Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	Διαδικτυακή Ιστοσελίδα Βιβλιοπωλείου Back-End Project API με ιατρικά βιβλία της νόσου Covid-19 Online Bookstore Website Back-End Project API with medical books dedicated on Covid-19 virus
Ονοματεπώνυμο Φοιτητή	Κωνσταντίνος Τσανεκλίδης
Πατρώνυμο	Χαράλαμπος
Αριθμός Μητρώου	ΜΠΠΛ 18064
Επιβλέπων	Ευθύμιος Αλέπης Αναπληρωτής Καθηγητής

Ιούλιος 2022

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Ευθύμιος Αλέπης Αναπληρωτής Καθηγητής	Μαρία Βίρβου Καθηγήτρια	Ευάγγελος Σακκόπουλος Αναπληρωτής Καθηγητής

Περιεχόμενα

1. Περίληψη

1.1 Περίληψη

Στη παρούσα μεταπτυχιακή διατριβή, παρουσιάζεται μια διαδικτυακή ιστοσελίδα βιβλιοπωλείου Back-End Project API με ιατρικά βιβλία της νόσου Covid-19. Μέσω της ιστοσελίδας αυτής, ο διαχειριστής δύναται να εποπτεύει και να διαμορφώνει σε πραγματικό χρόνο τις κατηγορίες των βιβλίων, τους συγγραφείς, όλες τις σχετικές πληροφορίες καθώς επίσης παρέχεται και η δυνατότητα στους εγγεγραμμένους χρήστες-αναγνώστες να κάνουν έγκυρες αξιολογήσεις.
Επεξηγείται η χρησιμοποιούμενη τεχνολογία (ASP Net Core) η οποία βασίστηκε στην αρχιτεκτονική του MVC. Μελλοντικές αναβαθμίσεις και επεκτάσεις της εφαρμογής αναφέρονται.

1.2. Abstract

The dissertation introduces an online bookstore website Back-End Project API with medical books dedicated on COVID-19 virus. Through this website, the Administrator can supervise and shape real-time categories of books, authors, all relevant information as well as the ability of registered users-readers to make valid evaluations.
The technology used (ASP.Net Core) based on the MVC architecture is explained. Future updates and extensions to the application are listed.

2. Εισαγωγή

2.1 Σκοπός

Σκοπός της παρούσας εργασίας είναι η δημιουργία μίας Διαδικτυακής ιστοσελίδας βιβλιοπωλείου Back-End Project API με ιατρικά βιβλία της νόσου Covid-19. Ο Γενικός Χρήστης έχει τη δυνατότητα να ενημερώνεται για όλες τις πληροφορίες σχετικά με το βιβλίο που επιθυμεί (τίτλος, συγγραφέας, πληροφορίες για τον συγγραφέα, αξιολόγηση βιβλίων από προηγούμενους χρήστες).

Ο Διαχειριστής θα έχει τη δυνατότητα να ενημερώνει-να αλλάζει όλες τις προηγούμενες πληροφορίες σε πραγματικό χρόνο μέσα στη βάση δεδομένων (C.R.U.D.).

2.2 Τεχνολογίες

Για την ανάπτυξη του συστήματος χρησιμοποιήθηκε η γλώσσα προγραμματισμού C#, με τεχνολογία (ASP Dot Net Core 6.05) η οποία βασίστηκε στην αρχιτεκτονική του MVC και τεχνολογίες του MVC. Για την δημιουργία της βάσης δεδομένων χρησιμοποιήθηκε η MySQL.

3. Ανάλυση Απαιτήσεων

3.1 Παρατίθενται οι λειτουργικές απαιτήσεις του συστήματος:

Για κάθε νέο χρήστη, ο γενικός διαχειριστής ενδιαφέρεται για:

- Ηλεκτρονική διεύθυνση χρήστη (Email)
- Κωδικό πρόσβασης (password),

Ο Διαχειριστής θα πρέπει να κάνει εγγραφή και ενημέρωση της βάσης δεδομένων για κάθε νέα δημοσίευση βιβλίου σχετικά με την νόσο Covid-19, με τις παρακάτω πληροφορίες:

- Τίτλος βιβλίου
- Συγγραφέας
- Πληροφορίες για τον συγγραφέα
- Κατηγορία βιβλίου
- Αξιολογήσεις-Σχόλια

3.2 Μη-Λειτουργικές Απαιτήσεις (ΜΛΑ)

Περιγράφουν ιδιότητες του συστήματος που συνήθως εκφράζονται βάσει των χαρακτηριστικών της μορφής:

- Απόδοση (Perfomance)
- Χρηστικότητα (Usability)
- Ασφάλεια (Security)
- Νομιμότητα (Legislative)
- Ιδιωτικότητα (Privacy)

Με άλλα λόγια περιγράφουν το πώς (ή το πόσο καλά) το σύστημα θα υποστηρίξει τις λειτουργικές απαιτήσεις.

Μπορούμε να θεωρήσουμε ως **«περιορισμούς»** που περιορίζουν τους τρόπους θα μπορούσαμε να πραγματώσουμε τις λειτουργικές απαιτήσεις**.** Επιγραμματικά κάποιες μη λειτουργικές απαιτήσεις είναι οι κάτωθι:

- Τα προσωπικά στοιχεία των χρηστών πρέπει να προστατεύονται.
- Μόνο ο διαχειριστής Administrator θα μπορεί να έχει πρόσβαση και να τροποποιεί (προσθέτει-αφαιρεί-ανανεώνει) την βάση δεδομένων του συστήματος.
- Το σύστημα πρέπει να λειτουργεί αδιάλειπτα.
- Το σύστημα θα πρέπει να είναι όσο το δυνατόν πιο φιλικό προς τον χρήστη.
- Να πραγματοποιείται έλεγχος για την ορθότητα των δεδομένων που εισάγουν οι χρήστες στην εφαρμογή και να εμφανίζεται το κατάλληλο μήνυμα ενημέρωσης προς τους χρήστες.

4. UML – Διαγράμματα περιπτώσεων χρήσης

4.1 Περιγραφή

Η UML, είναι μια γλώσσα μοντελοποίησης που μπορεί να χρησιμοποιηθεί τόσο από τον άνθρωπο όσο και από τις μηχανές. Χρησιμοποιείται για τον προσδιορισμό, την οπτικοποίηση με γραφικά σύμβολα καθώς και για την κατασκευή ενός συστήματος λογισμικού. Όπως θα δούμε παρακάτω, περιέχει ένα σύνολο διαγραμμάτων, τα οποία χρησιμοποιούνται για την περιγραφή των προδιαγραφών του λογισμικού που θέλουμε να υλοποιήσουμε.

Επίσης, επιτρέπει τη μοντελοποίηση των συστημάτων με βάση τις αρχές των αντικειμενοστραφών μοντέλων και χρησιμοποιεί κυρίως διαγράμματα για να εκφράσει αυτή την αντικειμενοστραφή ανάλυση και σχεδίαση έργων λογισμικού. Άλλο ένα πολύ σημαντικό σημείο που πρέπει να τονιστεί είναι ότι η γλώσσα αυτή είναι ανεξάρτητη της μεθοδολογίας μοντελοποίησης.

Η UML εφαρμόζεται για τη λύση object-oriented (OO) προβλημάτων. Για την επίλυση τέτοιου είδους προβλημάτων, θα πρέπει πρώτα να κατασκευαστεί κάποιο μοντέλο. Τα μοντέλα αποτελούνται από αντικείμενα που αλληλεπιδρούν στέλνοντας μηνύματα μεταξύ τους.

Τα αντικείμενα μπορούν να θεωρηθούν ως ζωντανοί οργανισμοί που έχουν χαρακτηριστικά γνωρίσματα, μπορούν να επιτελέσουν διάφορες λειτουργίες και μπορούν να έχουν διαφορετικές συμπεριφορές. Οι τιμές των

χαρακτηριστικών γνωρισμάτων των αντικειμένων καθορίζουν και την κατάστασή τους. Όσον αφορά τις κλάσεις, αυτές μπορούν να συμπεριλάβουν τα χαρακτηριστικά (ως δεδομένα) και τις συμπεριφορές (ως μεθόδους και λειτουργίες) σε μία ενιαία κι ευδιάκριτη οντότητα. Τα αντικείμενα αποτελούν στιγμιότυπα των κλάσεων.

4.2 Πεδίο χρήσης

Όπως προαναφέραμε, UML εφαρμόζεται για τη λύση object-oriented (OO) προβλημάτων. Για την επίλυση τέτοιου είδους προβλημάτων, θα πρέπει πρώτα να κατασκευαστεί κάποιο μοντέλο. Τα μοντέλα αποτελούνται από αντικείμενα που αλληλεπιδρούν στέλνοντας μηνύματα μεταξύ τους.
Επίσης, επιτρέπει τη μοντελοποίηση των συστημάτων με βάση τις αρχές των αντικειμενοστραφών μοντέλων και χρησιμοποιεί κυρίως διαγράμματα για να εκφράσει αυτή την αντικειμενοστραφή ανάλυση και σχεδίαση έργων λογισμικού.

Υπάρχουν διάφοροι τομείς όπου χρησιμοποιείται η γλώσσα αυτή, μερικοί από τους οποίους αναφέρονται παρακάτω:

- Οι consultancy εταιρείες, οι οποίες δίνουν λύσεις στον τομέα της βιομηχανίας, παρέχοντας συμβουλευτικές υπηρεσίες χρησιμοποιώντας τη UML ως εργαλείο μοντελοποίησης. Μεγάλες εταιρείες είναι η Capgemini, KPMG, Accenture, PA Consulting, Boston Consulting.
- Οι system engineers για να κάνουν capture τα απαιτούμενα ενός project.
- Οι software και hardware engineers, οι οποίοι χρησιμοποιούν τη UML σαν οδηγό για development μοντέλων και μεθοδολογιών.
- Οι μάνατζερ, στους οποίους η UML παρέχει ένα τρόπο να μοντελοποιήσουν τις στρατηγικές τους.

4.3 Βασικές έννοιες

Στη UML 2 υπάρχουν δύο βασικές κατηγορίες διαγραμμάτων: τα διαγράμματα δομής και τα διαγράμματα συμπεριφοράς. Τα διαγράμματα δομής μας δείχνουν τη στατική δομή του συστήματος που μοντελοποιείται. Αντίστοιχα, τα διαγράμματα συμπεριφοράς μας δείχνουν τη δυναμική συμπεριφορά μεταξύ των αντικειμένων του συστήματος. Στοιχεία που έχουν να κάνουν με τη δυναμική συμπεριφορά είναι οι μέθοδοι, οι συνεργασίες και οι δραστηριότητες των αντικειμένων. Με άλλα λόγια, το δυναμικό μοντέλο μας δείχνει τον τρόπο με τον οποίο αποκρίνεται το σύστημα στις ενέργειες των χρηστών ή σε άλλα εξωτερικά ερεθίσματα και πώς διαμορφώνεται η εσωτερική του κατάσταση κατά τη λειτουργία του. Επίσης, η δυναμική συμπεριφορά απεικονίζει την αλληλεπίδραση μεταξύ των αντικειμένων.

4.4 Διαγράμματα

Σε ανώτερο επίπεδο (Top level) έχουμε τα εξής διαγράμματα περιπτώσεων χρήσης (Use case):

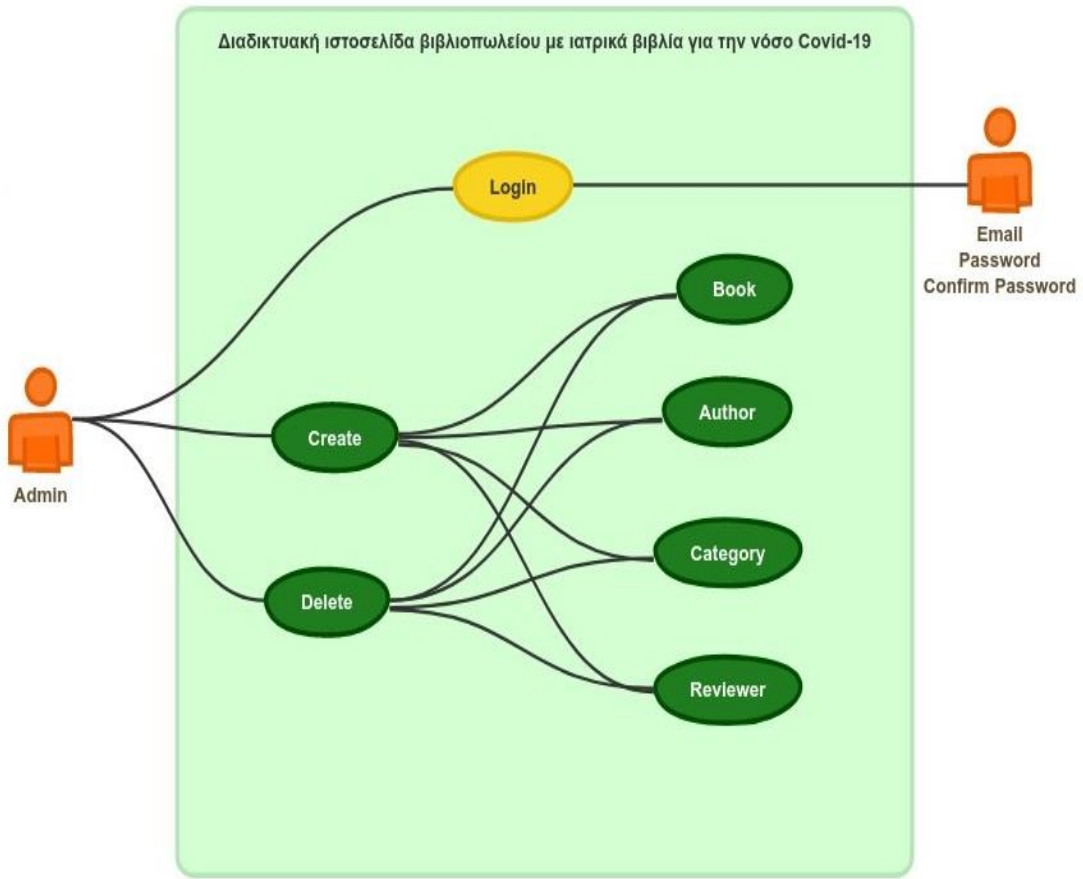
Σύνδεση με διαπιστευτήρια διαχειριστή

- LEVEL 1 - Admin:

- Register: Είσοδος χρήστη με Email, Password, Confirm Password.
- (Login) Σύνδεση του χρήστη (Admin).

Ο Admin μπορεί να εκτελέσει τις ενέργειες CREATE, DELETE και UPDATE σε:

- Book
- Author
- Category
- Reviewer



Στο LEVEL 2 έχουμε την σύνδεση οποιουδήποτε χρήστη πλην του Admin.

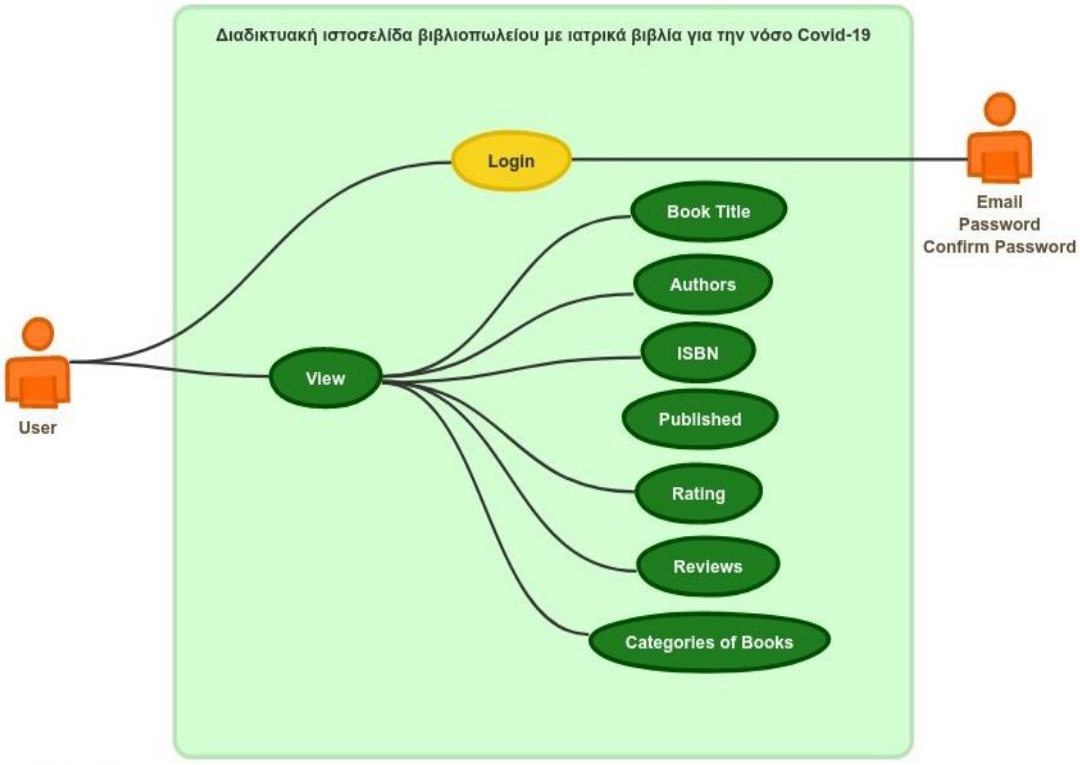
Η περίπτωση χρήσης του LEVEL 2 φαίνεται στο παρακάτω διάγραμμα:

- LEVEL 2 - User:

- Register: Είσοδος χρήστη με Email, Password, Confirm Password
- (Login) Σύνδεση του χρήστη (User)

Ο user μπορεί να εκτελέσει μόνο την ενέργεια του View σε:

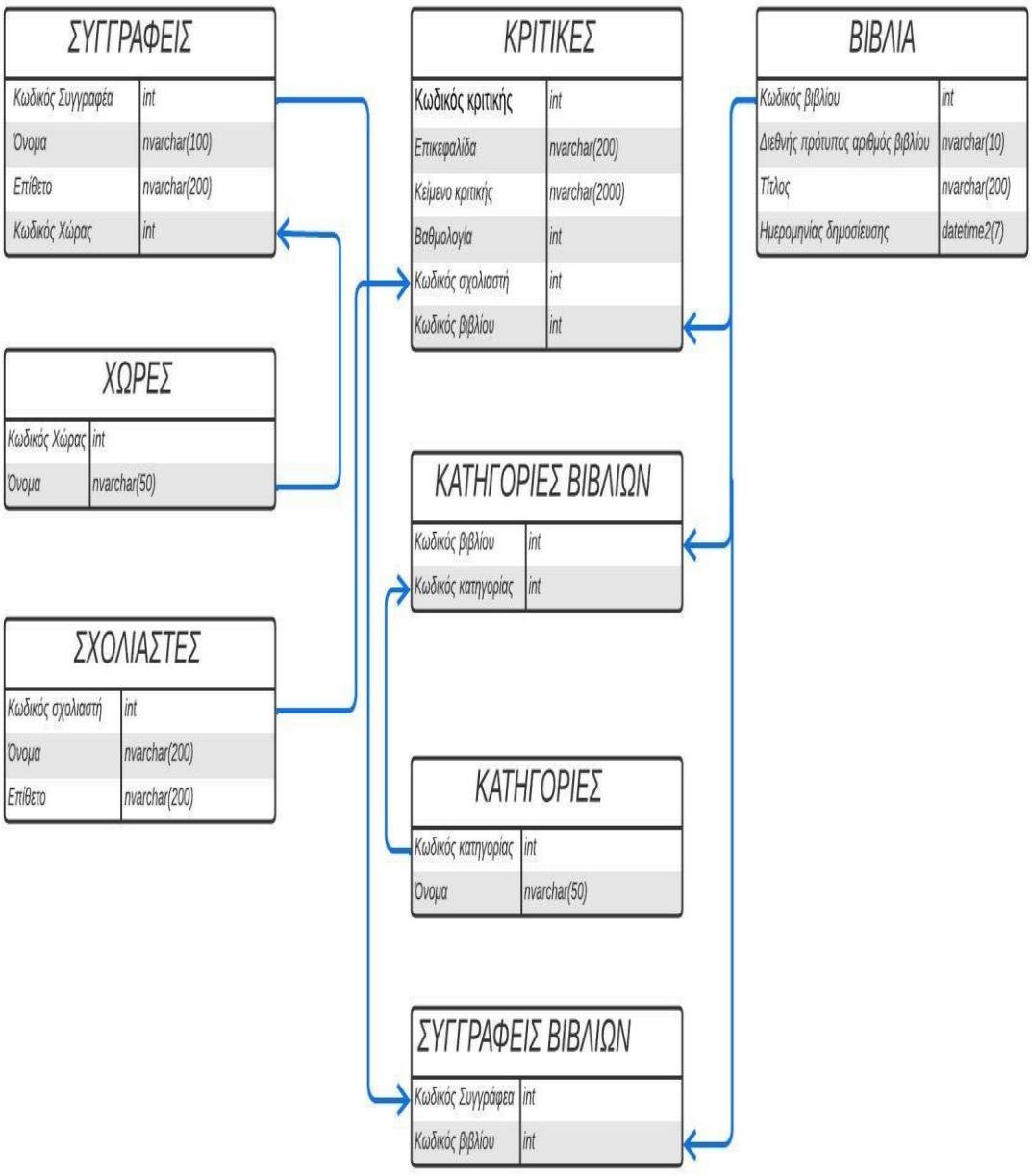
- Book Title
- Authors
- ISBN
- Published
- Rating
- Categories of Book
- Reviews

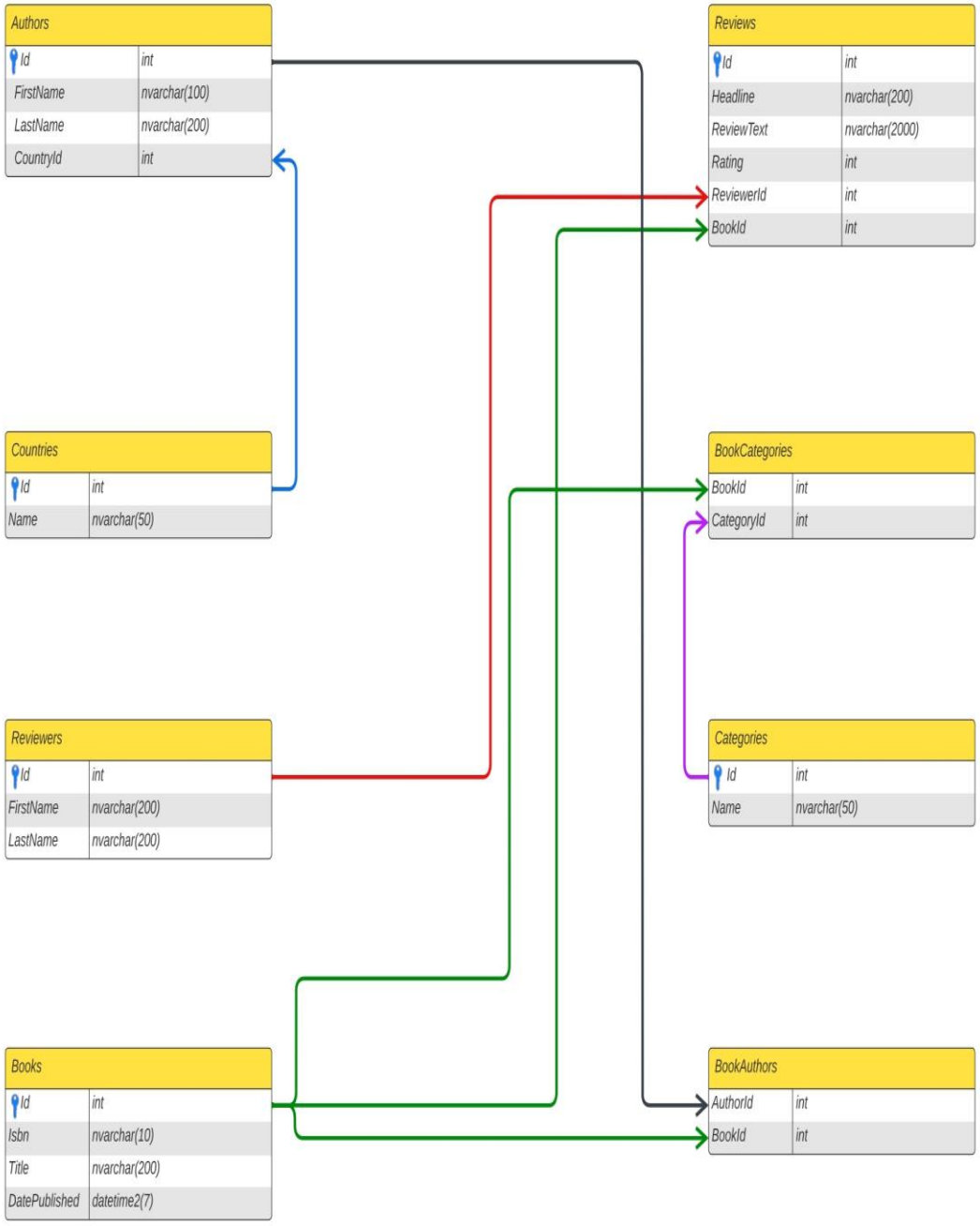


5. Βάση Δεδομένων

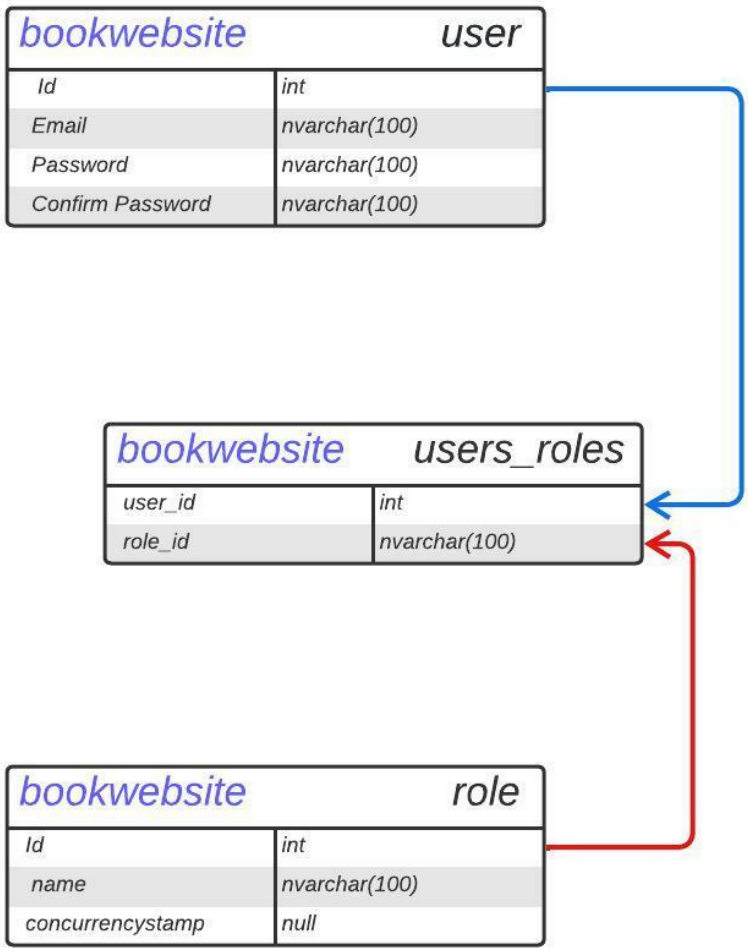
5.1 Διάγραμμα πινάκων βάσης

Συνεπώς, οι προς δημιουργία πίνακες στην βάση συνοπτικά όπως προέκυψαν από τον εννοιολογικό και λογικό σχεδιασμό, είναι όπως το κάτωθι διάγραμμα.




Authors
Id int
FirstName nvarchar(100)
LastName nvarchar(200)
CountryId int
Reviews
Id int
Headline nvarchar(200)
ReviewText nvarchar(2000)
Rating int
ReviewerId int
BookId int
Countries
Id int
Name nvarchar(50)
BookCategories
BookId int
CategoryId int
Reviewers
Id int
FirstName nvarchar(200)
LastName nvarchar(200)
Categories
Id int
Name nvarchar(50)
Books
Id int
Isbn nvarchar(10)
Title nvarchar(200)
DatePublished datetime2(7)
BookAuthors
AuthorId int
BookId int

Για την αυθεντικοποίηση και εξουσιοδότηση συνδεδεμένων χρηστών στην εφαρμογή σχεδιάστηκε το κάτωθι σχεσιακό μοντέλο. Οι χρήστες αποθηκεύονται στη βάση δεδομένων. Το πεδίο *role_id* του πίνακα *users_roles* αντιστοιχεί στα δικαιώματα πρόσβασης στην εφαρμογή (δικαιώματα διαχειριστή ή όχι), λειτουργία που εξυπηρετεί την εξουσιοδότηση χρηστών.



Για τον σχεδιασμό των οντοτήτων-διαγραμμάτων της βάσης, αξιοποιήθηκε μέσω του προγράμματος (Lucidchart).

6. Προγραμματιστικό περιβάλλον

6.1 Γλώσσα προγραμματισμού C#.

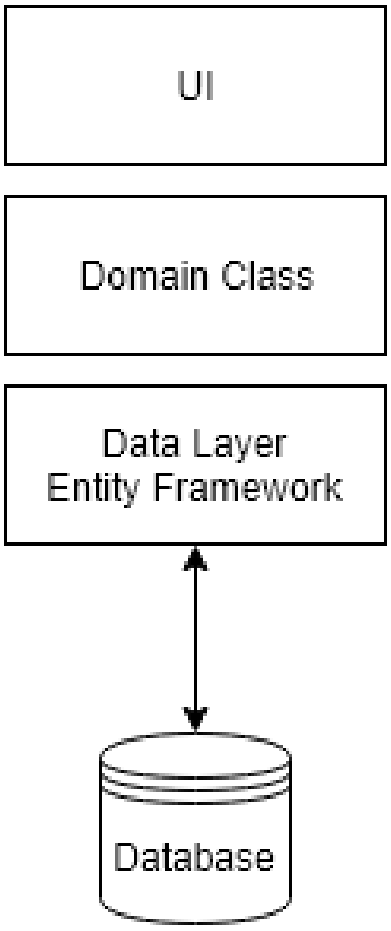
Για την πραγματοποίηση του κώδικα της παρούσας διπλωματικής εργασίας επιλέχθηκε η γλώσσα προγραμματισμού C#, καθώς χρησιμοποιείται για την ανάπτυξη διαδικτυακών ιστοσελίδων. Η C# δημιουργήθηκε από τη Microsoft και κυκλοφόρησε το 2001. Είναι μια γλώσσα προγραμματισμού γενικής χρήσης με αντικειμενοστρεφή κώδικα. Τα πλεονεκτήματά της είναι :

- γρήγορος χρόνος σύνταξης,
- γρήγορος χρόνος εκτέλεσης,
- αποτελεί κώδικας ασφαλείας,
- έχει πρόσβαση μόνο στη θέση μνήμης και εχει άδεια εκτέλεσης. Επομένως βελτιώνει την ασφάλεια του προγράμματος,
- είναι η πιο ισχυρή γλώσσα προγραμματισμού για .NET Framework,
- διαθέτει πλούσια βιβλιοθήκη.

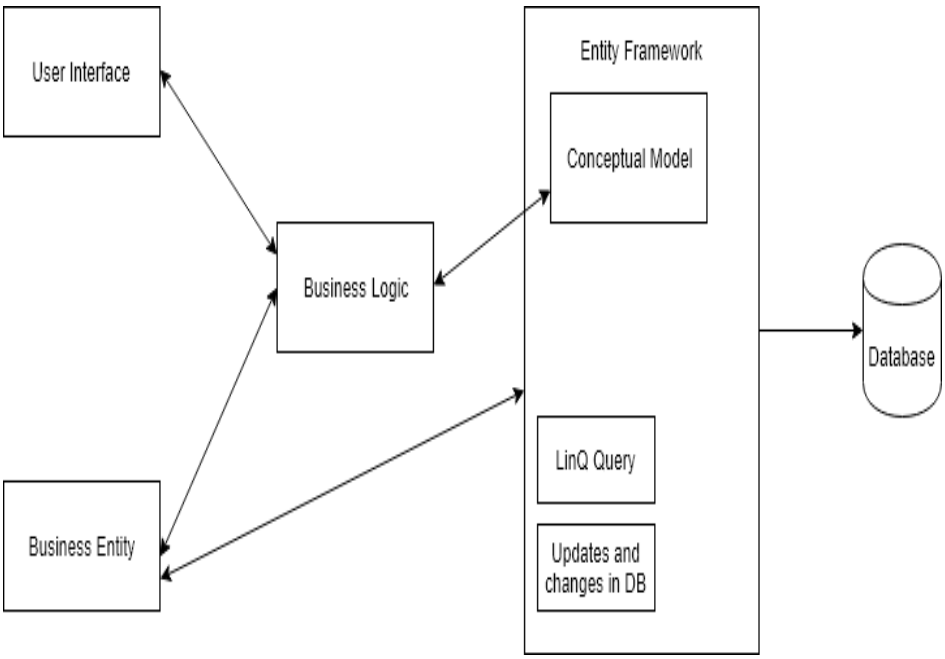
6.2 Entity Framework

Το Framework Entity είναι ένα πλαίσιο αντικειμενικής χρήσης ανοιχτού κώδικα για εφαρμογές .NET που υποστηρίζονται από τη Microsoft. Αυξάνει την παραγωγικότητα του προγραμματιστή, καθώς επιτρέπει στους προγραμματιστές να συνεργάζονται με δεδομένα χρησιμοποιώντας αντικείμενα κατηγοριών ειδικών για τον τομέα χωρίς να επικεντρώνονται στους υποκείμενους πίνακες βάσης δεδομένων και στις στήλες όπου αποθηκεύονται αυτά τα δεδομένα. Εξαλείφει την ανάγκη για το μεγαλύτερο μέρος του κώδικα πρόσβασης δεδομένων που χρησιμοποιείται για να αλληλεπιδράσει με τη βάση δεδομένων που οι προγραμματιστές συνήθως πρέπει να γράφουν. Μειώνει επίσης το μέγεθος του κώδικα των εφαρμογών ειδικών δεδομένων και επίσης η αναγνωσιμότητα του κώδικα αυξάνεται χρησιμοποιώντας το.

Το ακόλουθο σχήμα περιγράφει πού υπάρχει το πλαίσιο οντότητας στην εφαρμογή.



Το παραπάνω σχήμα αντιπροσωπεύει τον τρόπο με τον οποίο ένα πλαίσιο οντότητας αλληλεπιδρά με την κλάση και τη βάση δεδομένων. Παρέχει μια σύνδεση μεταξύ της επιχειρηματικής οντότητας και των πινάκων δεδομένων στη βάση δεδομένων. Εξοικονομεί δεδομένα αποθηκευμένα στις ιδιότητες των οντοτήτων και επίσης ανακτά δεδομένα από τη βάση δεδομένων και τα μετατρέπει αυτόματα σε επιχειρηματικές οντότητες. Το Entity Framework θα εκτελέσει το σχετικό ερώτημα στη βάση δεδομένων και στη συνέχεια θα υλοποιήσει τα αποτελέσματα σε περιπτώσεις των αντικειμένων.



Πλεονεκτήματα χρήσης τεχνολογίας Entity Framework

- Είναι ανεξάρτητη από την πλατφόρμα.
- Χρησιμοποιεί ερωτήματα LINQ για να χειριστεί τα δεδομένα στη βάση δεδομένων αντί για ερωτήματα SQL.
- Διατηρεί την τροχιά των τιμών που έχουν αλλάξει τις ιδιότητες των οντοτήτων.
- Εξοικονομεί επίσης αλλαγές που γίνονται Εισαγωγή, Διαγραφή ή Ενημέρωση Λειτουργίες.
- Επίσης, χειρίζεται ταυτόχρονα, έτσι ώστε τα δεδομένα να παρακάμπτουν από έναν χρήστη και να αντικατοπτρίζουν όταν η άλλη χρήση το θα το φέρει.
- Παρέχει προσωρινή αποθήκευση που σημαίνει ότι αποθηκεύει το αποτέλεσμα των συχνά χρησιμοποιούμενων ερωτημάτων.
- Ακολουθεί επίσης ορισμένες συμβάσεις για τον προγραμματισμό, ώστε να διαμορφώσει από προεπιλογή το μοντέλο EF.
- Επιτρέπει επίσης τη διαμόρφωση του μοντέλου EF από ένα άπταιστα API για να παρακάμψει την προεπιλεγμένη σύμβαση.
- Εάν κάνατε οποιεσδήποτε αλλαγές στο σχήμα βάσης δεδομένων τότε μπορείτε να αντικατοπτρίσετε αυτές τις αλλαγές στο μοντέλο EF γράφοντας migration command in CLI (Command Line Interface).
- Υποστηρίζει επίσης την αποθηκευμένη διαδικασία.
- Υποστηρίζει επίσης παραμετροποιημένα queries.

6.3 Consume CRUD(create,read,update,delete) API with ASP.NET Core 2.2 MVC using HttpClient

Η ιστοσελίδα δημιουργήθηκε με την τεχνολογία ASP.NET Core το οποίο βασίστηκε στην αρχιτεκτονική του M.V.C.

6.3.1 Τεχνολογία ASP.NET Core

Είναι ένα framework ανοικτού λογισμικού, το χρησιμοποιείται για web εφαρμογές που μπορούν να αναπτυχθούν και να τρέξουν windows, Linux και Mac.

Πλεονεκτήματα χρήσης ASP.NET Core:

- Καλύτερη απόδοση

Η συγκεκριμένη τεχνολογία αναβαθμίζεται συνεχώς με νέα χαρακτηριστικά, ο κώδικας βελτιώνεται και η απόδοση γίνεται ολοένα και καλύτερη. Ταυτόχρονα, έχεις τη δυνατότητα να δημιουργήσεις μια εφαρμογή σήμερα, η οποία θα μπορεί να αναβαθμιστεί στο μέλλον, χωρίς να χρειαστεί να επέμβεις ή να αλλάξεις τον κώδικα.

- Μεγαλύτερη ευελιξία

Όταν ασχολείσαι με το web application development είναι σημαντικό να εξασφαλίσεις ότι η εφαρμογή σου υποστηρίζεται και λειτουργεί το ίδιο αποτελεσματικά σε κάθε λειτουργικό σύστημα. Το ASP.NET Core υποστηρίζει το cross-platform development, που σημαίνει ότι μπορείς να δημιουργήσεις μια εφαρμογή για iOS και στη συνέχεια να χρησιμοποιήσεις τον ίδιο κώδικα για να δημιουργήσεις και μια Android εφαρμογή.

- Βελτιωμένη ποιότητα

Η εξέλιξη της τεχνολογίας απαιτεί την μειωμένη χρήση κώδικα (less coding), που σημαίνει ότι οι διαδικασίες αυτοματοποιούνται, κι εσύ έχεις ακόμη περισσότερο χρόνο για να αναπτύσσεις νέες εφαρμογές, να διαχειρίζεσαι εύκολα και να συντηρείς αποτελεσματικά ότι έχεις δημιουργήσει έως τώρα.

6.4 M.V.C. αρχιτεκτονική

Η αρχιτεκτονική του **MVC** περιλαμβάνει:
Model:

- interactions with database (SELECT, INSERT, UPDATE, DELETE), (σημείωση: βάση δεδομένων φτιάχνεται στο model)
- Δεδομένα που σχετίζονται με την λογική
- Επικοινωνούν με τους controllers
- Μπορεί μερικές φορές να αναβαθμίσει το View (εξαρτάται από το framework)

View:

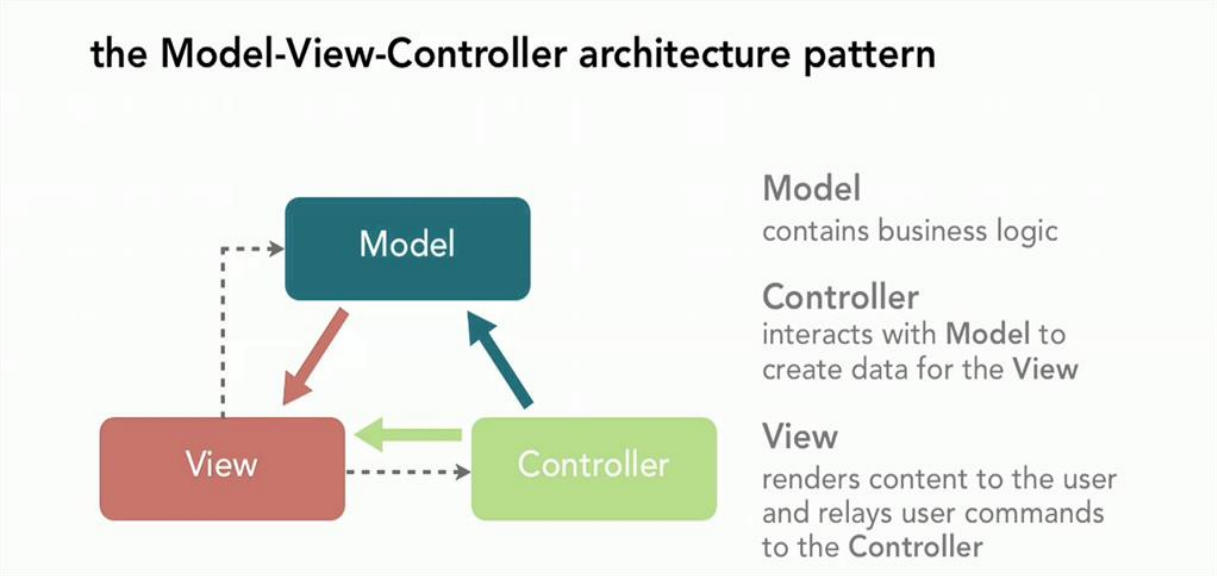
- Τι βλέπει ο τελικός χρήστης (end user) (UI: user Interface)
- Συνήθως περιλαμβάνει HTML/CSS
- Επικοινωνεί με τον controller
- Μπορεί να περάσει δυναμικές τιμές από τον controller
- Μηχανές προτύπων.

Controller: (διακινητής-διαμεσολαβητής end user με το model data)

- Λαμβάνει είσοδο (από View, url), λαμβάνει δεδομένα (data).
- Παίρνει δεδομένα (data) από το Model
- Περνάει δεδομένα (data) στο View.

Ο κώδικας της εφαρμογής χωρίζεται σε:

- Models (μοντέλα),
- Views (όψεις),
- Controllers (ελεγκτές)



Το πρότυπο αυτό βοηθά την διατήρηση ενός οργανωμένου κώδικα έτσι ώστε να είναι πιο κατανοητός και πιο εύκολα συντηρούμενος.

Το **model** είναι το τμήμα που διαχειρίζεται λειτουργίες πάνω σε δεδομένα της βάσης (π.χ. CRUD function)
Το **view** έχει να κάνει με το τι φαίνεται στο χρήστη, δηλαδή οι σελίδες της εφαρμογής και ο controller που βρίσκεται ενδιάμεσα.
Ο **controller** διαχειρίζεται τις ενέργειες του χρήστη, τις αναγνωρίζει και τις μεταφέρει στο κατάλληλο σημείο του κώδικα όπου θα επιτελέσει μια συγκεκριμένη λειτουργία. Επίσης, δέχεται τα δεδομένα που προκύπτουν από τα models και τα μεταφέρει στο κατάλληλο view.

6.5 Authentication – Authorization

AUTHENTICATION

Στην ιστοσέλιδα της εφαρμογής υλοποιήθηκε το κομμάτι του Authentication – Authorization.
Βήμα 1ο :
Στην αρχή δημιουργήσαμε μέσα στη βάση δεδομένων τους users μέσα από τους πίνακες των models της βάσεως δεδομένων.
Βήμα 2ο :
Login (έλεγχος διαπιστευτηρίων).
Δημιουργία και έλεγχος users με βάση τον έλεγχο διαπιστευτηρίων των users.
Βήμα 3ο :
Session cookies
Δημιουργήθηκαν τα sessions στο αντίστοιχο πεδίο μέσα στη βάση δεδομένων ώστε να πραγματοποιείται ο έλεγχος (μπρος-πίσω) των cookies στον browser (logged in…user).

AUTHORIZATION (Rules & Permissions)

Με βάση τον έλεγχο των διαπιστευτηρίων που έχουν γίνει από του εγγεγραμμένους χρήστες στη βάση δεδομένων ελέγχει το ρόλο τους (user, admin) και ανάλογα το ρόλο τον οποίο έχουν, τους δίνει τις αντίστοιχες εξουσιοδοτήσεις (π.χ. ο χρήστης που έχει συνδεθεί ως user στην εφαρμογή, μπορεί μόνο να του εμφανίζει σχετικές πληροφορίες για βιβλία, κατηγορίες βιβλίων, συγγραφείς, σχόλια χρηστών ενώ ο χρήστης που έχει συνδεθεί ως admin μπορεί να χρησιμοποιήσει την λειτουργία του C.R.U.D. δηλαδή να μπορεί να κάνει create-read-update-delete σε όλες τις κατηγορίες της εφαρμογής).

6.6 HTML-Bootstrap

Σχετικό με το γραφικό περιβάλλον της εφαρμογής (GUI) χρησιμοποιήθηκε η γλώσσα HTML και το Bootstrap Framework.

Η HTML είναι το ακρωνύμιο των λέξεων HyperText Markup Language, δηλαδή Γλώσσα Χαρακτηρισμού Υπερ-Κειμένου και βασίζεται στη γλώσσα SGML, Standard Generalized Markup Language, που είναι ένα πολύ μεγαλύτερο σύστημα επεξεργασία ς εγγράφων και είναι η βασική γλώσσα με την οποία πραγματοποιείται η δόμηση σελίδων του Παγκόσμιου Ιστού. Οι κυριότερες ετικέτες που έχουν χρησιμοποιηθεί στην ιστοσελίδα της εφαρμογής είναι:

Ετικέτα	Περιγραφή
<html>...</html>	Ορίζει την αρχή και το τέλος μιας ιστοσελίδας.
<head>...</head>	Ορίζει το τμήμα εκείνο της ιστοσελίδας στο οποίο αναφέρονται διαχειριστικής φύσεως πληροφορίες που αφορούν στο περιεχόμενο της ιστοσελίδας. Οι πληροφορίες αυτές δεν εμφανίζονται από τον φυλλομετρητή
<body>...</body>	Ορίζει το περιεχόμενο της ιστοσελίδας.
<title>...</title>	Ορίζει τον τίτλο της.
...	Ορίζει παράγραφο.
 	Δηλώνει αλλαγή γραμμής.
<img>	Ορίζει την εισαγωγή κάποιας εικόνας -image- και των παραμέτρων που αφορούν στη θέση της, το μέγεθός της, κ.ά.
...	Ορίζει δεσμό με ιστοσελίδα που βρίσκεται στο URL.

Το Bootstrap είναι σήμερα το πιο δημοφιλές πλαίσιο ιστού για την ανάπτυξη των εφαρμογών ιστού που ανταποκρίνονται. Προσφέρει μια σειρά από χαρακτηριστικά και οφέλη που μπορούν να βελτιώσουν την εμπειρία των χρηστών σας με τον ιστότοπό σας, είτε είστε αρχάριος στο σχεδιασμό και την ανάπτυξη του front-end είτε σε έναν εμπειρογνώμονα. Το Bootstrap αναπτύσσεται ως σύνολο αρχείων CSS και JavaScript και έχει σχεδιαστεί για να βοηθήσει τον ιστότοπό σας ή την κλίμακα εφαρμογής σας αποτελεσματικά από τηλέφωνα σε tablet σε επιτραπέζιους υπολογιστές.

7. API Πηγαίος Κώδικας M.V.C.

Ένα API (Application Programm Interface) είναι ένα σύνολο από συναρτήσεις, πρωτόκολλα και εργαλεία που χρησιμοποιούνται για την ανάπτυξη εφαρμογών λογισμικού. Ορίζει ένα μέρος λογισμικού ή εφαρμογής και περιγράφει τον τρόπο με τον οποίο τα συστατικά μέρη του αλληλεπιδρούν με το εξωτερικό περιβάλλον. Τα APIs χρησιμοποιούνται από άλλες εφαρμογές, δίνοντας την δυνατότητα χρήσης της λειτουργίας που επιτελούν. Πολλές φορές έχουν την μορφή βιβλιοθήκης η οποία περιέχει τις κλάσεις, τις δομές δεδομένων και τις συναρτήσεις που υλοποιεί και ενσωματώνεται στο λογισμικό από το οποίο χρησιμοποιείται. Σε άλλες περιπτώσεις, ένα API ορίζεται ως υπηρεσία η οποία δέχεται και ικανοποιεί απομακρυσμένες κλήσεις, με τις πιο δημοφιλείς να είναι οι υπηρεσίες REST και SOAP. Οι υπηρεσίες ιστού (Web Services) ακολουθούν παρόμοια λογική με αυτήν των APIs, δηλαδή εκτελούν μια συγκεκριμένη λειτουργία η οποία μπορεί να χρησιμοποιηθεί και από άλλο λογισμικό ή εφαρμογή. Ένα web service είναι μια υπηρεσία που παρέχεται από μια ηλεκτρονική συσκευή ή λογισμικό σε μια άλλη ενώ αυτές επικοινωνούν μεταξύ τους μέσω του παγκόσμιου Ιστού. Για την επικοινωνία και την μεταφορά του περιεχομένου της υπηρεσίας χρησιμοποιείται μια τεχνολογία ιστού η οποία συνήθως είναι το πρωτόκολλο HTTP. Η απόκριση της υπηρεσίας στο ερώτημα της εφαρμογής που την χρησιμοποιεί, είναι μορφοποιημένη συνήθως σύμφωνα με κάποια γλώσσα μορφοποίησης, με τις

επικρατέστερες σε χρήση να είναι η XML και η JSON. Έτσι, το πρόγραμμα στέλνει ένα ερώτημα στον χρήστη μέσω του πρωτοκόλλου και η υπηρεσία απαντάει με ένα μορφοποιημένο κείμενο, το οποίο στην συνέχεια μπορεί να αναλυθεί από την εφαρμογή και να προκύψει η τελική απάντηση. Αξίζει να σημειωθεί πως και στην περίπτωση των APIs αλλά και σε αυτήν των υπηρεσιών ιστού, η χρήση συνήθως γίνεται με κάποιο κλειδί (API Key), το οποίο ουσιαστικά λειτουργεί ως αριθμός ταυτότητας για τις διαφορετικές εφαρμογές που χρησιμοποιούν την υπηρεσία και προφανώς είναι μοναδικό για κάθε μία από αυτές. Φυσικά υπάρχουν και υπηρεσίες που διανέμονται χωρίς την απαίτηση χρήσης κάποιου κλειδιού, συνήθως αυτές που είναι εντελώς δωρεάν. Η ενσωμάτωση και χρήση APIs και Web Services από λογισμικά και εφαρμογές είναι πλέον μια κοινή πρακτική ενώ οι υπηρεσίες που είναι διαθέσιμες καλύπτουν μια μεγάλη ποικιλία θεμάτων. Έτσι, γίνεται πιο προσιτή η δημιουργία εφαρμογών που συγκεντρώνουν πολλές υπηρεσίες σε μία και παράλληλα μειώνεται σε πολλές περιπτώσεις ο κόπος αλλά και ο χρόνος που απαιτούν οι εφαρμογές για να αναπτυχθούν.

7.1 Models

- Author

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Models
{
    public class Author
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        [Required]
        [MaxLength(100, ErrorMessage ="First Name cannot be more than 100 characters")]
        public string FirstName { get; set; }
                      sider
        [Required]
        [MaxLength(200, ErrorMessage = "Last Name cannot be more than 200 characters")]
        public string LastName { get; set; }
        public virtual Country Country { get; set; }
        public virtual ICollection<BookAuthor> BookAuthors { get; set; }
    }
}
```

- Book

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Models
{
    public class Book
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        [Required]
        [StringLength(10,MinimumLength =3, ErrorMessage ="ISBN must be between 3 and 10 characters")]
        public string Isbn { get; set; }

        [Required]
        [MaxLength(200, ErrorMessage = "Title cannot be more than 200 characters")]
        public string Title { get; set; }

        public DateTime? DatePublished { get; set; }
        public virtual ICollection<Review> Reviews { get; set; }
        public virtual ICollection<BookAuthor> BookAuthors { get; set; }
        public virtual ICollection<BookCategory> BookCategories { get; set; }
    }
}
```

- BookAuthor

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Models
{
    public class BookAuthor
    {
        public int BookId { get; set; }
        public Book Book { get; set; }

        public int AuthorId { get; set; }
        public Author Author { get; set; }
    }
}
```

- BookCategory

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Models
{
    public class BookCategory
    {
        public int BookId { get; set; }
        public Book Book { get; set; }

        public int CategoryId { get; set; }
        public Category Category { get; set; }
    }
}
```

- Category

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Models
{
    public class Category
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        [Required]
        [MaxLength(50, ErrorMessage = "Category must be up to 50 characters in length")]
        public string Name { get; set; }
        public virtual ICollection<BookCategory> BookCategories { get; set; }
    }
}
```

- Country

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Models
{
    public class Country
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        [Required]
        [MaxLength(50, ErrorMessage = "Country must be up to 50 characters in length")]
        public string Name { get; set; }
        public virtual ICollection<Author> Authors { get; set; }
    }
}
```

- Review

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Models
{
    public class Review
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        [Required]
        [StringLength(200, MinimumLength = 10, ErrorMessage = "Headline needs to be between 10 and 200 characters")]
        public string Headline { get; set; }

        [Required]
        [StringLength(2000, MinimumLength = 50, ErrorMessage = "Review needs to be between 50 and 2000 characters")]
        public string ReviewText { get; set; }

        [Required]
        [Range(1,5,ErrorMessage ="Rating must be between 1 ad 5 stars")]
        public int Rating { get; set; }
        public virtual Reviewer Reviewer { get; set; }
        public virtual Book Book { get; set; }
    }
}
```

- Reviewer

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Models
{
    public class Reviewer
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        [Required]
        [MaxLength(100, ErrorMessage = "First Name must be up to 100 characters in length")]
        public string FirstName { get; set; }

        [Required]
        [MaxLength(200, ErrorMessage = "Last Name must be up to 200 characters in length")]
        public string LastName { get; set; }
        public virtual ICollection<Review> Reviews { get; set; }
    }
}
```

7.2 Controllers

- AuthorsController

```
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class AuthorsController : Microsoft.AspNetCore.Mvc.Controller
    {
        private IAuthorRepository _authorRepository;
        private IBookRepository _bookRepository;
        private ICountryRepository _countryRepository;
        0 references
        public AuthorsController(IAuthorRepository authorRepository, IBookRepository bookRepository,
            ICountryRepository countryRepository)
        {
            _authorRepository = authorRepository;
            _bookRepository = bookRepository;
            _countryRepository = countryRepository;
        }
```

- BooksController

```
using BookApiProject.Dtos;
using BookApiProject.Models;
using BookApiProject.Services;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class BooksController : Microsoft.AspNetCore.Mvc.Controller
    {
        private IBookRepository _bookRepository;
        private IAuthorRepository _authorRepository;
        private ICategoryRepository _categoryRepository;
        private IReviewRepository _reviewRepository;

        0 references
        public BooksController(IBookRepository bookRepository, IAuthorRepository authorRepository,
            ICategoryRepository categoryRepository, IReviewRepository reviewRepository )
        {
            _bookRepository = bookRepository;
            _authorRepository = authorRepository;
            _categoryRepository = categoryRepository;
            _reviewRepository = reviewRepository;
        }
```

- CategoriesController

```
using BookApiProject.Dtos;
using BookApiProject.Models;
using BookApiProject.Services;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class CategoriesController : Microsoft.AspNetCore.Mvc.Controller
    {
        private ICategoryRepository _categoryRepository;
        private IBookRepository _bookRepository;

        0 references
        public CategoriesController(ICategoryRepository categoryRepository, IBookRepository bookRepository)
        {
            _categoryRepository = categoryRepository;
            _bookRepository = bookRepository;
        }
```

- CountriesController

```
using BookApiProject.Dtos;
using BookApiProject.Models;
using BookApiProject.Services;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class CountriesController : Microsoft.AspNetCore.Mvc.Controller
    {
        private ICountryRepository _countryRepository;
        private IAuthorRepository _authorRepository;

        0 references
        public CountriesController(ICountryRepository countryRepository, IAuthorRepository authorRepository)
        {
            _countryRepository = countryRepository;
            _authorRepository = authorRepository;
```

- ReviewersController

```
using BookApiProject.Dtos;
using BookApiProject.Models;
using BookApiProject.Services;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class ReviewersController : Microsoft.AspNetCore.Mvc.Controller
    {
        private IReviewerRepository _reviewerRepository;
        private IReviewRepository _reviewRepository;

        0 references
        public ReviewersController(IReviewerRepository reviewerRepository, IReviewRepository reviewRepository)
        {
            _reviewerRepository = reviewerRepository;
            _reviewRepository = reviewRepository;
        }
```

- ReviewsController

```
using BookApiProject.Dtos;
using BookApiProject.Models;
using BookApiProject.Services;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookApiProject.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class ReviewsController : Microsoft.AspNetCore.Mvc.Controller
    {
        private IReviewerRepository _reviewerRepository;
        private IReviewRepository _reviewRepository;
        private IBookRepository _bookRepository;

        0 references
        public ReviewsController(IReviewerRepository reviewerRepository, IReviewRepository reviewRepository, IBookRepository bookRepository)
        {
            _reviewerRepository = reviewerRepository;
            _reviewRepository = reviewRepository;
            _bookRepository = bookRepository;
        }
```

7.3 Services

- AuthorRepository

```
namespace BookApiProject.Services
{
    2 references
    public class AuthorRepository : IAuthorRepository
    {
        private BookDbContext _authorContext;

        0 references
        public AuthorRepository(BookDbContext authorContext)
        {
            _authorContext = authorContext;
        }

        7 references
        public bool AuthorExists(int authorId)
        {
            return _authorContext.Authors.Any(a => a.Id == authorId);
        }

        2 references
        public bool CreateAuthor(Author author)
        {
            _authorContext.Add(author);
            return Save();
        }

        2 references
        public bool DeleteAuthor(Author author)
        {
            _authorContext.Remove(author);
            return Save();
        }

        3 references
        public Author GetAuthor(int authorId)
        {
            return _authorContext.Authors.Where(a => a.Id == authorId).FirstOrDefault();
        }

        2 references
        public ICollection<Author> GetAuthors()
        {
            return _authorContext.Authors.OrderBy(a => a.LastName).ToList();
        }

        2 references
        public ICollection<Author> GetAuthorsOfABook(int bookId)
        {
            return _authorContext.BookAuthors.Where(b => b.Book.Id == bookId).Select(a => a.Author).ToList();
        }

        3 references
        public ICollection<Book> GetBooksByAuthor(int authorId)
        {
            return _authorContext.BookAuthors.Where(a => a.Author.Id == authorId).Select(b => b.Book).ToList();
        }

        4 references
        public bool Save()
        {
            var saved = _authorContext.SaveChanges();
            return saved >= 0 ? true : false;
        }

        2 references
        public bool UpdateAuthor(Author author)
        {
            _authorContext.Update(author);
            return Save();
        }
    }
}
```

Παρόμοια ολοκληρώθηκαν και οι υπόλοιπες οντότητες για τα Services του API project.

7.4 Interfaces

Ένα interface στη C# μπορεί να θεωρηθεί ως μια αφηρημένη κλάση, η οποία δε δηλώνει μεταβλητές και όλες οι μέθοδοί της είναι αφηρημένες. Τα interfaces, όπως και οι κλάσεις, ορίζουν ιεραρχίες κληρονομικότητας και είναι δυνατό να δηλωθούν μεταβλητές με τύπο αναφοράς σε αυτές. Με την έννοια αυτή, είναι ισοδύναμες με πλήρως αφηρημένες κλάσεις. Η κλάση αυτή υλοποιεί μια μέθοδο display, με την ίδια επικεφαλίδα που ορίζεται στο interface Displayable. Είναι επομένως δυνατό να θεωρηθεί ως μια υλοποίηση του interface και αυτό φαίνεται στον ορισμό του με τη δήλωση implements Displayable. Τα interfaces καλύπτουν την ανάγκη για πολλαπλή κληρονομικότητα, δηλαδή την επέκταση περισσότερων της μιας κλάσης, Μια κλάση μπορεί να επεκτείνει μόνο μια κλάση βάση, μπορεί όμως να υλοποιεί οποιονδήποτε αριθμό από interfaces.
Κάθε λειτουργία που αναμένεται να υλοποιηθεί από ένα σύστημα, θα πρέπει να υλοποιείται από τρία διαφορετικά συστατικά : τον Controller, το Service και το Repository. Σε αυτό το σημείο, η σύνδεση των τριών αυτών συστατικών μπορεί να πραγματοποιηθεί με την χρήση interfaces. Ο καθορισμός interfaces για την επικοινωνία μεταξύ των τριών αυτών συστατικών προσφέρει μεγάλη ευελιξία στο σύστημά.

Για παράδειγμα:

```
using BookApiProject.Dtos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BookGUI.Services
{
    public interface ICountryRepositoryGUI
    {
        IEnumerable<CountryDto> GetCountries();
        CountryDto GetCountryById(int countryId);
        CountryDto GetCountryOfAnAuthor(int authorId);
        IEnumerable<AuthorDto> GetAuthorsFromACountry(int countryId);
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
using BookApiProject.Dtos;

namespace BookGUI.Services
{
    public class CountryRepositoryGUI : ICountryRepositoryGUI
    {
        public IEnumerable<AuthorDto> GetAuthorsFromACountry(int countryId)
        {

            IEnumerable<AuthorDto> authors = new List<AuthorDto>();

            using (var client = new HttpClient())
            {
                client.BaseAddress = new Uri("http://localhost:60039/api/");

                var response = client.GetAsync($"countries/{countryId}/authors");
                response.Wait();

                var result = response.Result;

                if (result.IsSuccessStatusCode)
                {
                    var readTask = result.Content.ReadAsAsync<IList<AuthorDto>>();
                    readTask.Wait();

                    authors = readTask.Result;
                }
            }

            return authors;
```

8. Εκτέλεση εφαρμογής

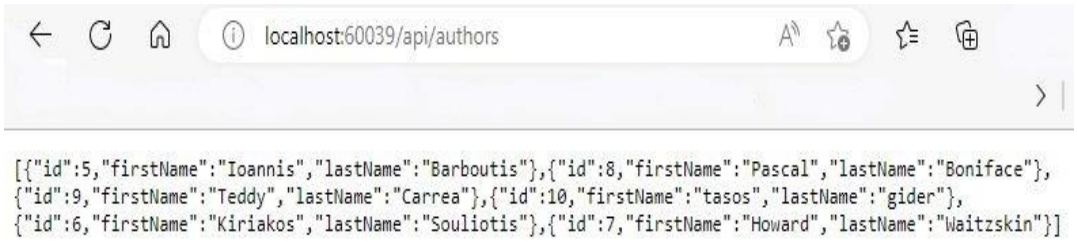
Το πρόγραμμα που χρησιμοποιήθηκε για την υλοποίηση της εφαρμογής είναι το Visual Studio 2022 6.2.0 των Windows και για το στήσιμο της βάσης δεδομένων χρησιμοποιήθηκε η MySQL

Συνημμένα τα αρχεία της βάσης δεδομένων στο αρχείο data.sql .
Για την εκτέλεση του προγράμματος της εφαρμογής θα πρέπει να πατήσουμε το «κλικ» στο runbutton του προγράμματος Visual Studio.

Ο σύνδεσμος για την πλοήγηση για το projectAPI: http://localhost:60039

9. Παρουσίαση εφαρμογής

1. Είσοδος στην εφαρμογή μέσω API.
   - Authors

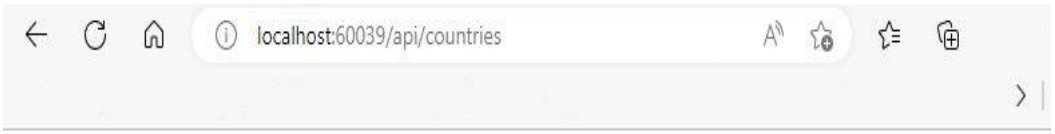


   - Books

- Categories

[{"id":1,"name":"Clinical Med"},{"id":2,"name":"Med"},{"id":3,"name":"Med and Pharmacy"},{"id":6,"name":"Medichines"},
{"id":4,"name":"Med-Programming"},{"id":7,"name":"Treatments"},{"id":5,"name":"Virus"}]

- Countries

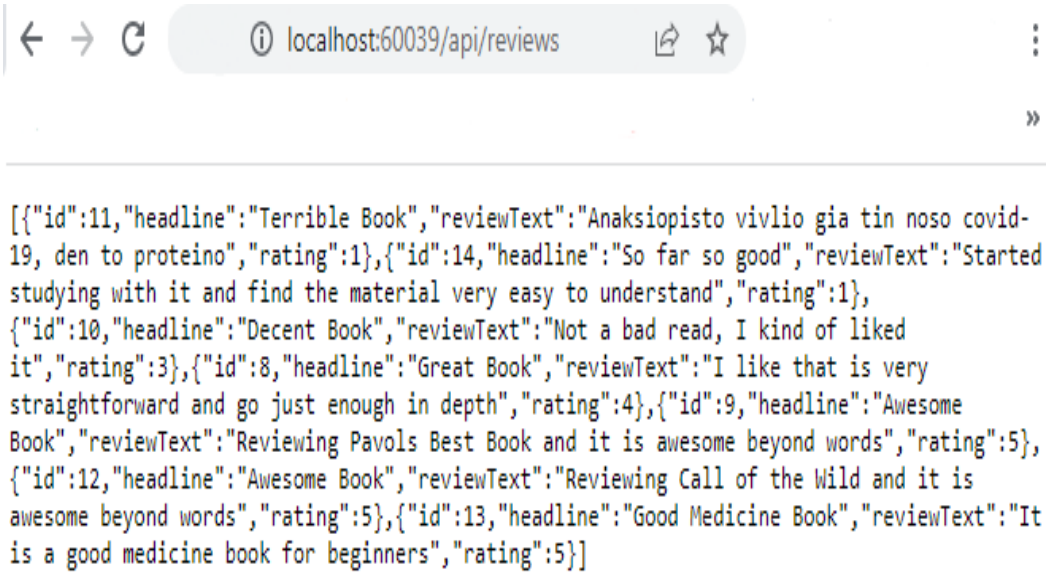


[{"id":5,"name":"England"},{"id":3,"name":"Germany"},{"id":1,"name":"Greece"},{"id":2,"name":"Greece"},
{"id":6,"name":"italy"},{"id":4,"name":"USA"}]

- Categories

[{"id":1,"name":"Clinical Med"},{"id":2,"name":"Med"},{"id":3,"name":"Med and Pharmacy"},{"id":6,"name":"Medichines"},
{"id":4,"name":"Med-Programming"},{"id":7,"name":"Treatments"},{"id":5,"name":"Virus"}]

- Reviews



- Reviewers

10. Συμπεράσματα - Επεκτάσεις

Η σωστή διαχείριση της ηλεκτρονικής ιστοσελίδας βιβλίων απαιτεί από τον διαχειριστή του συστήματος να είναι σε εγρήγορση ώστε να ενημερώνει συνεχώς την βάση δεδομένων και να την κρατάει επίκαιρη ώστε οι αναγνώστες και οι εγγεγραμμένοι χρήστες να έχουν μια έγκυρη ενημέρωση για όλα τα καινούργια διαθέσιμα βιβλία που πρωτοκυκλοφορούν στο εμπόριο.

Η παρούσα εφαρμογή θα μπορούσε να επεκταθεί και για άλλες παρόμοιες κατηγορίες βιβλίων αναφορικά με νόσους που σχετίζονται με τους κορωνοϊούς και θα μας απασχολούν τα επόμενα χρόνια. Επιπλέον ως μελλοντική επέκταση της εφαρμογής θα μπορούσε να αναφερθεί η υλοποίηση για κάποιο online chat στο οποίο θα μπορούσαν οι εγγεγραμμένοι χρήστες να αλληλεπιδρούν και να αλλάζουν τις απόψεις τους σε πραγματικό χρόνο.

Τέλος η συγκεκριμένη εφαρμογή θα μπορούσε να αναβαθμιστεί στο μέλλον για ένα πλήρες λειτουργικό ηλεκτρονικό ιστότοπο βιβλίων στον οποίο οι εγγεγραμμένοι χρήστες θα μπορούν ηλεκτρονικά με χρήση της χρεωστικής κάρτας τους να αγοράζουν οποιαδήποτε βιβλίο επιθυμούν, να ενημερώνονται για την διαθεσιμότητα του, σημείο παραλαβής και για όλες τις πληροφορίες σχετικά με το βιβλίο που επέλεξαν (π.χ συγγραφέας, κατηγορία, τιμή, σχόλια αναγνωστών κ.α).

11. Βιβλιογραφία

Διαδικτυακός τόπος:

- https://www.stackoverflow.com
- https://www.w3schools.com
- https://www.coursera.org
- https://www.udacity.com
- https://getbootstrap.com/docs/4.3/getting-started/download/