούς πίνακες για να έχουμε on-the-spot debugging των διάφορων inner joins.
7. **PreviewJoinID:** Περιέχει ενδεικτικά Join πάνω στους πιο βασικούς πίνακες για να έχουμε on-the-spot debugging των διάφορων inner joins με βάση το ID.
8. **PrintRegisteredUrls:** Τυπώνει όλα τα διαθέσιμα URL που κάνει expose το backend μας.

Το τελικό αποτέλεσμα

Login



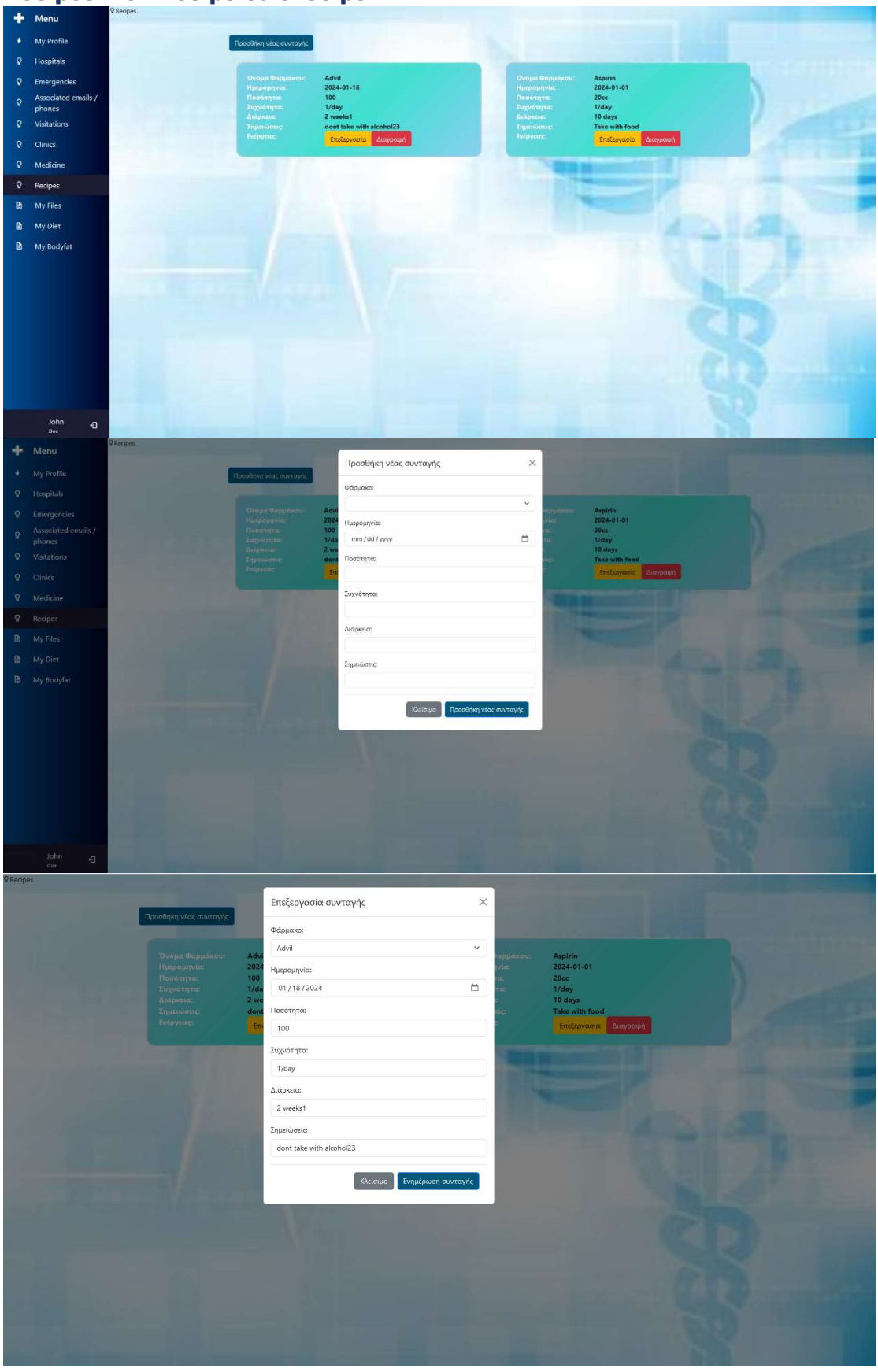


Signup

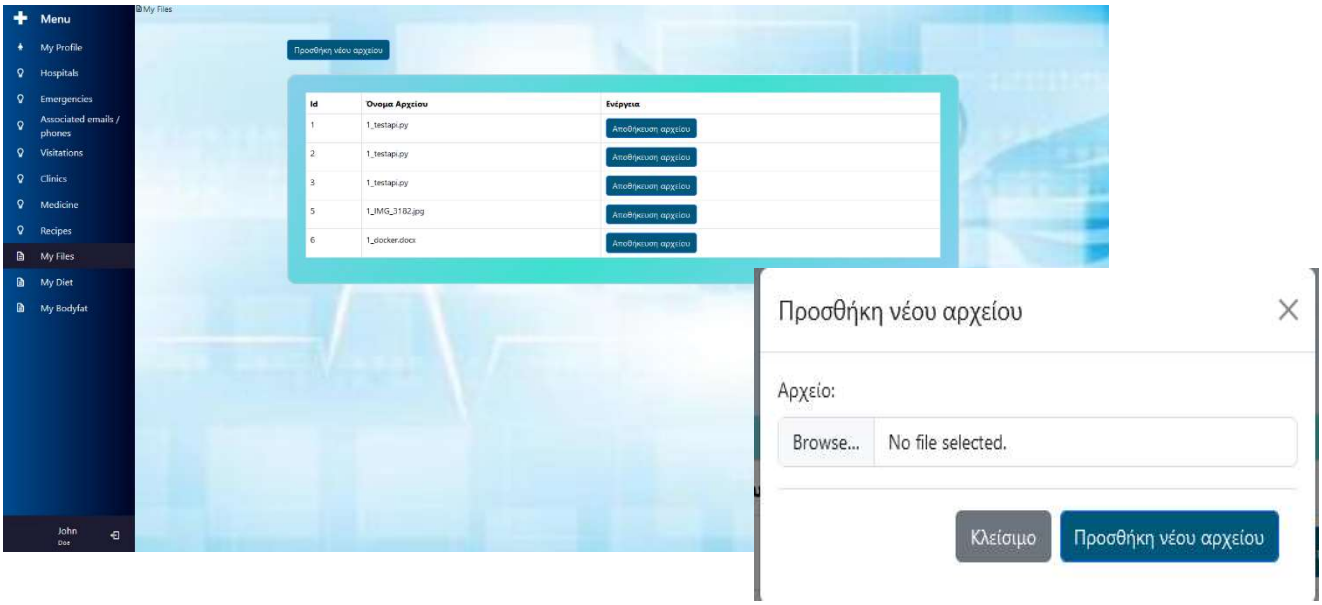
Myprofile


Menu
My Profile
Hospitals
Emergencies
Associated emails / phones
Visitations
Clinics
Medicine
Recipes
My Files
My Diet
My Bodyfat
John
Doe
My Profile
Προσωπικές Πληροφορίες
Όνομα:
John
Επώνυμο:
Doe
Τηλέφωνο:
123-456-7890
987-654-3213
987-654-3211
2105222025
987-654-3212
Email:
H@c.gr
email3@example.com
email1@example.com

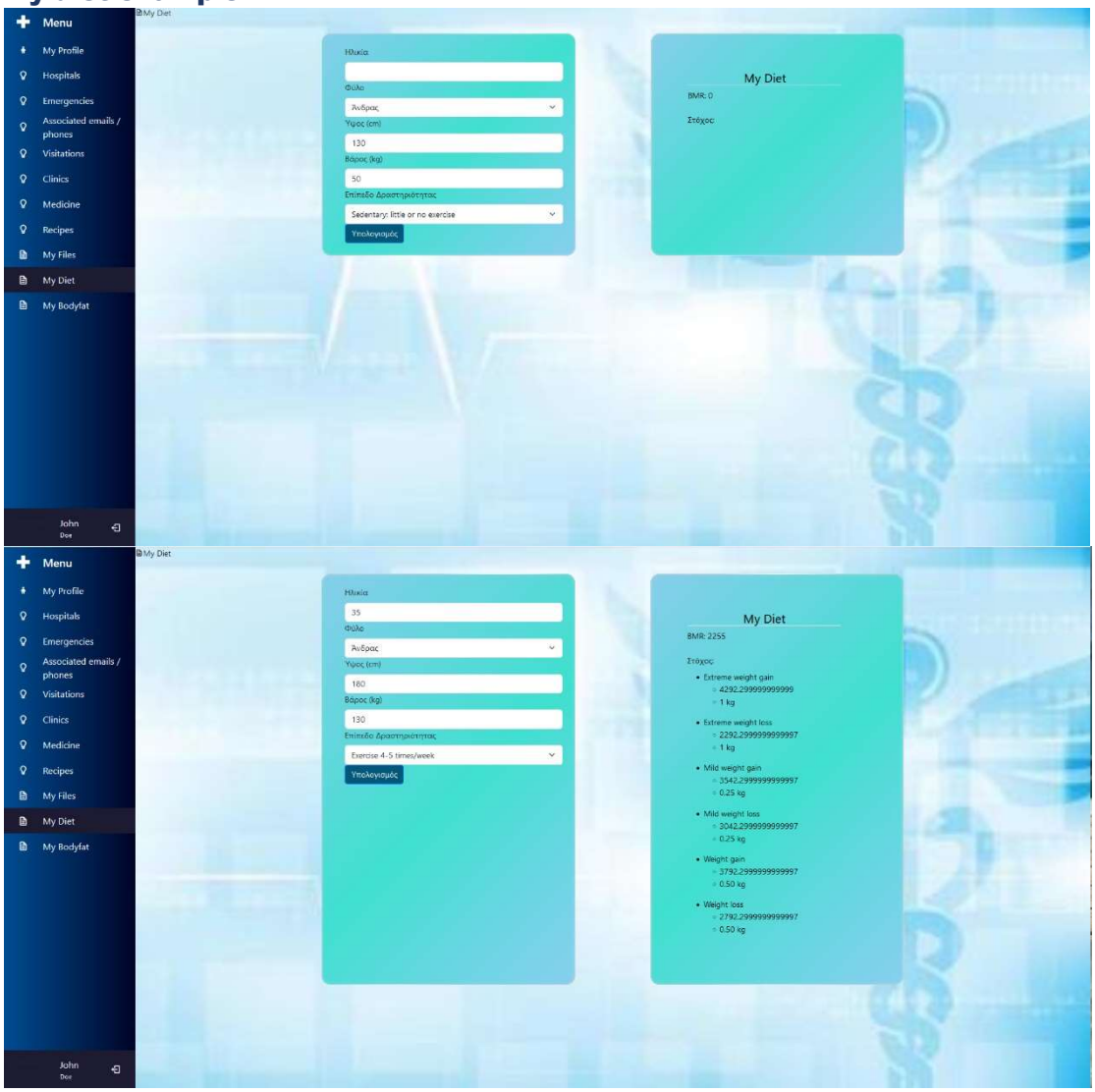
Recipes new-recipe edit-recipe



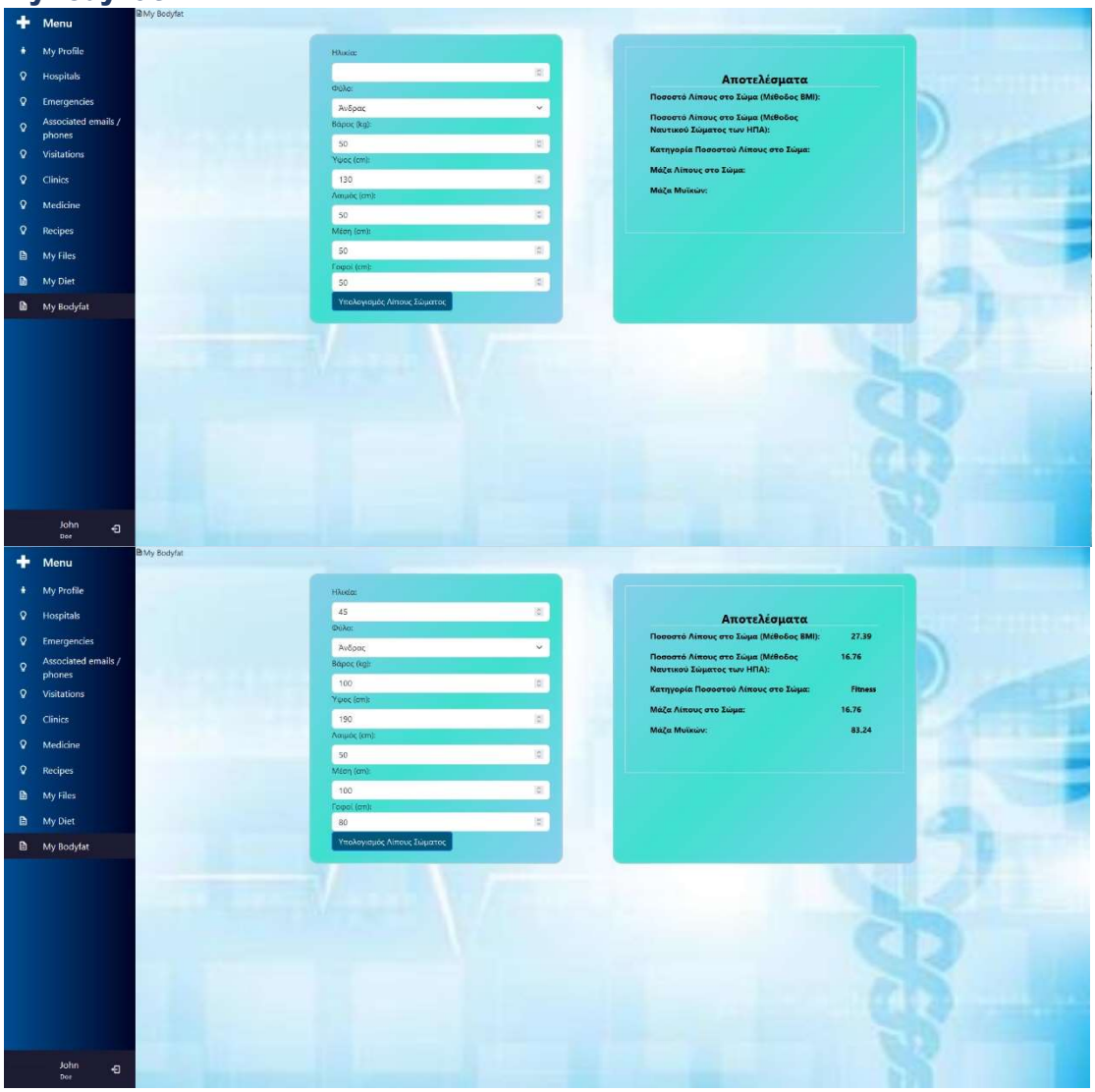
Files


Menu
My Profile
Hospitals
Emergencies
Associated emails / phones
Visitations
Clinics
Medicine
Recipes
My Files
My Diet
My Bodyfat
John Doe
My Files
Προσθήκη νέου αρχείου
Id
Όνομα Αρχείου
Ενέργεια
1 1_testapi.py Αποθήκευση αρχείου
2 1_testapi.py Αποθήκευση αρχείου
3 1_testapi.py Αποθήκευση αρχείου
5 1_IMG_3182.jpg Αποθήκευση αρχείου
6 1_docker.docx Αποθήκευση αρχείου
Προσθήκη νέου αρχείου
Αρχείο:
Browse...
No file selected.
Κλείσιμο
Προσθήκη νέου αρχείου

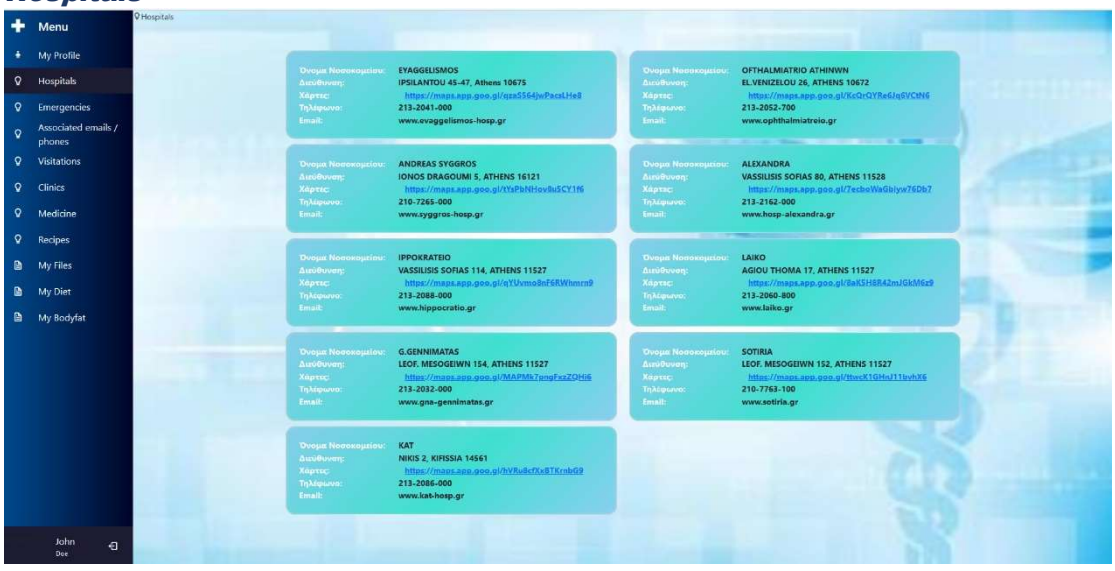
My diet example



MyBodyFat

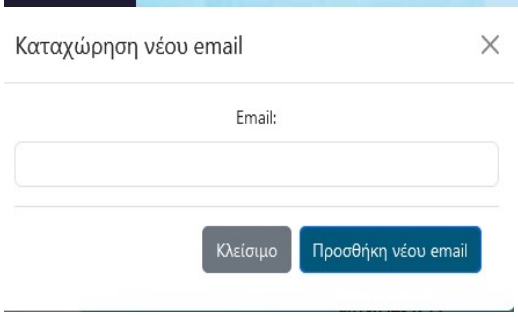


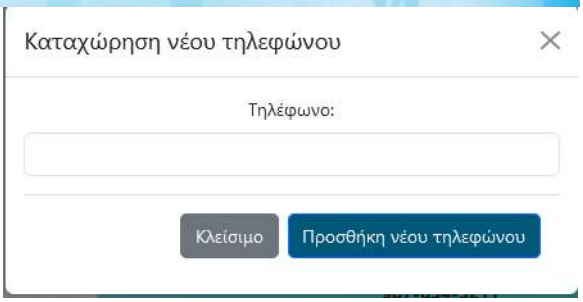
Hospitals



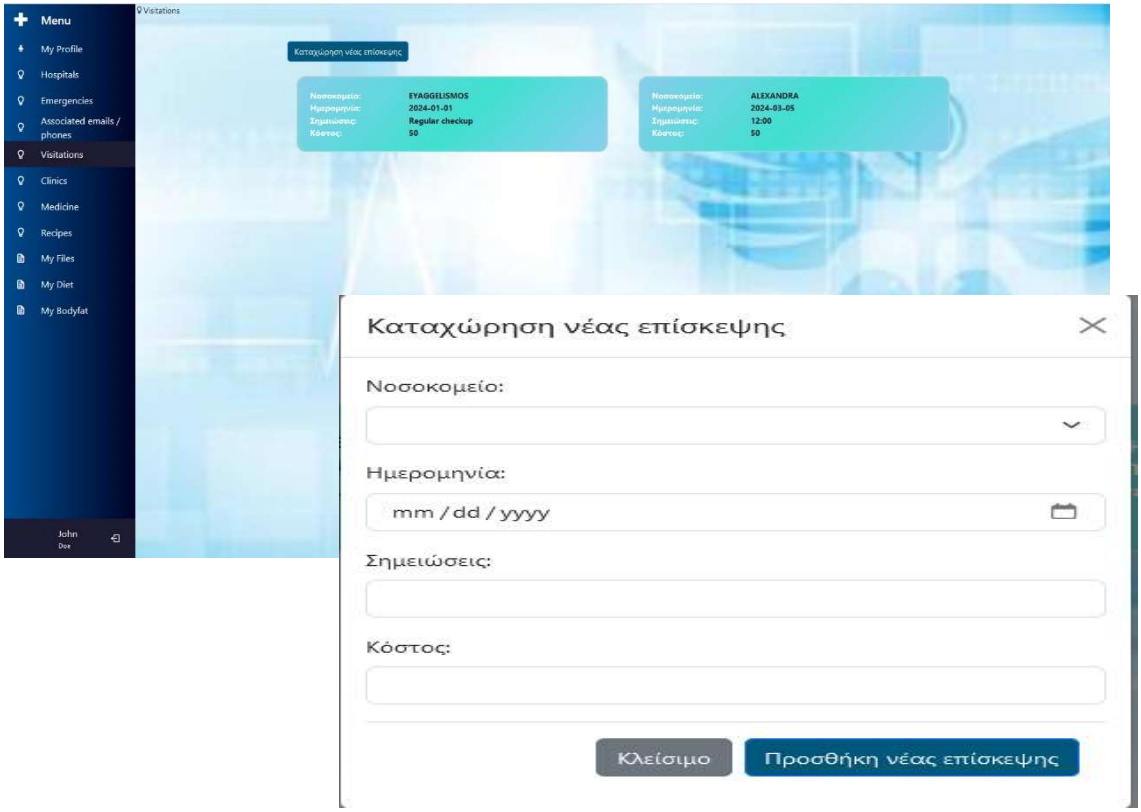
Emergencies

Phone-emails / new- phone new-email

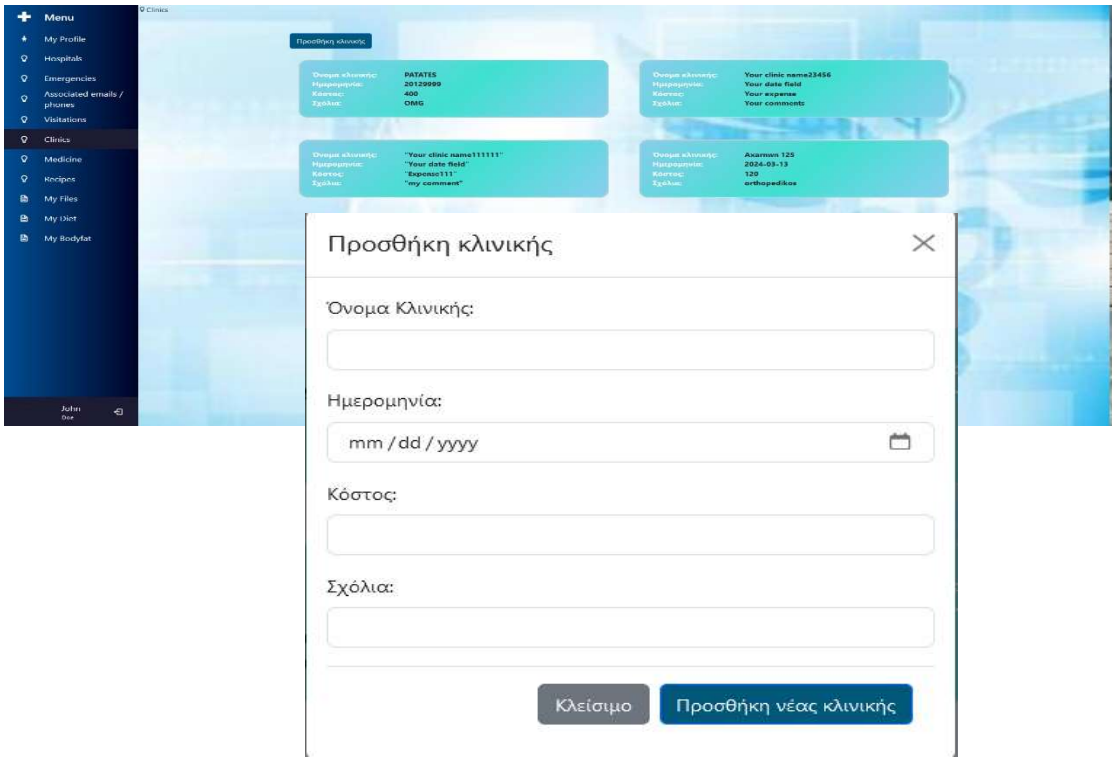




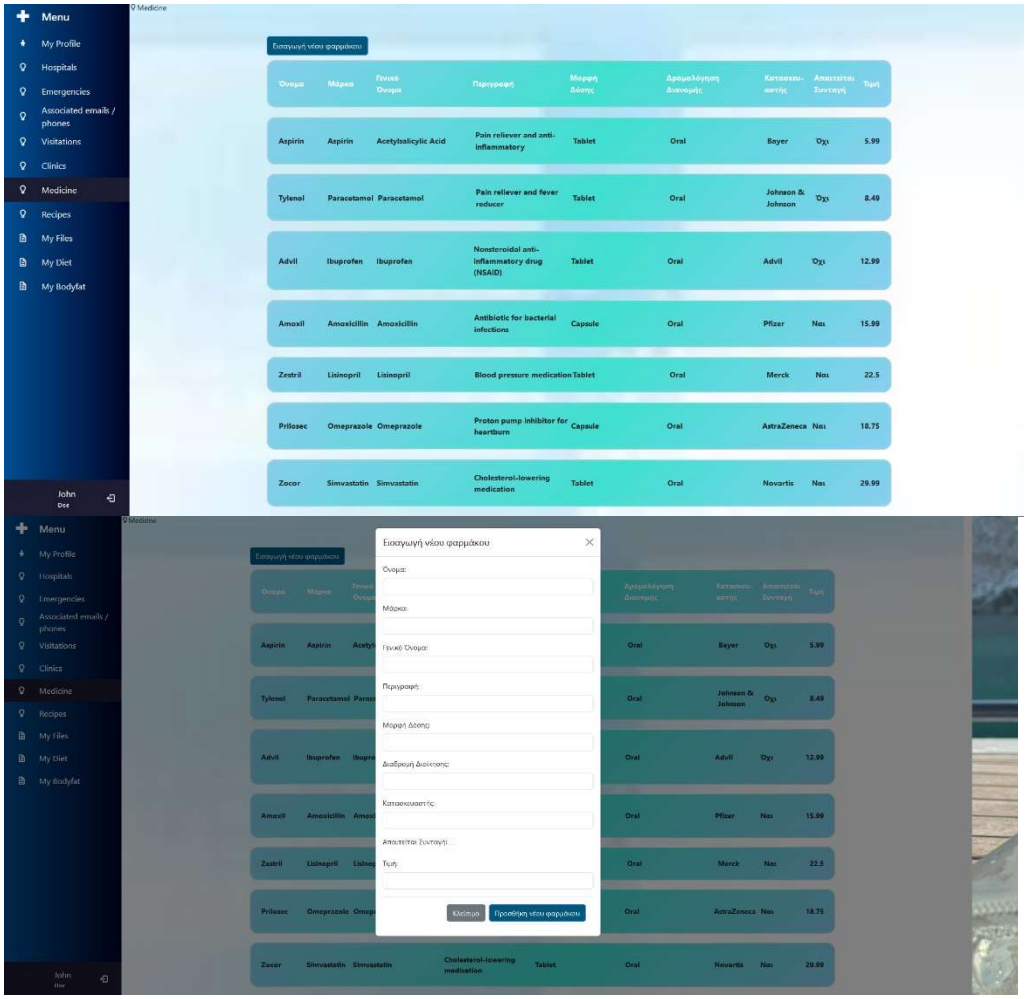
Visitations new-visitation



Clinics new-clinic



Medicine new-medicine



ΣΥΜΠΕΡΑΣΜΑΤΑ - ΠΕΡΙΛΗΨΗ

Όπως είδαμε και στις προηγούμενες ενότητες, η τελική υλοποίηση περιλαμβάνει αλληλεπίδραση πολλών τμημάτων που διαχειρίζονται διαφορετικές τεχνολογίες και απαιτεί αρκετά μεγάλη εξοικείωση με πολλές διαφορετικές γλώσσες και συντακτικά. Παρ όλα αυτά ο τρόπος που στήθηκε μας δίνει ουσιαστικά μια Modular και easily portable/deployable εφαρμογή που να μπορεί να εξυπηρετήσει οποιοδήποτε front-end η να δρα σαν standalone layer που κάνει expose συγκεκριμένα apis.

Είναι επίσης πολύ εύκολο να γίνει Port της εφαρμογής σε Mobile καθώς έχει επιλεχθεί Lightweight serverless database και ουσιαστικά η μόνη επέμβαση που θα χρειαζότανε να γίνει είναι στο εικαστικό κομμάτι έτσι ώστε να γίνει responsive.

Αντίστοιχα η εφαρμογή μας μπορεί μέσω του docker / docker-compose να γίνει πολύ εύκολα deploy σαν Platform-as-a-service υπηρεσία σε cloud περιβάλλοντα (ie Azure Kubernetes Cluster) και έτσι να επωφεληθεί από όλα τα οφέλη που θα μπορούσε να μας παρέχει ένα cloud deployment (Resiliency, High Availability, Scalability, Pay-as-you-go costing).

ΒΙΒΛΙΟΓΡΑΦΙΑ

- Alonistioti, Nancy, Evangelia Aikaterini Tsichrintzi, Konstantina Chrysafiadi, and Efthimios Alepis. 2023. “Requirements for Fuzzy Logic in Personalisation of Fire Emergency Alerts.” In *2023 14th International Conference on Information, Intelligence, Systems & Applications (IISA)*, 1–8. IEEE.
- Argyropoulos, Vasileios, Efthimios Alepis, and Constantinos Patsakis. 2022. “Semi-Decentralized File Sharing as a Service.” In *2022 13th International Conference on Information, Intelligence, Systems & Applications (IISA)*, 1–8. IEEE.
- Bilika, Domna, Nikoletta Michopoulou, Efthimios Alepis, and Constantinos Patsakis. “Hello Me, Meet the Real Me: Voice Synthesis Attacks on Voice Assistants.” *Computers & Security* 137: 103617.
- Douladiris, Anargyros, and Efthimios Alepis. 2023. “Covid-19 New Cases Correlation Analysis: Weather Conditions, Citizen Traffic and Vaccination Statistics Impact in NARX Estimated Regressions in Attica, Greece.” In *2023 14th International Conference on Information, Intelligence, Systems & Applications (IISA)*, 1–7. IEEE.
- Giannikis, Athanasios, Efthimios Alepis, and Maria Virvou. 2021. “Crowdsourcing Recognized Image Objects in Mobile Devices Through Machine Learning.” In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, 560–67. IEEE.
- Kapetanios, Constantinos, Theodoros Polyzos, Efthimios Alepis, and Constantinos Patsakis. 2021. “This Is Just Metadata: From No Communication Content to User Profiling, Surveillance and Exploitation.” *Advances in Core Computer Science-Based Technologies: Papers in Honor of Professor Nikolaos Alexandris*, 277–302.
- Kontogianni, Aristea, and Efthimios Alepis. 2020. “Smart Tourism: State of the Art and Literature Review for the Last Six Years.” *Array* 6: 100020.
- “AI, Blockchain & Cyber Tourism Joining the Smart Tourism Realm.” In *2022 13th International Conference on Information, Intelligence, Systems & Applications (IISA)*, 1–6. IEEE.
- “Social Network Data Enabling Smart Tourism.” In *2023 14th International Conference on Information, Intelligence, Systems & Applications (IISA)*, 1–6. IEEE.
- Kontogianni, Aristea, Efthimios Alepis, and Constantinos Patsakis. 2022a. “Promoting Smart Tourism Personalised Services via a Combination of Deep Learning Techniques.” *Expert Systems with Applications* 187: 115964.
- 2022b. “Smart Tourism and Artificial Intelligence: Paving the Way to the Post-Covid-19 Era.” *Advances in Artificial Intelligence-Based Technologies: Selected Papers in Honour of Professor Nikolaos G. Bourbakis—Vol. 1*, 93–109.
- Matzavela, Vasiliki, and Efthimios Alepis. 2021. “M-Learning in the COVID-19 Era: Physical Vs Digital Class.” *Education and Information Technologies* 26 (6): 7183–203.

- 2023. “An Application of Self-Assessment of Students in Mathematics with Intelligent Decision Systems: Questionnaire, Design and Implementation at Digital Education.” *Education and Information Technologies*, 1–16.
- Michail, Tselepatiotis, and Efthimios Alepis. 2023. “Design of Real-Time Multiplayer Word Game for the Android Platform Using Firebase and Fuzzy Logic.” In *2023 14th International Conference on Information, Intelligence, Systems & Applications (IISA)*, 1–8. IEEE.
- Patsakis, Constantinos, Eugenia Politou, Efthimios Alepis, and Julio Hernandez-Castro. 2023. “Cashing Out Crypto: State of Practice in Ransom Payments.” *International Journal of Information Security*, 1–14.
- Politou, Eugenia, Efthimios Alepis, Maria Virvou, and Constantinos Patsakis. 2022. “Privacy and Data Protection Challenges in the Distributed Era.” Springer.
- Politou, Eugenia, Efthimios Alepis, Maria Virvou, Constantinos Patsakis, Eugenia Politou, Efthimios Alepis, Maria Virvou, and Constantinos Patsakis. 2022a. “Open Questions and Future Directions.” *Privacy and Data Protection Challenges in the Distributed Era*, 175–80.
- 2022b. “State-of-the-Art Technological Developments.” *Privacy and Data Protection Challenges in the Distributed Era*, 69–91.
- Sigala, Effrosyni, Efthimios Alepis, and Constantinos Patsakis. 2020. “Measuring the Quality of Street Surfaces in Smart Cities Through Smartphone Crowdsensing.” In *2020 11th International Conference on Information, Intelligence, Systems and Applications (IISA*, 1–8. IEEE.
- Triantafyllou, Andreas M, George A Tsihrintzis, Maria Virvou, and Efthimios Alepis. 2021. “A Bimodal System for Emotion Recognition via Computer of Known or Unknown Persons in Normal or Fatigue Situations.” In *Advances in Core Computer Science-Based Technologies*, 9–35. Springer, Cham.
- Virvou, Maria, Efthimios Alepis, George A Tsihrintzis, and Lakhmi C Jain. 2020. “Machine Learning Paradigms: Advances in Learning Analytics.” *Machine Learning Paradigms: Advances in Learning Analytics*, 1–5.
- Toby J. Teorey, Sam S. Lightstone, Tom Nadeau - "Database Modeling and Design: Logical Design, Morgan Kaufmann"
- Louis Davidson, Jessica Moss - "Pro SQL Server Relational Database Design and Implementation, Apress"
- Nigel Poulton - "The Docker Book: Containerization Is the New Virtualization, Independently published"
- Sébastien Goasguen - "Docker Cookbook: Over 100 practical and insightful recipes to build distributed applications with Docker, Packt Publishing"
- Adrian Mouat - "Using Docker: Developing and Deploying Software with Containers, O'Reilly Media"
- Docker. (n.d.). Docker Documentation. Retrieved from https://docs.docker.com/
    - This is the official documentation for Docker, providing comprehensive information on Docker containers, Docker Engine, Docker Compose, and related tools.
- Docker Compose. (n.d.). Docker Documentation.
    - Retrieved from https://docs.docker.com/compose/
    - Official documentation for Docker Compose, offering guidance on how to define and run multi-container Docker applications.
- SQLite. (n.d.). SQLite Documentation.

    - Retrieved from https://www.sqlite.org/docs.html
    - Official documentation for SQLite, including guides, references, and tutorials for utilizing SQLite in your project.
- Python Software Foundation. (n.d.). Python Documentation.
    - Retrieved from https://docs.python.org/
    - Official documentation for Python programming language, providing guidance on Python syntax, libraries, and best practices.
- Vue.js. (n.d.). Vue.js Documentation.
    - Retrieved from https://vuejs.org/v2/guide/
    - Official documentation for Vue.js, offering guides and API references for Vue.js framework, which you are using for frontend development.
- Mitchell, N., & Docker, J. (2019). Docker Deep Dive. Berkely, CA: Apress.
    - This book provides an in-depth exploration of Docker concepts, architecture, and practical use cases, which can be helpful in understanding Docker for your project.
- Reitz, K., & Schlusser, T. (2018). The Docker Book: Containerization is the new virtualization. CreateSpace Independent Publishing Platform.
    - This book offers practical insights and examples for using Docker in various scenarios, which can be beneficial for your Docker-related tasks.
- Grinberg, M. (2014). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.
- John L. Viescas – SQL Queries for Mere Mortals – A Hands-On Guide to Data Manipulation in SQL. Addison- Wesley Publishing.
- Michael J Hernandez – Database Design for Mere Mortals – A Hands-On Guide to Relational Databases, Addison-Wesley
- Miguel Grinberg - Flask Web Development: Developing Web Applications with Python.
- Mark Ramm, Michael Bayer - "SQLAlchemy: Database Access Using Python, O'Reilly Media"
- Shalabh Aggarwal - "Flask Framework Cookbook, Packt Publishing"
- Eric Matthes - "Python Crash Course: A Hands-On, Project-Based Introduction to Programming, No Starch Press"
- Aidas Bendoraitis - "Web Development with Django Cookbook, Packt Publishing"
- Kenneth Reitz, Tanya Schlusser - "Flask Web Development, O'Reilly Media"
- Matt Makai - "Full Stack Python: Build, Deploy, and Operate Python-Based Applications, Full Stack Python"
- Daniel Gaspar, Jack Stouffer - "Flask By Example, Packt Publishing"
- Joseph Howse - "Mastering OpenCV 4 with Python: A practical guide covering topics from image processing, augmented reality to deep learning with OpenCV 4 and Python 3.7, Packt