Σελίδα τίτλου

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΚΩΝ ΚΑΙ ΔΙΟΙΚΗΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΣΧΕΔΙΑΣΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΔΙΑΔΥΚΤΙΑΚΗΣ ΕΦΑΡΜΟΓΗΣ

Ιωάννης Ποντίκης

Επιβλέπων: Βαρτζιώτης Φώτιος

ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ

Τόπος έκδοσης, Μήνας, Έτος ολοκλήρωσης

DESIGN AND IMPLEMENTATION OF ONLINE APPLICATION

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 25/09/2024

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Φώτιος Βαρτζιώτης,

2. Μέλος επιτροπής

   Σπυριδούλα

   Μαργαρίτη,

3. Μέλος επιτροπής

   Ελευθέριος

   Στεργίου,

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα μεταπτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Επίθετο, Όνομα
Ποντίκης Ιωάννης

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή της πτυχιακής μου εργασίας κύριο Φώτιο Βαρτζιώτη που με βοήθησε να υλοποιήσω την εργασία.

ΠΕΡΙΛΗΨΗ

Ένας σημαντικός παράγοντας για την δημιουργία μιας εφαρμογής είναι οι web developers και μέσα από αυτούς έχουν δημιουργηθεί πολλές χρήσιμες εφαρμογές στην σημερινή εποχή μια χρήσιμη εφαρμογή είναι η δημοπρασία αυτοκινήτων όπου ο καθένας μπορεί να πουλήσει το δικό το όχημα χωρίς να χρειάζεται να έρθει σε επικοινωνία με κάποιον πωλητή οχημάτων.

Σκοπός αυτής της πτυχιακής εργασίας είναι η δημιουργία μιας εφαρμογής δημοπρασίας αυτοκινήτων που επιτρέπει στους χρήστες να δημιουργήσουν λογαριασμούς και να διαχειριστούν τις δικές τους δημοπρασίες ή να κάνουν προσφορές για την αγορά αυτοκινήτων από άλλους πωλητές.

Η εφαρμογή χρησιμοποιεί τεχνολογίες όπως C#, .NET, NextJS, Docker για τη δημιουργία και διαχείριση containers, ενώ επικοινωνεί μεταξύ των συστημάτων της χρησιμοποιώντας το RabbitMQ. Ως βάσεις δεδομένων χρησιμοποιείται η PostgreSQL και την MongoDB.

Επίσης, η εφαρμογή περιέχει τη δυνατότητα αναζήτησης αυτοκινήτων και την εμφάνιση δημοπρασιών με βάση τον χρόνο λήξης τους προσφέρουν στους χρήστες ευκαιρίες να παρακολουθούν τις δραστηριότητες στην πλατφόρμα με άνεση. Η λειτουργία ειδοποιήσεων σχετικά με τη λήξη δημοπρασιών και την αποδοχή των προσφορών τους ενισχύει την επικοινωνία και την εμπειρία τους στην πλατφόρμα. Επιπλέον, η χρήση του tokken για την αυθεντικότητα των χρηστών συμβάλλει στην ενίσχυση της ασφάλειας και της προστασίας των προσωπικών τους δεδομένων. Με όλα αυτά τα χαρακτηριστικά, η εφαρμογή παρέχει μια ολοκληρωμένη εμπειρία για την αγορά και πώληση αυτοκινήτων μέσω δημοπρασιών, ενώ εξασφαλίζει την ασφάλεια και την άνεση των χρηστών της.

Λέξεις-κλειδιά: **Σύστημα δημοπρασίας αυτοκινήτων, τεχνολογίες C#, .NET, NextJS, Docker, επικοινωνία RabbitMQ, βάσεις δεδομένων PostgreSQL και MongoDB, αναζήτηση αυτοκινήτων, ειδοποιήσεις λήξης δημοπρασιών, αυθεντικότητα μέσω token.**

ABSTRACT

An important factor in the creation of an application is the web developers and through them many useful applications have been created in today's time a useful application is the car auction where anyone can sell their own vehicle without having to communicate with someone vehicle salesman.

The purpose of this thesis is to create a car auction application that allows users to create accounts and manage their own auctions or bid on cars from other sellers.

The application uses technologies such as C#, .NET, NextJS, Docker to create and manage containers, while communicating between its systems using RabbitMQ. PostgreSQL and MongoDB are used as databases.

Also, the application contains the possibility to search for cars and the display of auctions based on their expiration time offer users opportunities to monitor the activities on the platform with comfort. Enabling notifications about auctions ending and accepting their bids enhances their communication and experience on the platform. In addition, the use of tokken to authenticate users helps to enhance the security and protection of their personal data. With all these features, the app provides a comprehensive experience for buying and selling cars through auctions while ensuring the safety and convenience of its users.

Keywords: **Car auction system, C# technologies, .NET, NextJS, Docker, RabbitMQ communication, PostgreSQL and MongoDB databases, car search, auction end notifications, token authentication.**

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΕΥΧΑΡΙΣΤΙΕΣ 6

ΠΕΡΙΛΗΨΗ 7

ABSTRACT 8

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ 9

ΚΑΤΑΛΟΓΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ/ΕΙΚΟΝΩΝ 10

1. Αντικείμενο πτυχιακής εργασίας 12

1.1 Σύνοψη πτυχιακής εργασίας 12

2. Τεχνολογίες Ανάπτυξης Εφαρμογής 13

2.1 C# και .Net 13

2.2 NextJS 15

2.3 MongoDB και PostgreSQL 17

2.4 RabbitMQ 22

2.5 Duendy Software 25

2.6 Docker 28

3. Υλοποίηση της εφαρμογής 30

3.1 Δομή της εφαρμογής 31

3.2 Δομή της βάσης δεδομένων 35

3.3 Controllers 37

3.4 Models 44

3.5 Interface εφαρμογής 51

3.6 Δημιουργία Δημοπρασίας 54

ΣΥΜΠΕΡΑΣΜΑΤΑ 55

ΒΙΒΛΙΟΓΡΑΦΙΑ 55

ΚΑΤΑΛΟΓΟΣ ΔΙΑΓΡΑΜΜΑΤΩΝ/ΕΙΚΟΝΩΝ

Εικόνα 1. Αρχιτεκτονική ASP.NET..... σελ.15

Εικόνα 2. Συνδεσιμότητα Next.Js με Authentication…….....…...σελ. 17

Εικόνα 3. Τρόπος λειτουργίας MongoDB..……...…….....σελ. 19

Εικόνα 4. PostgreSQL λογότυπο..…….…….....…….....σελ. 21

Εικόνα 5. Επικοινωνία Βάσης δεδομένων με τους πίνακες …….....…….....σελ. 22

Εικόνα 6. Τρόπος επικοινωνίας Producer με Consumer μέσο RabbitMq ….....…….....σελ. 24

Εικόνα 7. Δημιουργία Producer …….....…….....σελ. 25

Εικόνα 8. Δημιουργία Consumer..…….....σελ. 26

Εικόνα 9. Τρόπος λειτπυργίας Duendy Software με OAuth 2.0 …….....σελ. 27

Εικόνα 10. Σύστημα διαχείρισης Authentication …….....σελ. 28

Εικόνα 11. Δημιουργία και χρήση του Docker…….....σελ. 30

Εικόνα 12. Sketch diagram.....…….....σελ. 33

Εικόνα 13.Uml diagram.....…….....….....σελ. 34

Εικόνα 14. Sequence diagram.....σελ.35

Εικόνα 15. Προσθήκη Authorization στους χρήστες μέσο του Duendy..…….....σελ. 36

Εικόνα 16. Πίνακας Auctions..…….....…...σελ. 37

Εικόνα 17. Πίνακας Auctions με στοιχεία της δημοπρασίας...…….....σελ. 37

Εικόνα 18. Πίνακας Items με τα στοιχεία από τα οχήματα.……...…σελ. 38

Εικόνα 19. Πίνακας OutboxMesage...…….....…….....σελ. 38

Εικόνα 20. Πίνακας OutboxState.....…….....…….....σελ. 40

Εικόνα 21. Δημιουργία GET μεθόδου..…….....…….....σελ. 41

Εικόνα 22. Μέθοδος POST και PUT ..... σελ. 42

Εικόνα 23. Μεθοδος Delete..…….....…….....……σελ. 43

Εικόνα 24. Δημιουργία POST μεθόδου για τις προσφορές…….....……σελ. 44

Εικόνα 25. Δημιουργία Get μεθόδου για τις προσφορές....…….....……σελ. 45

Εικόνα 26. Δημιουργία Get για τον SearchController....….....……σελ. 46

Εικόνα 27. Δημιουργία Auction Model..…….....……σελ. 48

Εικόνα 28. Δημιουργία Item Model..……..….....……σελ. 49

Εικόνα 29. Δημιουργία model Status..…….....……σελ. 50

Εικόνα 30. Δημιουργία και επικοινωνία Auction model..…...…….....……σελ. 50

Εικόνα 32. Δημιουργία Bid Model..…….....……σελ. 51

Εικόνα 32. Δημιουργία model BidStatus..…….....…...……σελ. 52

Εικόνα 33. Δημιουργία Item model στο SearchService..…….....……σελ. 53

Εικόνα 34. Αρχική σελίδα εφαρμογής..……..….....……σελ. 54

Εικόνα 35. Φίλτρα αναζήτησης..…….....…….....……σελ. 54

Εικόνα 36. Προσθήκη προσφοράς..…….....…….....……σελ. 55

Εικόνα 37. Τωρινή προσφορά στο όχημα..…….....……..……σελ. 55

Εικόνα 38. Έγκυρη προσφορά ..... σελ. 56

Εικόνα 39. Η φόρμα με τα στοιχεία της δημοπρασίας.....……σελ. 57

Εικόνα 40. Επιτυχής φόρμα δημοπρασίας..……..……σελ. 57

1.Αντικείμενο πτυχιακής εργασίας

Αντικείμενο της εργασίας αυτής αποτελεί η δημιουργία μιας διαδικτυακής πλατφόρμας δημοπρασιών μηχανοκίνητων οχημάτων. Στην εφαρμογή υπάρχουν διαφορετικοί ρόλοι χρηστών με αντίστοιχα δικαιώματα, όπως για παράδειγμα ο διαχειριστής της κάθε δημοπρασίας όπου μπορεί να δει ποιες δημοπρασίες του ανήκουν και είναι ενεργές. Επίσης μπορεί να δει ποιες δημοπρασίες έχουν λήξει και ακόμα να ορίσει τον χρόνο που θέλει να διαρκέσει η κάθε δημοπρασία. Επιπλέον υπάρχει και ο απλός χρήστης ο οποίος μπορεί να κάνει προφορές σε όποια δημοπρασία είναι διαθέσιμη εκείνη την στιγμή. Με βάση τα παραπάνω η κύρια τεχνολογία που χρησιμοποιήθηκε για την ανάπτυξη της εφαρμογής ήταν το framework ASP.NET σε συνδυασμό με την Next.JS για την δημιουργία της ιστοσελίδας, ενώ τα δεδομένα αποθηκεύτηκαν στις βάσεις δεδομένων PostgreSQL και MongoDB.

1.1 Σύνοψη πτυχιακής εργασίας

Η παρούσα εργασία αποτελείται από τρία κεφάλαια. Στο πρώτο κεφάλαιο περιγράφεται το αντικείμενο της εργασίας. Στο δεύτερο κεφάλαιο πραγματοποιείται μια παρουσίαση των τεχνολογιών που χρησιμοποιήθηκαν. Στο τρίτο και τελευταίο κεφάλαιο ακολουθεί η περιγραφή της ανάπτυξης της εφαρμογής παρουσιάζοντας τον κώδικα και την εφαρμογή.

2.Τεχνολογίες Ανάπτυξης Εφαρμογής

Στο κεφάλαιο αυτό γίνεται μια παρουσίαση των τεχνολογιών που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής της παρούσας εργασίας. Πιο συγκεκριμένα, παρουσιάζονται η γλώσσα προγραμματισμού C# με το framework της .Net το οποίο συνδυάζεται με την NextJS όπου χρησιμοποιείται στο front κομμάτι της εφαρμοφής. Επίσης χρησιμοποιώ ως βάση δεδομένων την PostgreSQL και την MongoDB για την καταχώρηση και την αποθήκευση τον στοιχείων. Επιπλέον για την επικοινωνία μεταξύ των συστημάτων χρησιμοποιήθηκε το λογισμικό της RabbitMQ. Για την ταυτοποίηση τον user επωφελήθηκα με την χρήση του Duende και επίσης για την διαχείριση των συστημάτων χρησιμοποιήθηκε η τεχνολογία του Docker.

2.1 C# και .Net

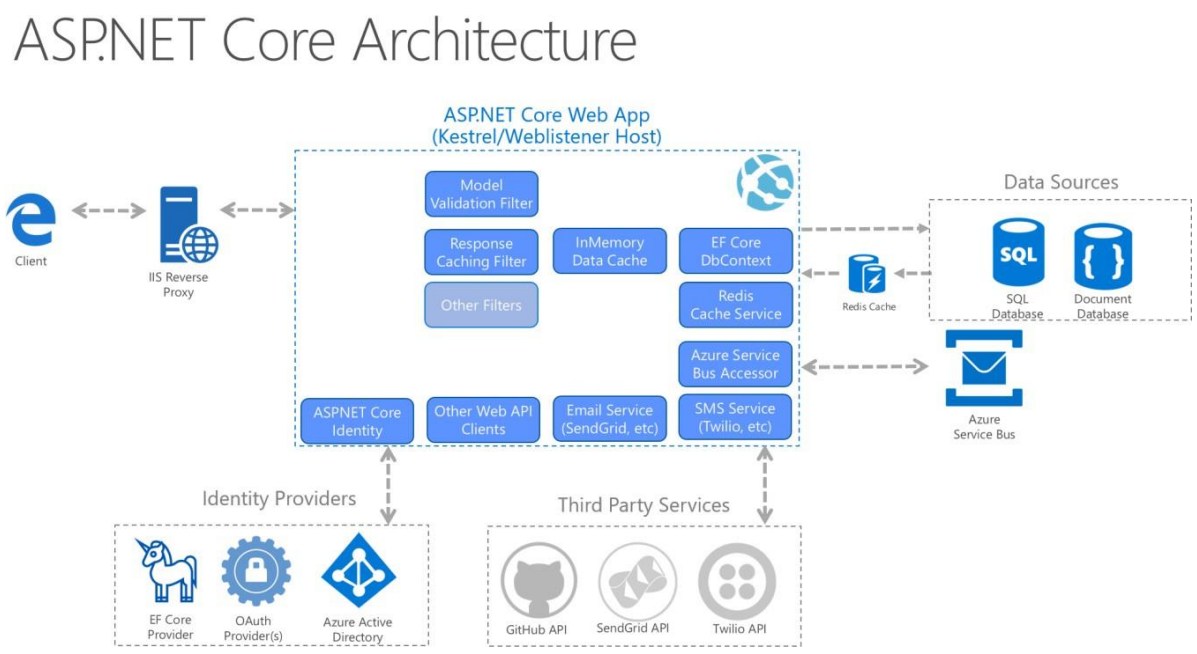
Η C# είναι μια σύγχρονη, αντικειμενοστραφής γλώσσα προγραμματισμού που σχεδιάστηκε για την ανάπτυξη εφαρμογών στην πλατφόρμα .NET της Microsoft. Η C# είναι γνωστή για την ευκολία χρήσης και την ισχυρή της υποστήριξη για ανάπτυξη εφαρμογών που κυμαίνονται από απλά προγράμματα γραμμής εντολών μέχρι σύνθετες εφαρμογές ιστού και κινητών συσκευών.

Το .NET είναι ένα ολοκληρωμένο πλαίσιο λογισμικού που παρέχει ένα σύνολο εργαλείων και βιβλιοθηκών για την ανάπτυξη λογισμικού. Υποστηρίζει διάφορες γλώσσες προγραμματισμού, με την C# να είναι η πιο διαδεδομένη. Το .NET επιτρέπει στους προγραμματιστές να δημιουργούν εφαρμογές που εκτελούνται σε πολλαπλές πλατφόρμες, όπως Windows, macOS και Linux, προσφέροντας δυνατότητες όπως διαχείριση μνήμης, ασφάλεια και διαχείριση εξαιρέσεων.

Η C# συνδυάζει την απλότητα και την αποτελεσματικότητα, καθιστώντας την κατάλληλη τόσο για αρχάριους όσο και για έμπειρους προγραμματιστές. Υποστηρίζει χαρακτηριστικά όπως τύπους δεδομένων, αντικειμενοστραφή προγραμματισμό, και σύγχρονο προγραμματισμό με ασύγχρονες μεθόδους, επιτρέποντας τη δημιουργία εφαρμογών υψηλής απόδοσης και διαθεσιμότητας.

Στο .NET, η ανάπτυξη εφαρμογών υποστηρίζεται από ένα ευρύ φάσμα εργαλείων, όπως το Visual Studio και το Visual Studio Code, τα οποία προσφέρουν περιβάλλοντα ανάπτυξης που διευκολύνουν την συγγραφή, τον εντοπισμό σφαλμάτων και τη διαχείριση του κώδικα. Επιπλέον, η πλατφόρμα .NET Core και η μετεξέλιξή της, το .NET 5 και οι επόμενες εκδόσεις, έχουν ανοιχτό κώδικα και είναι πολλαπλών πλατφορμών, καθιστώντας την ανάπτυξη εφαρμογών ακόμα πιο ευέλικτη και προσαρμοστική στις ανάγκες των σύγχρονων προγραμματιστών.

Με την C# και το .NET, οι προγραμματιστές έχουν στη διάθεσή τους ένα ισχυρό εργαλείο για τη δημιουργία σταθερών, ασφαλών και αποδοτικών εφαρμογών για διάφορες πλατφόρμες[1].



Εικόνα 1: Αρχιτεκτονική ASP.NET

Το .NET Framework (προφέρεται ως "ντοτ νετ") ξεκίνησε ως ένα πλαίσιο λογισμικού που ανήκει στη Microsoft και κυρίως λειτουργεί σε περιβάλλοντα Microsoft Windows. Αρχικά, υπηρετούσε ως το κυρίαρχο πλαίσιο της Κοινής Γλωσσικής Υποδομής (CLI) πριν αντικατασταθεί από το έργο .NET που υποστηρίζει πολλές πλατφόρμες. Περιλαμβάνει μια εκτεταμένη βιβλιοθήκη κλάσεων, γνωστή ως Framework Class Library (FCL), που παρέχει δυνατότητες διαλειτουργικότητας μεταξύ γλωσσών προγραμματισμού και πολλές άλλες λειτουργίες. Στην εικόνα 1 βλέπουμε την αρχιτεκτονική επικοινωνίας την γλώσσας

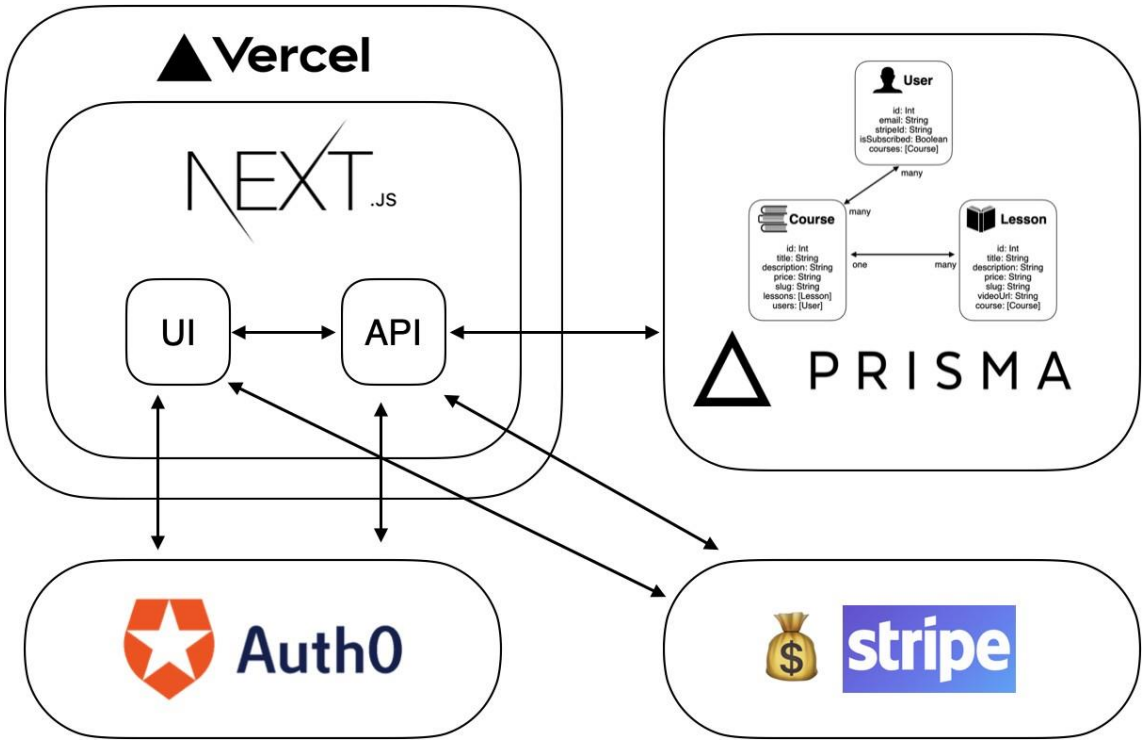
ASP.NET. Τα προγράμματα που αναπτύσσονται για το .NET Framework εκτελούνται σε ένα περιβάλλον λογισμικού, γνωστό ως Common Language Runtime (CLR). Το CLR είναι μια εικονική μηχανή εκτέλεσης που παρέχει υπηρεσίες όπως ασφάλεια, διαχείριση μνήμης και διαχείριση εξαιρέσεων. Επομένως, ο κώδικας που αναπτύσσεται με το .NET Framework αναφέρεται ως "διαχειριζόμενος κώδικας". Η FCL και το CLR αποτελούν μαζί το .NET Framework.

Η FCL παρέχει λειτουργίες διεπαφής χρήστη, πρόσβαση σε δεδομένα, συνδεσιμότητα βάσεων δεδομένων, κρυπτογραφία, ανάπτυξη διαδικτυακών εφαρμογών, αριθμητικούς αλγόριθμους και δικτυακή επικοινωνία. Οι προγραμματιστές δημιουργούν λογισμικό συνδυάζοντας τον κώδικά τους με το .NET Framework και άλλες βιβλιοθήκες. Αυτό το πλαίσιο προορίζεται να χρησιμοποιηθεί από τις περισσότερες νέες εφαρμογές που αναπτύσσονται για την πλατφόρμα Windows. Επιπλέον, η Microsoft παρέχει ένα ενσωματωμένο περιβάλλον ανάπτυξης λογισμικού .NET, το οποίο ονομάζεται Visual Studio.

Επιπλέον, το .NET Framework ξεκίνησε ως ιδιόκτητο λογισμικό, αλλά η εταιρεία εργάστηκε για να τυποποιήσει σχεδόν αμέσως τη στοίβα λογισμικού, ακόμη και πριν από την πρώτη του κυκλοφορία. Παρά τις προσπάθειες τυποποίησης, οι προγραμματιστές, ειδικά αυτοί στις κοινότητες ελεύθερου και ανοιχτού κώδικα, εξέφρασαν ανησυχία για τους επιλεγμένους όρους και τις προοπτικές οποιασδήποτε εφαρμογής ανοιχτού κώδικα, ειδικά όσον αφορά τις πατέντες λογισμικού. Από τότε, η Microsoft άλλαξε την πορεία ανάπτυξης του .NET προς ένα πιο στενότερο μοντέλο σύμφωνα με τις σύγχρονες αρχές ενός έργου λογισμικού που αναπτύχθηκε από την κοινότητα, συμπεριλαμβανομένης της ενημέρωσης στο δίπλωμα ευρεσιτεχνίας που υπόσχεται να αντιμετωπίσει αυτές τις ανησυχίες[2].

2.2 NextJS

Η Next.js είναι ένα framework της React που παρέχει πολλές επιπλέον δυνατότητες, συμπεριλαμβανομένης της απόδοσης από την πλευρά του διακομιστή και της δημιουργίας στατικών ιστότοπων. Η React είναι μια βιβλιοθήκη JavaScript που συνήθως χρησιμοποιείται για τη δημιουργία εφαρμογών ιστού που αποδίδονται στο πρόγραμμα περιήγησης του πελάτη με JavaScript. Ωστόσο, οι προγραμματιστές αντιλαμβάνονται πολλά προβλήματα με αυτήν τη στρατηγική, όπως η μη εξυπηρέτηση χρηστών που δεν έχουν πρόσβαση στην JavaScript ή το έχουν απενεργοποιήσει, πιθανά ζητήματα ασφαλείας, σημαντικοί χρόνοι φόρτωσης σελίδων και προβλήματα στη συνολική βελτιστοποίηση για μηχανές αναζήτησης. Πλαίσια όπως η Next.js αντιμετωπίζουν αυτά τα προβλήματα επιτρέποντας σε μέρος ή ολόκληρο τον ιστότοπο να αποδοθεί από την πλευρά του διακομιστή πριν αποσταλεί στον πελάτη. Η Next.js είναι ένα από τα πιο δημοφιλή πλαίσια για το React και απαιτεί Node.js για την αρχικοποίησή του.



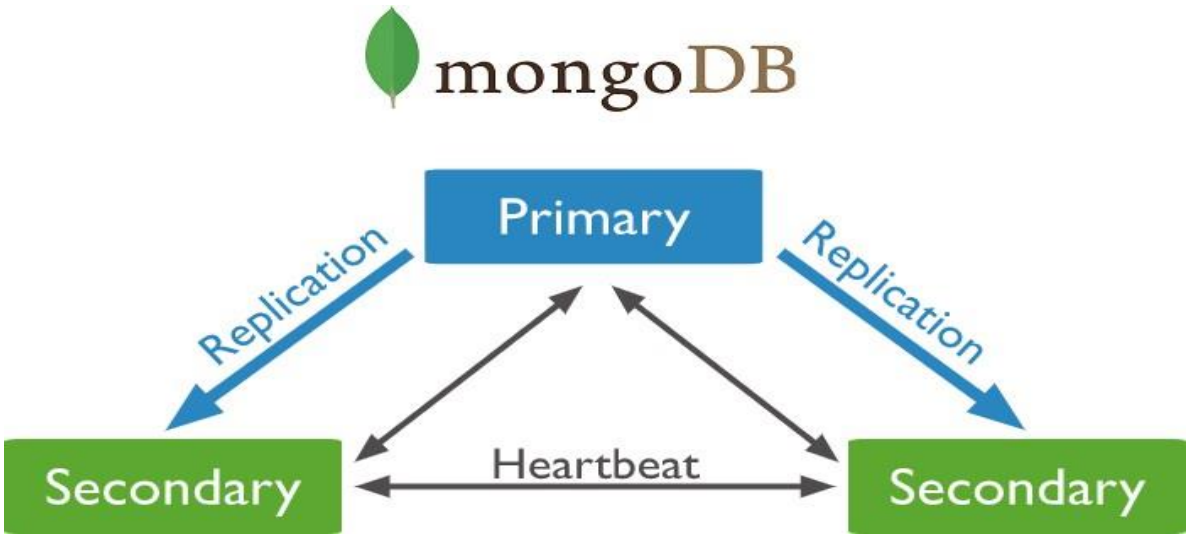
Εικόνα 2 : Συνδεσιμότητα Next.Js με Authentication

Η Next.js υποστηρίζει στυλ με CSS καθώς και προμεταγλωττισμένα Scss και Sass, CSS-in-JS και JSX με στυλ. Επιπλέον, είναι κατασκευασμένο με υποστήριξη TypeScript και έξυπνη δέσμη. Στην εικόνα 2 βλέπουμε την Next.Js να επικοινωνεί με ένα σύστημα API και το Authentication. Το transpiler ανοιχτού κώδικα SWC χρησιμοποιείται για τη μετατροπή και τη μεταγλώττιση κώδικα σε JavaScript που μπορεί να χρησιμοποιηθεί από ένα πρόγραμμα περιήγησης. Το Webpack, ένα άλλο εργαλείο ανοιχτού κώδικα, χρησιμοποιείται για τη δέσμη των μονάδων στη συνέχεια, ωστόσο αυτή τη στιγμή αντικαθίσταται με το TurboPack. Όλα αυτά τα εργαλεία χρησιμοποιούνται με npm σε ένα τερματικό[3]. Με την NextJS έχουμε την δυνατότητα να σχεδιάσουμε το δικό μας front κομμάτι το οποίο έχει βοηθήσει και στην ανάπτυξη της εφαρμογής. Στο πρόγραμμα θα βρούμε τον κώδικα της NextJS στον φάκελο NextJS με όνομα front εκεί έχουν γίνει όλες οι μετατροπές και οι αναβαθμίσεις που χρειάζονται.

2.3 MongoDB και PostgreSQL

- **MongoDB**

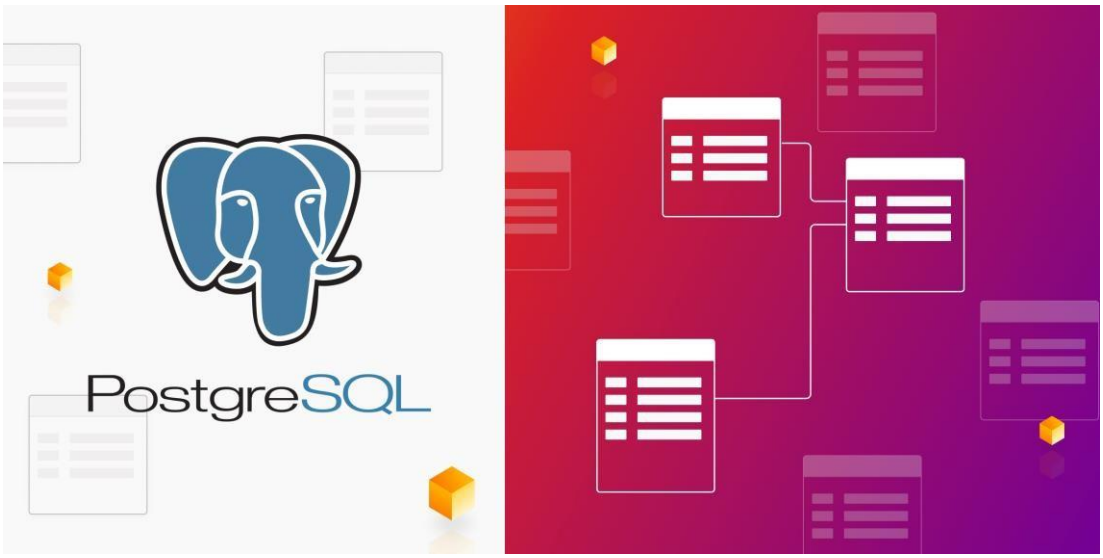
Η MongoDB είναι ένα πρόγραμμα βάσης δεδομένων προσανατολισμένο σε έγγραφα, διαθέσιμο σε πολλαπλές πλατφόρμες. Ταξινομημένο ως προϊόν βάσης δεδομένων NoSQL, η MongoDB χρησιμοποιεί έγγραφα τύπου JSON με προαιρετικά σχήματα. Η MongoDB αναπτύχθηκε από την MongoDB Inc. και οι τρέχουσες εκδόσεις έχουν άδεια χρήσης βάσει της δημόσιας άδειας χρήσης από την πλευρά του διακομιστή (SSPL). Η MongoDB είναι μέλος της MACH Alliance. Η MongoDB υποστηρίζει αναζητήσεις πεδίων, εύρους και κανονικής έκφρασης. Τα ερωτήματα (queries) μπορούν να επιστρέψουν συγκεκριμένα πεδία εγγράφων και επίσης να περιλαμβάνουν συναρτήσεις JavaScript που ορίζονται από το χρήστη. Τα queries μπορούν επίσης να ρυθμιστούν ώστε να επιστρέφουν ένα τυχαίο δείγμα αποτελεσμάτων ενός δεδομένου μεγέθους.



Εικόνα 3: Τρόπος λειτουργίας MongoDB

Η MongoDB προσφέρει υψηλή διαθεσιμότητα με αντίγραφα σετ. Ένα σύνολο αντιγράφων αποτελείται από δύο ή περισσότερα αντίγραφα των δεδομένων. Κάθε μέλος σετ αντιγράφων μπορεί να ενεργεί ως πρωτεύον ή δευτερεύον αντίγραφο ανά πάσα στιγμή. Όλες οι εγγραφές και οι αναγνώσεις γίνονται στο πρωτεύον αντίγραφο από προεπιλογή. Τα δευτερεύοντα αντίγραφα διατηρούν ένα αντίγραφο των δεδομένων του πρωτεύοντος χρησιμοποιώντας ενσωματωμένη αναπαραγωγή. Όταν ένα πρωτεύον αντίγραφο αποτυγχάνει, το σύνολο αντιγράφων διεξάγει αυτόματα μια εκλογική διαδικασία για να καθορίσει ποιο δευτερεύον θα γίνει το κύριο. Τα δευτερεύοντα μπορούν προαιρετικά να εξυπηρετούν λειτουργίες ανάγνωσης, αλλά αυτά τα δεδομένα είναι μόνο τελικά συνεπή από προεπιλογή. Η εικόνα 3 μας δείχνει τον τρόπο λειτουργίας της MongoDB[4]. Με βάση τον τρόπο λειτουργείας της MongoDB επέλεξα να είναι μια από τις δυο βάσεις δεδομένων που θα χρησιμοποιηθούν στο πρόγραμμα. Η συγκεκριμένη επιλογή έγινε επειδή η MongoDB είναι μία βάση δεδομένων NoSQL και αποθηκεύει τα αρχεία της σε μορφή JSON. Είναι ένας γρήγορος τρόπος για δεδομένα που αποθηκεύονται και χρειάζονται ανάγνωση και εγγραφή. Χρησιμοποίησα την συγκεκριμένη βάση δεδομένων για να αποθηκεύσουμε τα δεδομένα από την κάθε προσφορά αλλά και από την αναζήτηση οχημάτων που γίνεται στην εφαρμογή δηλαδή στο SearchService και στο BiddingService.

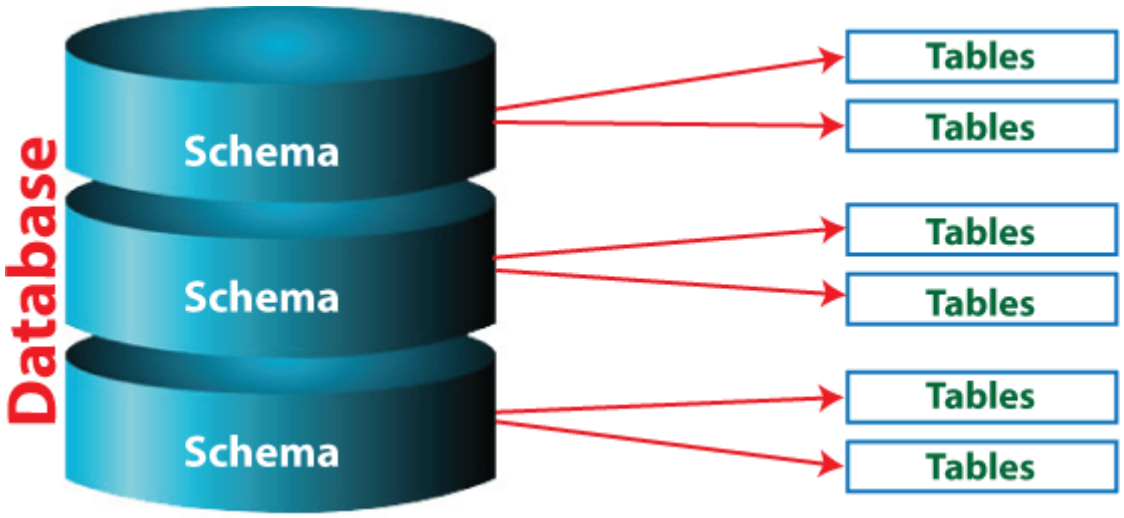
- **PostgreSQL**



Εικόνα 4: PostgreSQL λογότυπο

Το έργο, με την ονομασία POSTGRES, είχε ως στόχο να ενσωματώσει τα ελάχιστα χαρακτηριστικά που απαιτούνταν για την πλήρη υποστήριξη τύπων δεδομένων. Αυτά περιλάμβαναν τη δυνατότητα ορισμού τύπων και πλήρους περιγραφής σχέσεων, κάτι που χρησιμοποιείται ευρέως αλλά συχνά απαιτείται να οριστεί εξ ολοκλήρου από τον χρήστη.

Στην POSTGRES, η βάση δεδομένων κατανόησε τις σχέσεις και μπορούσε να ανακτήσει πληροφορίες σε σχετικούς πίνακες με φυσικό τρόπο, χρησιμοποιώντας κανόνες. Πολλές από τις ιδέες του Ingres ενσωματώθηκαν στο POSTGRES, αλλά όχι ο κώδικας του. Η PostgreSQL παρέχει ενσωματωμένη υποστήριξη για κανονικά ευρετήρια B-tree και κατακερματισμού πινάκων, καθώς και τέσσερις μεθόδους πρόσβασης σε ευρετήρια γενικευμένα δέντρα αναζήτησης (GiST), γενικευμένα ανεστραμμένα ευρετήρια (GIN), Space-Partitioned GiST (SP-GiST) και Block Range Indexes. Επιπλέον, μπορούν να δημιουργηθούν μέθοδοι ευρετηρίου που ορίζονται από το χρήστη, αν και αυτή είναι μια αρκετά περίπλοκη διαδικασία.



Εικόνα 5: Επικοινωνία Βάσης δεδομένων με τους πίνακες

Τα σχήματα στο PostgreSQL λειτουργούν ως χώροι ονομάτων, επιτρέποντας σε αντικείμενα με τον ίδιο τύπο και όνομα να υπάρχουν στην ίδια βάση δεδομένων. Αυτά δεν πρέπει να μπερδευτούν με ένα σχήμα βάσης δεδομένων, το οποίο είναι μια αφηρημένη, δομική, οργανωτική προδιαγραφή που καθορίζει τις σχέσεις των δεδομένων σε διάφορους πίνακες. Όλα τα αντικείμενα της βάσης δεδομένων στο PostgreSQL, εκτός από κάποια καθολικά, όπως οι ρόλοι και οι χώροι πίνακα, ανήκουν σε ένα συγκεκριμένο σχήμα. Τα σχήματα δεν μπορούν να είναι ενσωματωμένα, δηλαδή ένα σχήμα δεν μπορεί να περιέχει άλλα σχήματα. Το σύστημα δικαιωμάτων ελέγχει την πρόσβαση στα σχήματα και τα περιεχόμενά τους. Από προεπιλογή, οι νέες δημιουργίες θέσεων για βάσεις δεδομένων έχουν ένα προεπιλεγμένο σχήμα με το όνομα "public", επίσης μπορούν να προστεθούν και άλλα σχήματα, και το "public" σχήμα δεν είναι υποχρεωτικό. Στην παραπάνω εικόνα 5 βλέπουμε την δομή μιας βάσης δεδομένων.

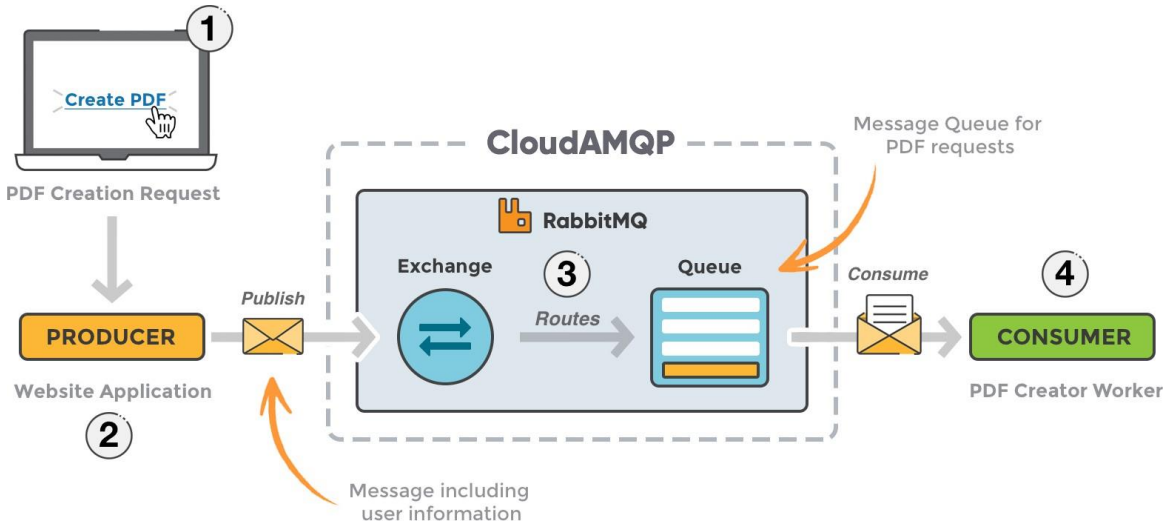
Τέλος υπάρχει υποστήριξη για μια εκτεταμένη ποικιλία ενσωματωμένων τύπων δεδομένων, όπως:

- Boolean
- Αυθαίρετα-αριθμητικά ακριβείας
- Χαρακτήρας (κείμενο, varchar, χαρακτήρας)
- Δυαδικός
- Ημερομηνία/ώρα (χρονοσήμανση/ώρα με/χωρίς ζώνη ώρας, ημερομηνία, διάστημα)
- Χρήματα
- Enum
- Συμβολοσειρές bit
- Τύπος αναζήτησης κειμένου
- Σύνθετος
- HStore, ένα κατάστημα κλειδιού-τιμής εντός της PostgreSQL με δυνατότητα επέκτασης
- Πίνακες (μεταβλητού μήκους και μπορούν να είναι οποιουδήποτε τύπου δεδομένων, συμπεριλαμβανομένων τύπων κειμένου και σύνθετων τύπων) με συνολικό μέγεθος αποθήκευσης έως 1 GB
- Γεωμετρικά πρωτόγονα
- Διευθύνσεις IPv4 και IPv6
- Μπλοκ δρομολόγησης μεταξύ τομέων χωρίς κλάση (CIDR) και διευθύνσεις MAC
- XML με υποστήριξη ερωτημάτων XPath
- Καθολικά μοναδικό αναγνωριστικό (UUID)
- Σημειογραφία αντικειμένου JavaScript (JSON) και γρηγορότερο δυαδικό JSONB (δεν είναι το ίδιο με το BSON) [5].

Μια ακόμα βάση δεδομένων που χρησιμοποιήθηκε στο πρόγραμμα είναι η PostgreSQL. Η συγκεκριμένη databse επιλέχτηκε ως η κύρια βάση του προγράμματος επειδή έχει την δυνατότητα να αποθηκεύσει τα δεδομένα με ακεραιότητα δίνοντας στον προγραμματιστή μια παραπάνω ασφάλεια. Την PostgreSQL την χρησιμοποιούμε στον AuctionService όπου στο συγκεκριμένο αποθηκεύουμε τα δεδομένα που δημιουργείται η κάθε δημοπρασία. Τέλος η χρήση δύο διαφορετικών βάσεων δεδομένων βασίζεται για την καλύτερη απόδοση και κλιμάκωση των δεδομένων για να είναι ποιο εύχρηστα προς τους προγραμματιστές που τα επεξεργάζονται.

2.4 RabbitMQ

Το RabbitMQ είναι ένα δημοφιλές σύστημα μηνυμάτων ανοιχτού κώδικα που επιτρέπει την ανταλλαγή μηνυμάτων μεταξύ διαφορετικών εφαρμογών, συστημάτων ή υπηρεσιών. Βασίζεται στην αρχιτεκτονική του "message broker" και υποστηρίζει διάφορα πρωτόκολλα μηνυμάτων, όπως το AMQP (Advanced Message Queuing Protocol). Στον πυρήνα του, το RabbitMQ διαχειρίζεται ουρές μηνυμάτων, όπου οι παραγωγοί (producers) στέλνουν μηνύματα και οι καταναλωτές (consumers) τα λαμβάνουν. Αυτό επιτρέπει την ασύγχρονη επικοινωνία, βοηθώντας στη διαχείριση της φόρτωσης και στη βελτίωση της απόδοσης των συστημάτων. Το RabbitMQ είναι εξαιρετικά ευέλικτο και κλιμακούμενο, καθιστώντας το ιδανικό για εφαρμογές μεγάλης κλίμακας που απαιτούν αξιόπιστη και αποδοτική επικοινωνία μεταξύ διαφόρων υπηρεσιών ή μικροϋπηρεσιών (microservices).



Εικόνα 6: Τρόπος επικοινωνίας Producer με Consumer μέσο RabbitMq

Παραδείγματα

Παρακάτω δημιουργούμε ένα απλό παράδειγμα χρησιμοποιώντας τη βιβλιοθήκη RabbitMQ για την αλληλεπίδραση με έναν RabbitMQ broker σε Java. Στο παράδειγμα αυτό, θα δημιουργήσουμε έναν παραγωγό (producer) που θα στέλνει ένα μήνυμα στην ουρά, και έναν καταναλωτή (consumer) που θα λαμβάνει και θα εκτυπώνει αυτό το μήνυμα. Στην εικόνα 7 και 8 έχουμε κάποια παραδείγματα από την δημιουργία ενός producer και ενός consumer[6].

```
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using System;
using System.Text;
using System.Threading;

class Program
{
    static void Main(string[] args)
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.QueueDeclare(queue: "hello",
                                 durable: false,
                                 exclusive: false,
                                 autoDelete: false,
                                 arguments: null);

            var consumer = new EventingBasicConsumer(channel);
            consumer.Received += (model, ea) =>
            {
                var body = ea.Body.ToArray();
                var message = Encoding.UTF8.GetString(body);
                Console.WriteLine(" [x] Received {0}", message);
            };
            channel.BasicConsume(queue: "hello",
                                 autoAck: true,
                                 consumer: consumer);

            Console.WriteLine(" Press [enter] to exit.");
            Console.ReadLine();
        }
    }
}
```

Εικόνα 7: Δημιουργία Producer

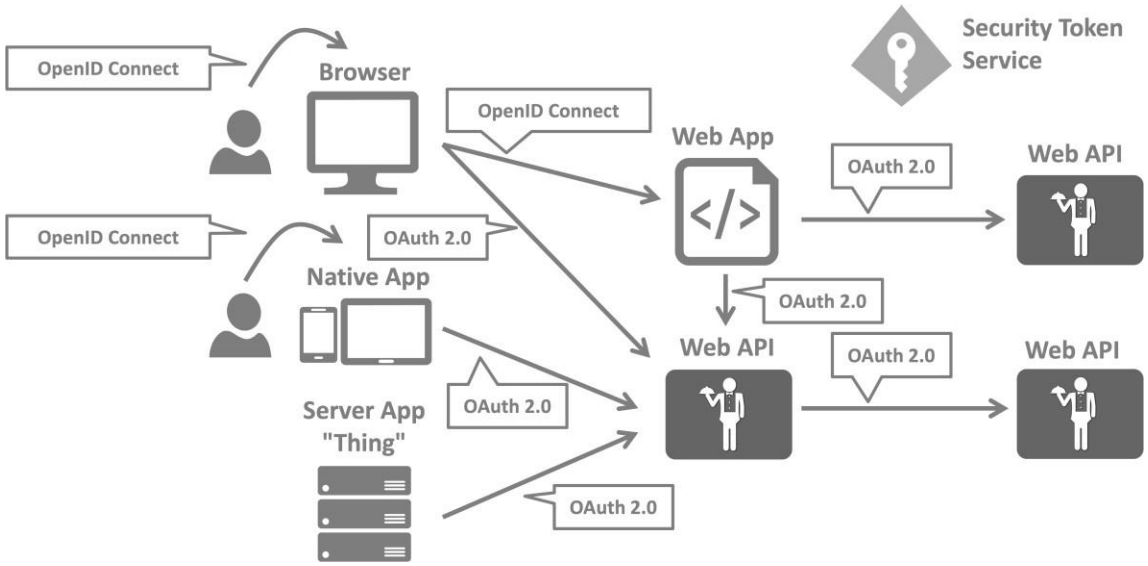
Ομοίως, το ακόλουθο πρόγραμμα λαμβάνει μηνύματα από την ουρά και τα εκτυπώνει στην οθόνη: (Σημείωση: Αυτό το παράδειγμα δεν επιβεβαιώνει τη λήψη του μηνύματος.)

```
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using System;
using System.Text;

class Receive
{
    public static void Main()
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        using (var channel = connection.CreateModel())
        {
            channel.QueueDeclare(queue: "hello",
                                 durable: false,
                                 exclusive: false,
                                 autoDelete: false,
                                 arguments: null);

            var consumer = new EventingBasicConsumer(channel);
            consumer.Received += (model, ea) =>
            {
                var body = ea.Body.ToArray();
                var message = Encoding.UTF8.GetString(body);
                Console.WriteLine(" [x] Received {0}", message);
            };
            channel.BasicConsume(queue: "hello",
                                 autoAck: true,
                                 consumer: consumer);

            Console.WriteLine(" Press [enter] to exit.");
            Console.ReadLine();
        }
    }
}
```

Εικόνα 8: Δημιουργία Consumer

Στο πρόγραμμα χρησιμοποιούμε το RabbitMq στα παρακάτω service το AuctionService, το GatewayService, το NotificationService και το SearchService. Τα συγκεκριμένα service συνδέονται με το RabbitMq και δείχνουν μέσα από notifications τι ειδοποιήσεις έρχονται στον χρήστη.

2.5 Duende Software



Εικόνα 9: Τρόπος λειτουργίας Duende Software με OAuth 2.0

Οι πάροχοι ταυτότητας (IDP) είναι συστήματα που δημιουργούν, διατηρούν και διαχειρίζονται πληροφορίες ταυτότητας για τους χρήστες και παρέχουν υπηρεσίες ελέγχου ταυτότητας σε εφαρμογές εντός μιας ομοσπονδίας ή ενός κατανεμημένου δικτύου. Οι IDP προσφέρουν έλεγχο ταυτότητας χρήστη ως υπηρεσία. Οι εφαρμογές εξαρτώμενων μερών, όπως οι ιστοσελίδες, αναθέτουν το βήμα ελέγχου ταυτότητας χρήστη σε έναν αξιόπιστο πάροχο ταυτότητας. Μια τέτοια εφαρμογή ονομάζεται ομοσπονδιακή, δηλαδή χρησιμοποιεί ομοσπονδιακή ταυτότητα. Ένας IDP είναι "ένας αξιόπιστος πάροχος που σας επιτρέπει να χρησιμοποιείτε ενιαία σύνδεση (SSO) για πρόσβαση σε άλλους ιστότοπους. Το SSO βελτιώνει τη χρηστικότητα μειώνοντας την ανάγκη επαναλαμβανόμενης εισόδου του κωδικού πρόσβασης. Παρέχει επίσης αυξημένη ασφάλεια μείωσης της επιφάνειας επίθεσης. Οι IDP μπορούν να διευκολύνουν τις συνδέσεις μεταξύ των πόρων υπολογιστικού νέφους και των χρηστών, μειώνοντας την ανάγκη για επαναλαμβανόμενο έλεγχο ταυτότητας όταν χρησιμοποιούνται εφαρμογές κινητής τηλεφωνίας και υπηρεσίες περιαγωγής. Στην εικόνα 9 έχουμε τον τρόπο λειτουργίας του Duendy Software με ένα σύστημα OAuth2.0.

Συστήματα διαχείρισης

Ένα σύστημα διαχείρισης ταυτότητας (IAM) αναφέρεται σε ένα σύνολο τεχνολογιών και διαδικασιών που χρησιμοποιούνται για τη διαχείριση της ταυτότητας και της πρόσβασης σε πόρους σε ένα επιχειρησιακό περιβάλλον ή ένα δίκτυο.

Οι παρακάτω όροι είναι συνυφασμένοι με το "σύστημα διαχείρισης ταυτότητας":

- Σύστημα πρόσβασης-διακυβέρνησης
- Σύστημα διαχείρισης ταυτότητας και πρόσβασης
- Σύστημα διαχείρισης δικαιωμάτων
- Σύστημα παροχής χρηστών

Εικόνα 10: Σύστημα διαχείρισης Authentication

Η διαχείριση ταυτότητας, γνωστή και ως διαχείριση ταυτότητας και πρόσβασης (IAM), αποτελεί ένα πλαίσιο ασφαλείας ταυτότητας που επιβλέπει την επικύρωση της ταυτότητας και την αδειοδότηση της πρόσβασης των χρηστών σε πόρους, όπως εφαρμογές, δεδομένα, συστήματα και πλατφόρμες cloud. Στοχεύει στο να εξασφαλίσει ότι μόνο οι αρμόδιοι χρήστες έχουν πρόσβαση στους κατάλληλους πόρους για τους κατάλληλους λόγους.

Καθώς ο ψηφιακός κόσμος εξελίσσεται, το ίδιο κάνει και η διαχείριση ταυτότητας. Οι "διαχειριστές ταυτότητας" και οι "διαχειριστές πρόσβασης και ταυτότητας" (ή AIM) είναι όροι που χρησιμοποιούνται παράλληλα με τη διαχείριση ταυτότητας, ενώ η ίδια η διαχείριση ταυτότητας ανήκει στον ευρύτερο τομέα της ασφάλειας πληροφοριών και του απορρήτου πληροφοριών, καθώς και της χρηστικότητας και ενσωμάτωσης. Στην εικόνα 10 βλέπουμε τον τρόπο λειτουργίας για ένα Authentication.

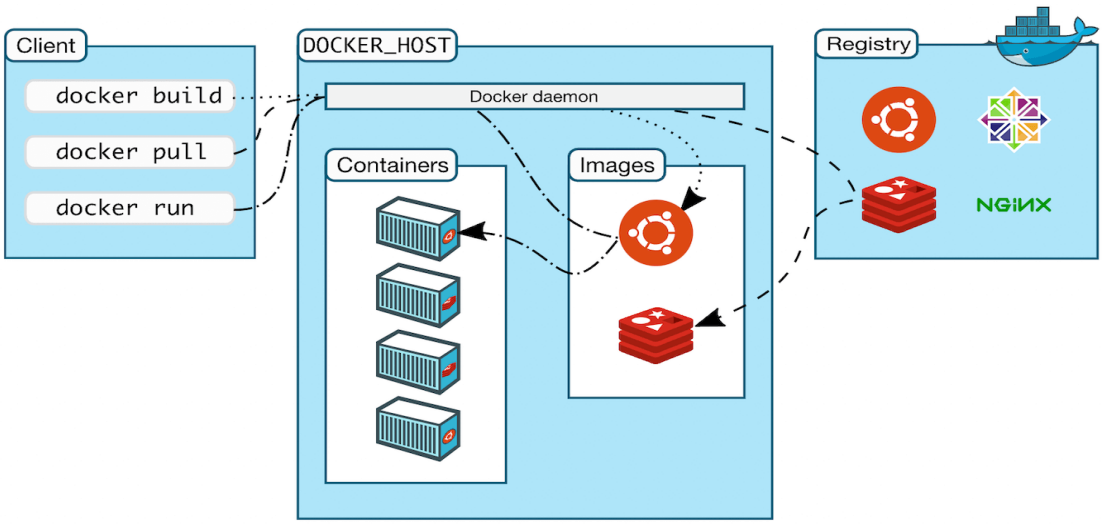
Τα στοιχεία του συστήματος διαχείρισης ταυτότητας και πρόσβασης (IAM) περιλαμβάνουν:

- Διαχείριση πρόσβασης/Μοναδική σύνδεση: Επικύρωση της ταυτότητας των χρηστών πριν από την παροχή πρόσβασης σε δίκτυα και εφαρμογές.
- Διακυβέρνηση ταυτότητας: Διασφάλιση ότι η πρόσβαση των χρηστών είναι σύμφωνη με τις πολιτικές πρόσβασης και τις αλλαγές στους ρόλους/τις ευθύνες.
- Προνομιακή διαχείριση πρόσβασης: Έλεγχος και παρακολούθηση της πρόσβασης σε προνομιακούς λογαριασμούς, εφαρμογές και στοιχεία του συστήματος.
- Αυτές οι τεχνολογίες μπορούν να συνδυαστούν χρησιμοποιώντας τη διακυβέρνηση ταυτότητας, η οποία παρέχει τη βάση για αυτοματοποιημένες ροές εργασίας και διαδικασίες[7].

Μέσα στο πρόγραμμα έχω δημιουργήσει ένα IdentityService από το οποίο μπορούμε να ταυτοποιήσουμε τον κάθε χρήστη που μπαίνει στην εφαρμογή με το IdentityKey. Είναι ξεχωριστό για τον κάθε χρήστη αλλά και κάθε φορά που κάποιος χρήστης δημιουργεί έναν καινούργιο λογαριασμό το Duende δημιουργεί ένα καινούργιο κλειδί για τον χρήστη. Το Duendy χρησιμοποιείται στην εφαρμογή για την προστασία των χρηστών και τον δεδομένων.

2.6 Docker

Το Docker είναι μια πλατφόρμα υπηρεσιών που χρησιμοποιεί εικονικοποίηση σε επίπεδο λειτουργικού συστήματος. Χρησιμοποιείται για τη δημιουργία και παράδοση λογισμικών σε πακέτα που ονομάζονται κοντέινερ. Η πλατφόρμα αυτή περιλαμβάνει τόσο δωρεάν όσο και premium εκδόσεις, ενώ το λογισμικό που φιλοξενεί και διαχειρίζεται τα κοντέινερ ονομάζεται Docker Engine. Κυκλοφόρησε για πρώτη φορά το 2013 και αναπτύχθηκε από την Docker, Inc. Το Docker λειτουργεί ως εργαλείο αυτοματοποίησης για τη διαδικασία ανάπτυξης εφαρμογών μέσω ελαφρών κοντέινερ, επιτρέποντας στις εφαρμογές να λειτουργούν αποδοτικά σε διάφορα μοναδικά περιβάλλοντα. Στην παρακάτω εικόνα 11 έχουμε τον τρόπο λειτουργίας του Docker.



Εικόνα 11: Δημιουργία και χρήση του Docker

Τα κοντέινερ είναι περιβάλλοντα απομόνωσης που περιλαμβάνουν το δικό τους λογισμικό,

βιβλιοθήκες και ρυθμίσεις διαμόρφωσης. Κάθε κοντέινερ λειτουργεί ανεξάρτητα από τα άλλα, επιτρέποντάς τους να εκτελούνται σε ένα ενιαίο σύστημα χωρίς να επηρεάζονται από τα υπόλοιπα. Με τη χρήση καθορισμένων καναλιών, τα κοντέινερ μπορούν να επικοινωνούν μεταξύ τους, διευκολύνοντας τη δημιουργία πολύ-συστημάτων. Επειδή τα κοντέινερ χρησιμοποιούν κοινό πυρήνα λειτουργικού συστήματος, αξιοποιούν τους πόρους του συστήματος αποτελεσματικότερα σε σύγκριση με τις εικονικές μηχανές, οι οποίες απαιτούν ξεχωριστό πυρήνα για κάθε εικονική μηχανή.

Το Docker είναι ένα εργαλείο που μπορεί να συσκευάσει μια εφαρμογή και τις αντίστοιχες εξαρτήσεις της σε ένα εικονικό κοντέινερ, το οποίο μπορεί να εκτελεστεί σε οποιονδήποτε υπολογιστή με Linux, Windows ή macOS. Αυτό παρέχει τη δυνατότητα στην εφαρμογή να εκτελείται σε διάφορα περιβάλλοντα όπως εγκαταστάσεις, δημόσιο ή ιδιωτικό cloud. Όταν εκτελείται σε Linux, το Docker χρησιμοποιεί τις δυνατότητες απομόνωσης πόρων του πυρήνα του Linux, όπως τα cgroups και τα namespaces, καθώς και ένα σύστημα αρχείων overlay, προκειμένου να επιτρέψει στα κοντέινερ να εκτελούνται σε ένα μόνο περιβάλλον Linux. Αυτό αποφεύγει την επιβάρυνση του εκκίνησης και της συντήρησης εικονικών μηχανών. Στο macOS, το Docker χρησιμοποιεί μια εικονική μηχανή Linux για την εκτέλεση των κοντέινερ. Επιπλέον, επειδή τα κοντέινερ Docker είναι ελαφριά, μπορούν να εκτελεστούν πολλά κοντέινερ ταυτόχρονα σε έναν διακομιστή ή μια εικονική μηχανή. Η υποστήριξη του πυρήνα Linux για τους χώρους ονομάτων απομονώνει τα κοντέινερ από το περιβάλλον τους, ενώ οι ομάδες cgroups παρέχουν περιορισμό πόρων όπως η μνήμη και η CPU. Τέλος, το Docker υλοποιεί ένα υψηλού επιπέδου API που επιτρέπει τη δημιουργία ελαφρών κοντέινερ που μπορούν να εκτελέσουν διαδικασίες μεμονωμένα[8].

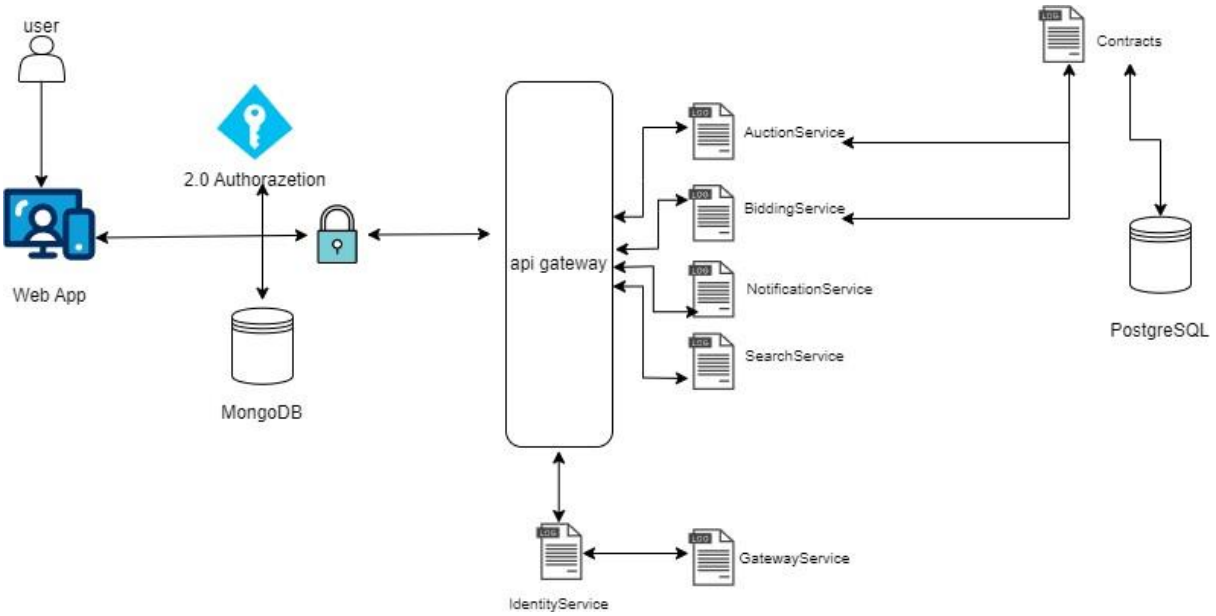
3. Υλοποίηση της εφαρμογής

Στο κεφάλαιο αυτό γίνεται μια περιγραφή του τρόπου υλοποίησης της πλατφόρμας Δημοπρασίας Αυτοκινήτων. Πιο συγκεκριμένα παρουσιάζεται η κυρίως δομή της εφαρμογής αυτής, καθώς και η βασική λειτουργία της. Όπως αναφέρουμε και παραπάνω, τα εργαλεία που χρησιμοποιήθηκαν, ήταν η C# με το framework της .NET. Επίσης σαν βάση δεδομένων χρησιμοποιήθηκε η PostgreSQL και η MongoDB. Επιπλέον στην εφαρμογή έχουν χρησιμοποιηθεί διαφορετικά Services τα οποία κάνουν μια διαφορετική δουλειά όπου θα αναλυθεί στα επόμενα κεφάλαια.

3.1 Δομή της εφαρμογής

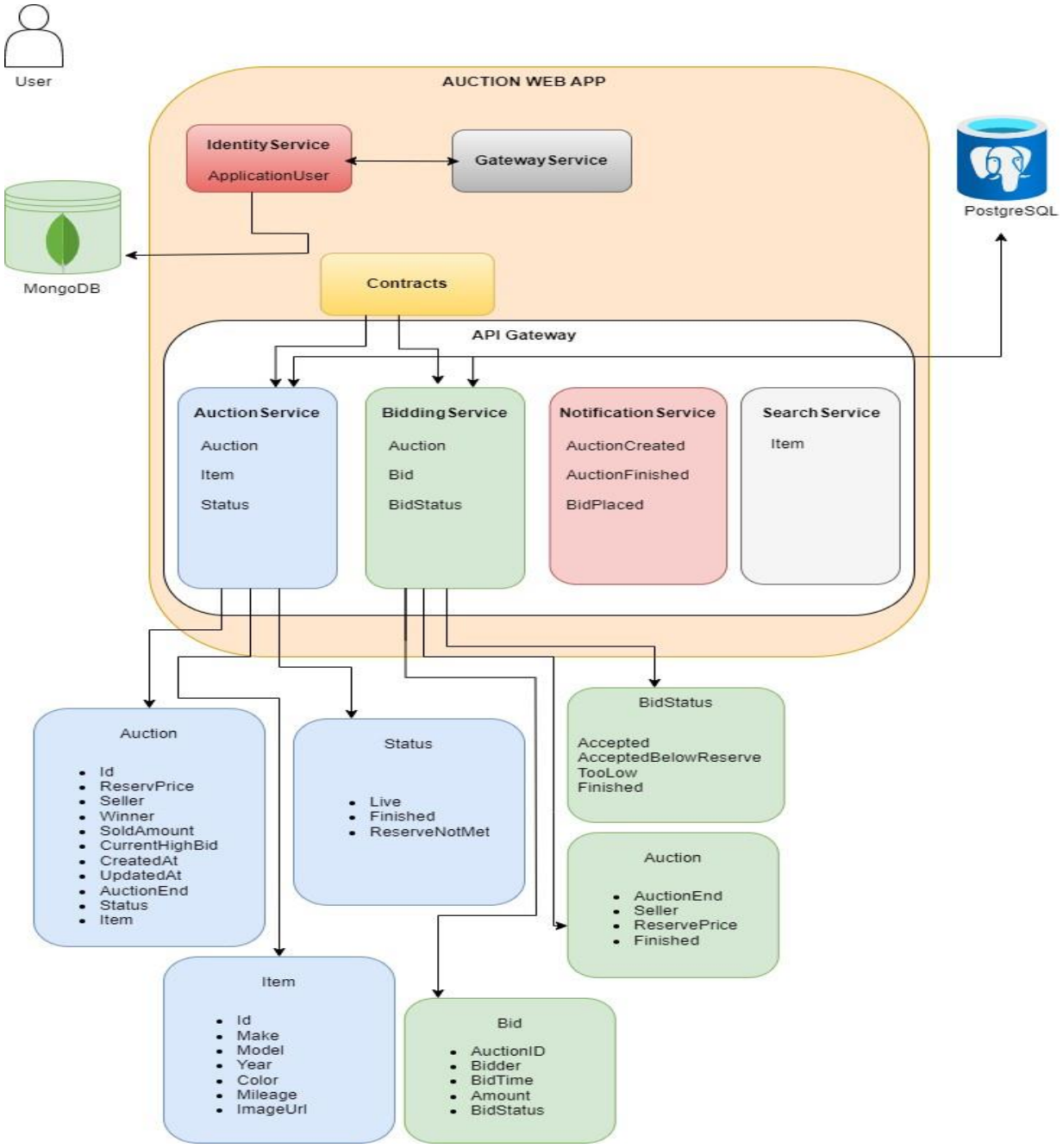
Η εφαρμογή της δημοπρασίας αυτοκινήτων, δημιουργήθηκε με στόχο μιας πλατφόρμας, μέσω της οποίας οι χρήστες που θα δημιουργούν τον δικό τους λογαριασμό και θα μπορούν να δημιουργήσουν την δικιά τους δημοπρασία για το όχημα τους. Μέσα από την εφαρμογή ο καθένας θα μπορεί να κάνει αναζήτηση με βάση το όνομα του οχήματος που θέλει να αναζητήσει. Επίσης ο κάθε χρήστης θα μπορεί να προσθέσει την δική του προσφορά σε άλλες δημοπρασίες που πραγματοποιούνται εκείνη την στιγμή. Ένα ακόμα επιπλέον χαρακτηριστικό είναι πως στον χρήστη που έχει κάνει κάποια προσφορά σε μια δημοπρασία θα του έρχεται ειδοποίηση σχετικά με την εξέλιξη της δημοπρασίας.

Για να μπορεί μια εφαρμογή να εκτελεστή σωστά πρέπει πρώτα να δημιουργηθούν κάποια διαγράμματα όπου με βάση τα συγκεκριμένα θα ξεκινήσει και η υλοποίηση της εφαρμογής.



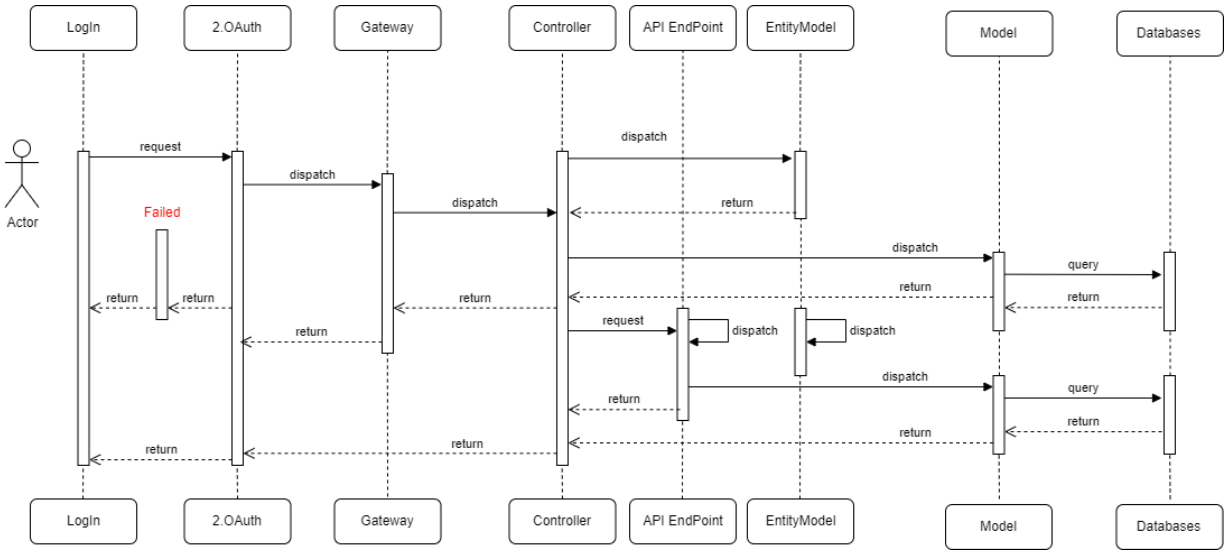
Εικόνα 12: Sketch diagram

Ένα από τα πρώτα πράματα που σχεδίασα ήταν ένα Sketch diagram, όπως βλέπουμε και στην εικόνα 12. Με το συγκεκριμένο διάγραμμα δημιούργησα τον σκελετό της εφαρμογής. Πρόσθεσα τα services που ήθελα να φτιάξω, της βάσης δεδομένων που θα χρησιμοποιούσα αλλά και πως θα λειτουργούσε το authorization της εφαρμογής να επικοινωνήσει με τον χρήστη. Στην συνέχεια δημιούργησα ένα UML diagram το οποίο είναι στην εικόνα 13. Όρισα πως θα ήθελα να είναι η δομή από το κάθε service αλλά και πως θα έπρεπε να επικοινωνούν τα services μεταξύ τους. Με βάση το συγκεκριμένο διάγραμμα έκανα μια μεγάλη πρόοδο σχετικά με την δομή του προγράμματος, με τον τρόπο επικοινωνίας των services, την μέθοδο που θα χρησιμοποιούσα για το authorazation και πως θα επιβεβαίωνε τον χρήστη που πήγαινε να εισέλθει στην εφαρμογή.



Εικόνα 13: Uml diagram

Για τέλος άφησα το Sequence diagram. Το συγκεκριμένο διάγραμμα δημιουργήθηκε για την κατανόηση της σωστής επικοινωνίας μεταξύ τον services αλλά και την σωστή επικοινωνία όλου του συστήματος. Με βάση το παρακάτω διάγραμμα στην εικόνα 14 δίνεται μια ξεκάθαρη εικόνα για την λειτουργία του συστήματος.



Εικόνα 14: Sequence diagram

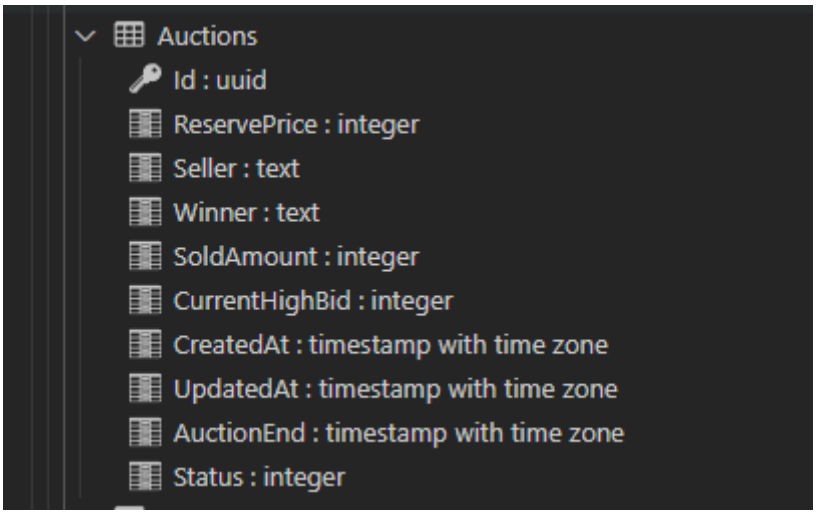
Με βάση τα παραπάνω διαγράμματα η εφαρμογή απευθύνεται στα άτομα που θέλουν να πουλήσουν ή να αγοράσουν σε χαμηλότερο κόστος κάποιο όχημα. Για να μπορέσει κάποιος να έχει πρόσβαση στην πλατφόρμα θα πρέπει να κάνει πρώτα μια εγγραφή στην σελίδα μέσα από την οποία ο κάθε χρήστης θα αποκτήσει ένα μοναδικό Authorization token με την βοήθεια του Duende. Το token θα ανανεώνεται κάθε φορά που ο χρήστης πραγματοποιεί μια σύνδεση στην πλατφόρμα. Με την επιτυχημένη σύνδεση του χρήστη στην πλατφόρμα του δίνεται η δυνατότητα να επιλέξει μέσα από μια λίστα δημοπρασιών που είναι ενεργές εκείνη την στιγμή σε ποιο όχημα θα μπορεί να προσθέσει την δική του προσφορά. Επίσης ο κάθε χρήστης μπορεί να μπει στο προφίλ του και να προσθέσει την δική του δημοπρασία για το όχημα που θέλει να πουλήσει. Με την δημιουργία της δημοπρασίας μπορεί να ορίσει ποιος θα είναι ο χρόνος που θα διαρκέσει η δημοπρασία και ποια θα είναι η τελικά τιμή που θέλει να πουλήσει το όχημα του. Ένα ακόμα χαρακτηριστικό είναι πως ο καθένας που πραγματοποιεί την σύνδεση του στην πλατφόρμα θα μπορεί να κάνει αναζήτηση το όνομα του οχήματος που θέλει. Μπορεί επίσης να προσθέσει φίλτρα και να δει ποια δημοπρασία έχει λήξη και ποια βρίσκεται προς το τέλος της. Επιπλέον μπορεί να δει ποια δημοπρασία είναι ενεργή και έχει το μεγαλύτερο περιθώριο χρόνου για να λήξη. Στην εικόνα 12 έχουμε την προσθήκη Authorization στους χρήστες της εφαρμογής μέσο του Duendy Software.

```
src > IdentityService > Models > ApplicationUser.cs > ...
1   // Copyright (c) Duende Software. All rights reserved.
2   // See LICENSE in the project root for license information.
3
4
5   using Microsoft.AspNetCore.Identity;
6
7   namespace IdentityService.Models;
8
9   // Add profile data for application users by adding properties to the ApplicationUser class
    0 references
10  public class ApplicationUser : IdentityUser
11  {
12  }
13
```

Εικόνα 15 : Προσθήκη Authorization στους χρήστες μέσο του Duende

3.2 Δομή της βάσης δεδομένων

Η εφαρμογή στηρίζεται πάνω σε δύο βάσεις δεδομένων στις οποίες θα αποθηκεύονται οι δημοπρασίες και τα item που θέλουν να πουληθούν. Μέσο της βάσης δεδομένων η πλατφόρμα μπορεί να έχει την σωστή δομή και να μην γίνουν λάθη στην δομή του κώδικα. Αρχικά υπάρχουν δύο βασικά tables Auction και Items. Το καθένα από αυτά κάνει μια ξεχωριστή δουλεία. Ο πρώτος πίνακας περιέχει την δημοπρασία που θέλει να φτιάξει ο κάθε user επίσης περιέχει τα απαραίτητα στοιχεία που χρειάζεται για να ξεκινήσει η δημοπρασία. Επιπλέον είναι ο πίνακας με τα items συνδέεται με τον πίνακα Auction και με βάση το item που θέλουμε να πουλήσουμε υπάρχουν τα στοιχεία του στην βάση δεδομένων. Επίσης η κατάσταση της κάθε δημοπρασίας αποθηκεύει τα απαραίτητα στοιχεία στην βάση δεδομένων.



Εικόνα 16 : Πίνακας Auctions

Στην εικόνα 16 βλέπουμε πως είναι σχεδιασμένος ο πίνακας των δημοπρασιών. Ο πίνακας περιλαμβάνει ένα μοναδικό id της κάθε δημοπρασίας, ένα reserve price όπου είναι η τελική τιμή που θέλουμε να έχει το όχημα μας. Τα στοιχεία του seller δηλαδή του user που θέλει να πουλήσει το όχημα. Επίσης περιέχει τα στοιχεία του user ο οποίος νίκησε στην δημοπρασία την τιμή στην οποία πουλήθηκε το όχημα, την μεγαλύτερη τιμή που υπάρχει μέχρι στιγμής στην δημοπρασία ως προσφορά και την ημέρα που έχει δημιουργηθεί η δημοπρασία. Επίσης περιλαμβάνει την ημέρα την οποία έχουμε θέσει ως τελική ημερομηνία αλλά και την ημερομηνία που κάνει update για κάθε φορά που γίνεται μια προσφορά.

	Id uuid	ReservePrice integer	Seller text	Winner text	SoldAmount integer	CurrentHighBid integer	CreatedAt timestamp with time zone	UpdatedAt timestamp with time zone	AuctionEnd timestamp with time zone	Status integer
1	155225c1-4448-4066-9886-6786536e05ea	50000	tom	*null*	*null*	*null*	2024-03-15 14:20:19.803958+00	2024-03-15 14:20:19.803958+00	2024-03-05 14:20:19.803958+00	2
2	3659ac24-29dd-407a-81f5-ecfe6f924b9b	20000	bob	*null*	*null*	*null*	2024-03-15 14:20:19.803963+00	2024-03-15 14:20:19.803963+00	2024-05-02 14:20:19.803963+00	0
3	40490065-dac7-46b6-acc4-df507e0d6570	20000	tom	*null*	*null*	*null*	2024-03-15 14:20:19.803963+00	2024-03-15 14:20:19.803963+00	2024-04-04 14:20:19.803963+00	0
4	466e4744-4dc5-4987-aae0-b621acfc5e39	20000	alice	*null*	*null*	*null*	2024-03-15 14:20:19.803958+00	2024-03-15 14:20:19.803958+00	2024-04-14 14:20:19.803959+00	0
5	47111973-d176-4feb-848d-0ea22641c31a	150000	alice	*null*	*null*	*null*	2024-03-15 14:20:19.803962+00	2024-03-15 14:20:19.803962+00	2024-03-28 14:20:19.803962+00	0
6	6a5011a1-fe1f-47df-9a32-b5346b289391	0	bob	*null*	*null*	*null*	2024-03-15 14:20:19.803962+00	2024-03-15 14:20:19.803962+00	2024-04-03 14:20:19.803962+00	0
7	afbee524-5972-4075-8800-7d1f9d7b0a0c	20000	bob	*null*	*null*	*null*	2024-03-15 14:20:19.803575+00	2024-03-15 14:20:19.803575+00	2024-03-25 14:20:19.803728+00	0
8	bbab4d5a-8565-48b1-9450-5ac2a5c4a654	0	bob	*null*	*null*	*null*	2024-03-15 14:20:19.803957+00	2024-03-15 14:20:19.803957+00	2024-03-19 14:20:19.803958+00	0
9	c8c3ec17-01bf-49db-82aa-1ef80b833a9f	90000	alice	*null*	*null*	*null*	2024-03-15 14:20:19.803954+00	2024-03-15 14:20:19.803954+00	2024-05-14 14:20:19.803956+00	0
10	dc1e4071-d19d-459b-b848-b5c3cd3d151f	20000	bob	*null*	*null*	*null*	2024-03-15 14:20:19.803961+00	2024-03-15 14:20:19.803961+00	2024-04-29 14:20:19.803961+00	0

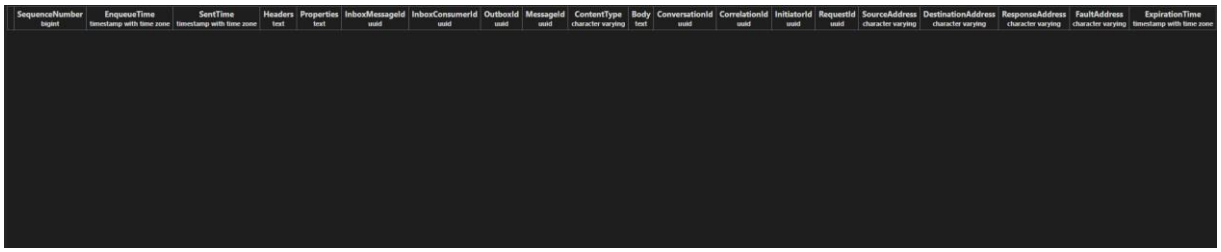
Εικόνα 17 : Πίνακας Auctions με στοιχεία της δημοπρασίας

Στην παραπάνω εικόνα (Εικόνα 17) βλέπουμε πως είναι ένας πίνακας όταν δημιουργείται μια δημοπρασία. Επίσης στον πίνακα Auctions βλέπουμε ότι η κάθε δημοπρασία συνδέεται με ένα id όπου απευθύνεται σε κάθε διαφορετικό item που υπάρχει στον πίνακα. Στην επόμενη εικόνα βλέπουμε πως είναι ο πίνακας item (Εικόνα 18).

	Id uuid	Make text	Model text	Year integer	Color text	Mileage integer	ImageUrl text	AuctionId uuid
1	05cf64a2-ccfe-49d9-ba90-a760eaaa2187	Ferrari	Spider	2015	Red	50000	https://cdn.pixabay.com/photo/2017/11/09/01/49/ferrari-458-spider-2932191_960_720.jpg	dc1e4071-d19d-459b-b848-b5c3cd3d151f
2	3b9daad9-d51d-49f9-8a24-c554c660960f	BMW	X1	2017	White	90000	https://cdn.pixabay.com/photo/2017/08/31/05/47/bmw-2699538_960_720.jpg	466e4744-4dc5-4987-aae0-b621acfc5e39
3	6adf88d4-94fa-4732-b70a-e488f2f50ea9	Ford	Model T	1938	Rust	150150	https://cdn.pixabay.com/photo/2017/08/02/19/47/vintage-2573090_960_720.jpg	3659ac24-29dd-407a-81f5-ecfe6f924b9b
4	7c45c751-acb6-4e7b-98c1-32d4a62b5b57	Ford	Mustang	2023	Black	65125	https://cdn.pixabay.com/photo/2012/11/02/13/02/car-63930_960_720.jpg	bbab4d5a-8565-48b1-9450-5ac2a5c4a654
5	7f885483-8b06-4efe-8ccd-92170ef0c2de	Mercedes	SLK	2020	Silver	15001	https://cdn.pixabay.com/photo/2016/04/17/22/10/mercedes-benz-1335674_960_720.png	155225c1-4448-4066-9886-6786536e05ea
6	bbbf5d52-002a-4d80-a3a1-1a61552199a2	Audi	TT	2020	Black	25400	https://cdn.pixabay.com/photo/2016/09/01/15/06/audi-1636320_960_720.jpg	40490065-dac7-46b6-acc4-df507e0d6570
7	bd8e831e-ee45-47a8-8424-6944d27dbc54	Bugatti	Veyron	2018	Black	15035	https://cdn.pixabay.com/photo/2012/05/29/00/43/car-49278_960_720.jpg	c8c3ec17-01bf-49db-82aa-1ef80b833a9f
8	cbf3ad7e-bc20-41a7-aa72-33ab5bb15584	Ford	GT	2020	White	50000	https://cdn.pixabay.com/photo/2016/05/06/16/32/car-1376190_960_720.jpg	afbee524-5972-4075-8800-7d1f9d7b0a0c
9	d29e1a01-8fe6-48da-a1c7-b59dde67b2d3	Audi	R8	2021	White	10050	https://cdn.pixabay.com/photo/2019/12/26/20/50/audi-r8-4721217_960_720.jpg	6a5011a1-fe1f-47df-9a32-b5346b289391
10	fc5e351a-a7af-41e6-8ee8-1f1482804a1e	Ferrari	F-430	2022	Red	5000	https://cdn.pixabay.com/photo/2017/11/08/14/39/ferrari-f430-2930661_960_720.jpg	47111973-d176-4feb-848d-0ea22641c31a

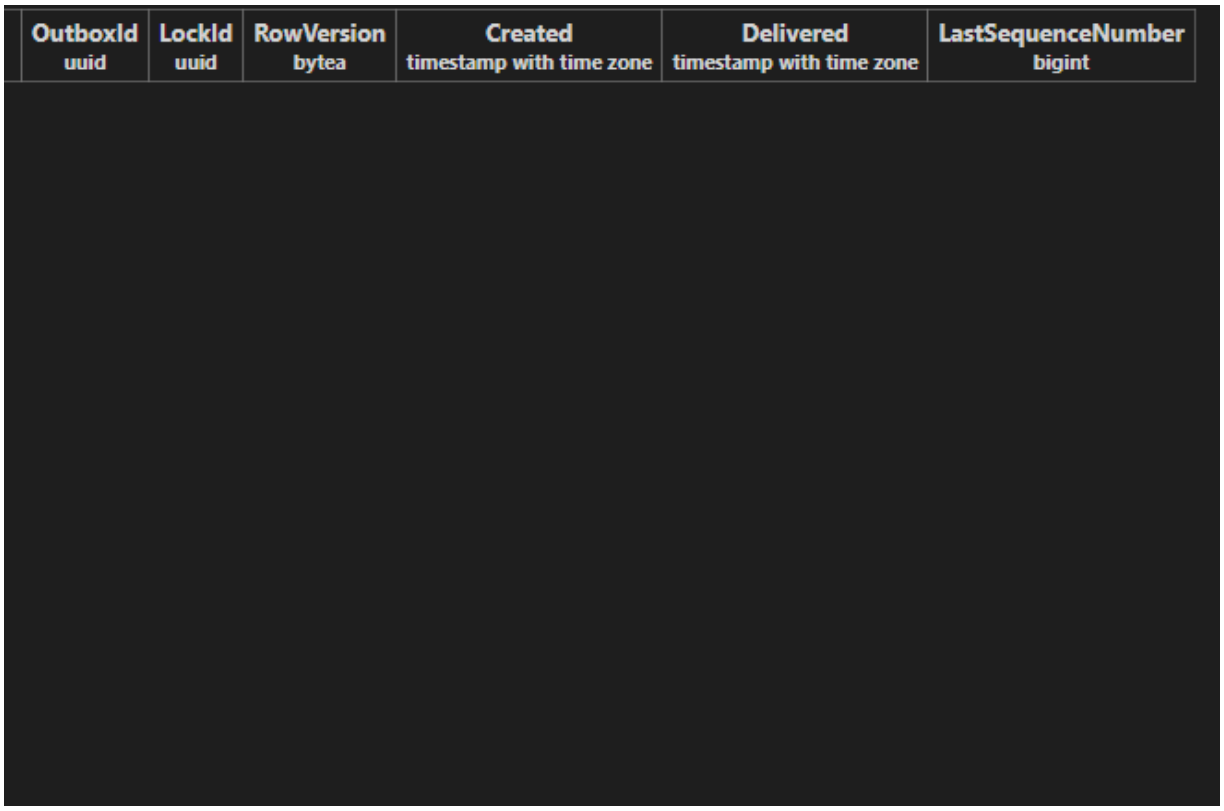
Εικόνα 18 : Πίνακας Items με τα στοιχεία από τα οχήματα

Ο πίνακας items περιλαμβάνει τα στοιχεία από το κάθε όχημα που θέλει κάποιος να πουλήσει όπως χρώμα, μοντέλο, χρονολογία, χιλιόμετρα επιπλέον ο χρήστης μπορεί να προσθέσει φωτογραφίες του οχήματος. Ακόμα υπάρχουν τα εισερχόμενα και εξερχόμενα μηνύματα όπου από εκεί μπορεί ο admin να δει σε ποίον έχει σταλέι ένα μήνυμα και δεν το έχει απαντήσει ή το έχει απαντήσει και ποια χρονολογική στιγμή (Εικόνα 19).



Εικόνα 19 : Πίνακας OutboxMesage

Τέλος δημιουργείται ο πίνακας OutboxState όπου δείχνει την κατάσταση των μηνυμάτων που έχει στείλει ο κάθε χρήστης (Εικόνα 20).



Εικόνα 20 : Πίνακας OutboxState

3.3 Controllers

Οι Controllers σε ένα Web Application είναι οι διαχειριστές όλου του interface της εφαρμογής. Στην ουσία είναι ο συνδετικός κρίκος μεταξύ των Models που θα αναφέρουμε και παρακάτω. Ο τρόπος λειτουργίας τους είναι διαφορετικός από τον συνηθισμένο. Όταν κάποιος χρήστης εισέρχεται στην σελίδα ή κάνει κάποια αλληλεπίδραση οι controllers είναι υπεύθυνοι για την διαχείριση των αιτημάτων (request) που προέρχονται από τους χρήστες (users – clients) αλλά και για την επιστροφή των κατάλληλων απαντήσεων.

Σε ένα API σύστημα οι controllers δέχονται αιτήσεις (requests), όταν ένας χρήστης ή ένα σύστημα κάνει ένα αίτημα στο API (π.χ., για να δημιουργήσει έναν νέο πόρο, να διαβάσει δεδομένα, να ενημερώσει έναν υπάρχοντα πόρο, ή να διαγράψει έναν πόρο), το αίτημα αυτό αποστέλλεται στον κατάλληλο controller. Ο controller αναλύει το αίτημα και καλεί τις κατάλληλες υπηρεσίες ή λειτουργίες για να εκπληρώσει το αίτημα. Αυτό μπορεί να περιλαμβάνει επικοινωνία με τη βάση δεδομένων, εκτέλεση επιχειρηματικής λογικής, ή άλλες λειτουργίες. Επιστρέφουν Απαντήσεις (Responses) αφού επεξεργαστεί το αίτημα, ο controller επιστρέφει μια απάντηση στον χρήστη. Η απάντηση αυτή μπορεί να περιλαμβάνει δεδομένα (π.χ., τα αποτελέσματα μιας ερώτησης στη βάση δεδομένων) ή πληροφορίες σχετικά με την κατάσταση της αίτησης (π.χ., αν η δημιουργία ενός νέου πόρου ήταν επιτυχής).

Παράδειγμα Χρήσης Controllers σε ένα API

Μπορούμε να πούμε ότι υπάρχει ένα API σύστημα για την διαχείριση ενός καταστήματος βιβλίων. Μπορούμε να δημιουργήσουμε ένα controller για τα βιβλία, ένα για τους συγγραφείς και ένα για τις παραγγελίες. Ο controller για τα βιβλία μπορεί να περιέχει τις εξής μεθόδους όπως: createBook (POST /books), getBooks (GET /books) Ανακτά μια λίστα με όλα τα βιβλία. getBookById (GET /books/{id}) Ανακτά τις πληροφορίες για ένα συγκεκριμένο βιβλίο. updateBook (PUT /books/{id}) Ενημερώνει τις πληροφορίες ενός συγκεκριμένου βιβλίου. deleteBook (DELETE /books/{id}) Διαγράφει ένα συγκεκριμένο βιβλίο.

Πλεονεκτήματα της Χρήσης Controllers

- **Οργάνωση Κώδικα:** Οι controllers βοηθούν στον διαχωρισμό των λειτουργιών που σχετίζονται με την επεξεργασία αιτημάτων από την επιχειρηματική λογική και τη διαχείριση δεδομένων.
- **Ευκολία Συντήρησης:** Ο κώδικας είναι πιο δομημένος και ευκολότερος στη συντήρηση και την επέκταση.
- **Επαναχρησιμοποίηση Κώδικα:** Οι λειτουργίες μπορούν να επαναχρησιμοποιηθούν σε διαφορετικά σημεία του συστήματος.
- **Διαχωρισμός Ευθυνών:** Κάθε controller έχει σαφείς ευθύνες, γεγονός που καθιστά το σύστημα πιο καθαρό και κατανοητό.

Στην παρακάτω εικόνα 21 θα δούμε πως είναι δομημένος ο controller για την δημιουργία μιας δημοπρασίας.

```
namespace AuctionService.Controllers;

[ApiController]
[Route("api/auctions")]
1 reference
public class AuctionsController : ControllerBase
{
    10 references
    private readonly IAuctionRepository _repo;
    5 references
    private readonly IMapper _mapper;
    4 references
    private readonly IPublishEndpoint _publishEndpoint;

    0 references
    public AuctionsController(IAuctionRepository repo, IMapper mapper,
        IPublishEndpoint publishEndpoint)
    {
        _repo = repo;
        _mapper = mapper;
        _publishEndpoint = publishEndpoint;
    }

    [HttpGet]
    0 references
    public async Task<ActionResult<List<AuctionDto>>> GetAllAuctions(string date)
    {
        return await _repo.GetAuctionsAsync(date);
    }

    [HttpGet("{id}")]
    1 reference
    public async Task<ActionResult<AuctionDto>> GetAuctionById(Guid id)
    {
        var auction = await _repo.GetAuctionByIdAsync(id);

        if (auction == null) return NotFound();

        return auction;
    }
```

Εικόνα 21 : Δημιουργία GET μεθόδου

Στην εικόνα 21 βλέπουμε πως με την μέθοδο Get το api σύστημα επιστρέφει όλες τις δημοπρασίες με βάση την ημερομηνία που θέτουμε στο σύστημα. Επιπλέον έχουμε προσθέσει και μια επιπλέον μέθοδο Get όπου με βάση το id του κάθε χρήστη σου δείχνει ποιες δημοπρασίες έχει στην βάση δεδομένων του. Στην περίπτωση που δεν υπάρχει ο χρήστης μας εμφανίζει user Not found.

```
[Authorize]
[HttpPost]
0 references
public async Task<ActionResult<AuctionDto>> CreateAuction(CreateAuctionDto auctionDto)
{
    var auction = _mapper.Map<Auction>(auctionDto);

    auction.Seller = User.Identity.Name;

    _repo.AddAuction(auction);

    var newAuction = _mapper.Map<AuctionDto>(auction);

    await _publishEndpoint.Publish(_mapper.Map<AuctionCreated>(newAuction));

    var result = await _repo.SaveChangesAsync();

    if (!result) return BadRequest("Could not save changes to the DB");

    return CreatedAtAction(n (local variable) AuctionDto newAuction
        new { auction.Id }, newAuction);
}

[Authorize]
[HttpPut("{id}")]
0 references
public async Task<ActionResult> UpdateAuction(Guid id, UpdateAuctionDto updateAuctionDto)
{
    var auction = await _repo.GetAuctionEntityById(id);

    if (auction == null) return NotFound();

    if (auction.Seller != User.Identity.Name) return Forbid();

    auction.Item.Make = updateAuctionDto.Make ?? auction.Item.Make;
    auction.Item.Model = updateAuctionDto.Model ?? auction.Item.Model;
    auction.Item.Color = updateAuctionDto.Color ?? auction.Item.Color;
    auction.Item.Mileage = updateAuctionDto.Mileage ?? auction.Item.Mileage;
    auction.Item.Year = updateAuctionDto.Year ?? auction.Item.Year;

    await _publishEndpoint.Publish(_mapper.Map<AuctionUpdated>(auction));

    var result = await _repo.SaveChangesAsync();

    if (result) return Ok();

    return BadRequest("Problem saving changes");
}
```

Εικόνα 22 : Μέθοδος POST και PUT

Στην εικόνα 22 παρατηρούμε πως έχουμε δημιουργήσει δύο μεθόδους POST και PUT, στην μέθοδο POST φτιάχνουμε μια δημοπρασία με τα απαραίτητα στοιχεία που χρειάζονται για να είναι η δημοπρασία επιτυχής. Στην περίπτωση που δεν έχουμε συμπληρώσει όλα τα στοιχεία της δημοπρασίας βλέπουμε πως μας εμφανίζει ένα error δηλαδή ένα Bad Request το οποίο μας εμφανίζει το μήνυμα “Could not save to the DB”. Επίσης αν η δημοπρασία είναι σωστή μας επιστρέφει τα δεδομένα της δημοπρασίας.

Η επόμενη μέθοδος είναι η μέθοδος PUT μέσα από την οποία μπορούμε να αλλάξουμε τα στοιχεία της κάθε δημοπρασίας με βάση το id. Δηλαδή μόνο αν υπάρχει το id της δημοπρασίας μπορούμε να αλλάξουμε τα δεδομένα της για παράδειγμα αν το id = 1 και υπάρχει αυτή η δημοπρασία μπορούμε να αλλάξουμε τα στοιχεία της. Στην περίπτωση που δεν υπάρχει το συγκεκριμένο id που θέλουμε να ψάξουμε μας εμφανίζει ένα μήνυμα από Bad Request.

Το ίδιο κάνουμε και για την τελευταία μέθοδο που χρειάζεται ένα σύστημα api, την μέθοδο delete. Στην παρακάτω εικόνα 23 με την μέθοδο delete και με βάση το id της κάθε δημοπρασίας μπορούμε να διαγράψουμε όποια δημοπρασία θέλουμε με βάση τις δημοπρασίες που έχει την κατοχή του ο κάθε user.

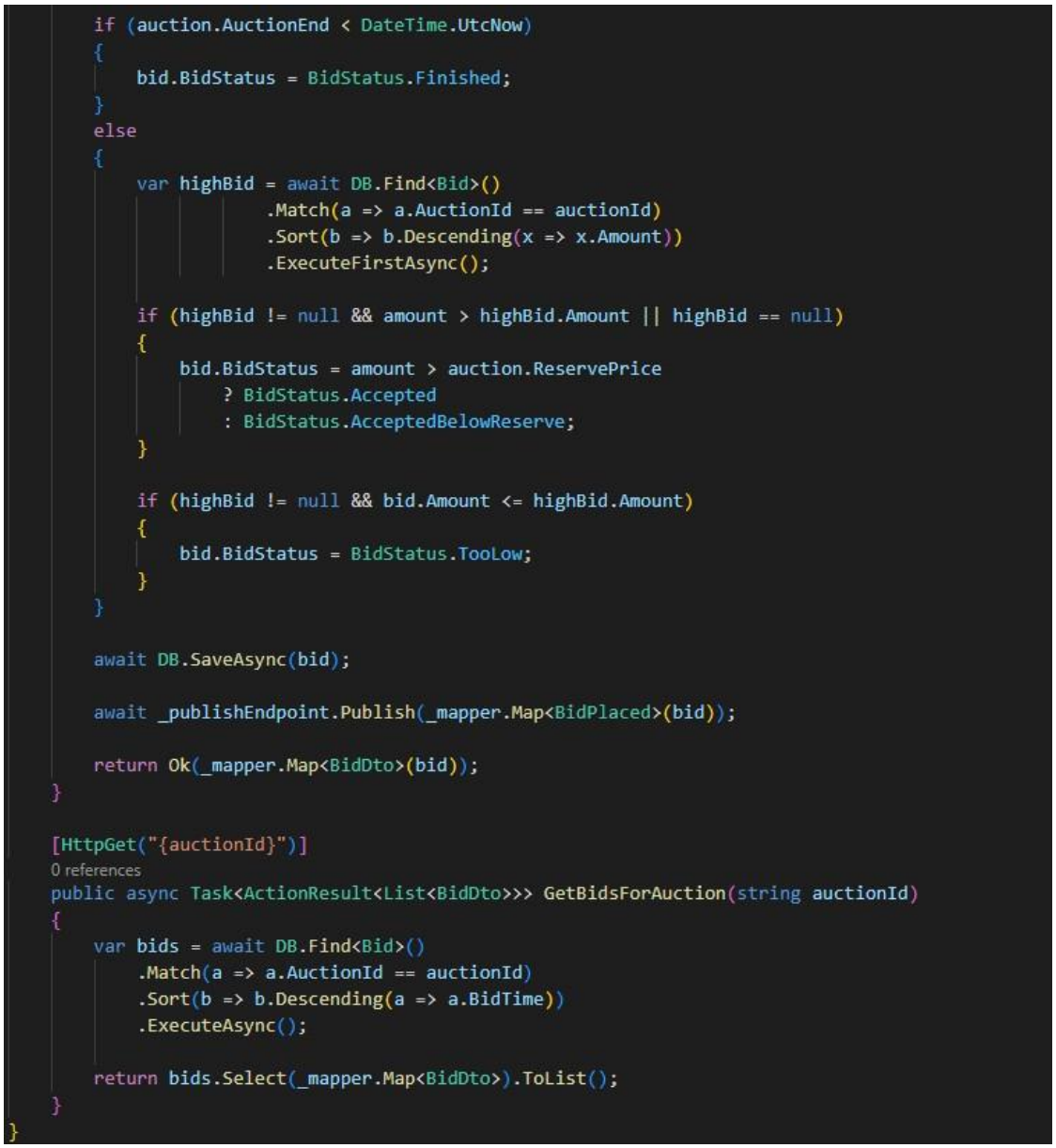
```
    [Authorize]
    [HttpDelete("{id}")]
    0 references
    public async Task<ActionResult> DeleteAuction(Guid id)
    {
        var auction = await _repo.GetAuctionEntityById(id);

        if (auction == null) return NotFound();

        if (auction.Seller != User.Identity.Name) return Forbid();

        _repo.RemoveAuction(auction);

        await _publishEndpoint.Publish<AuctionDeleted>(new { Id = auction.Id.ToString() });

        var result = await _repo.SaveChangesAsync();

        if (!result) return BadRequest("Could not update DB");

        return Ok();
    }
}
```

Εικόνα 23: Μεθοδος Delete

Στην συνέχεια θα δημιουργήσουμε το σύστημα στο όποιο ο κάθε χρήστης θα μπορεί να κάνει την δίκη του προσφορά στις δημοπρασίες οι οποίες είναι ενεργές βλέπουμε εικόνα 24 και 25.

```
[ApiController]
[Route("api/[controller]")]
1 reference
public class BidsController : ControllerBase
{
    4 references
    private readonly IMapper _mapper;
    2 references
    private readonly IPublishEndpoint _publishEndpoint;
    2 references
    private readonly GrpcAuctionClient _grpcClient;

    0 references
    public BidsController(IMapper mapper, IPublishEndpoint publishEndpoint,
        GrpcAuctionClient grpcClient)
    {
        _mapper = mapper;
        _publishEndpoint = publishEndpoint;
        _grpcClient = grpcClient;
    }

    [Authorize]
    [HttpPost]
    0 references
    public async Task<ActionResult<BidDto>> PlaceBid(string auctionId, int amount)
    {
        var auction = await DB.Find<Auction>().OneAsync(auctionId);

        if (auction == null)
        {
            auction = _grpcClient.GetAuction(auctionId);

            if (auction == null) return BadRequest("Cannot accept bids on this auction at this time");
        }

        if (auction.Seller == User.Identity.Name)
        {
            return BadRequest("You cannot bid on your own auction");
        }

        var bid = new Bid
        {
            Amount = amount,
            AuctionId = auctionId,
            Bidder = User.Identity.Name
        };
```

Εικόνα 24: Δημιουργία POST μεθόδου για τις προσφορές



```
        if (auction.AuctionEnd < DateTime.UtcNow)
        {
            bid.BidStatus = BidStatus.Finished;
        }
        else
        {
            var highBid = await DB.Find<Bid>()
                        .Match(a => a.AuctionId == auctionId)
                        .Sort(b => b.Descending(x => x.Amount))
                        .ExecuteFirstAsync();

            if (highBid != null && amount > highBid.Amount || highBid == null)
            {
                bid.BidStatus = amount > auction.ReservePrice
                    ? BidStatus.Accepted
                    : BidStatus.AcceptedBelowReserve;
            }

            if (highBid != null && bid.Amount <= highBid.Amount)
            {
                bid.BidStatus = BidStatus.TooLow;
            }
        }

        await DB.SaveAsync(bid);

        await _publishEndpoint.Publish(_mapper.Map<BidPlaced>(bid));

        return Ok(_mapper.Map<BidDto>(bid));
    }

    [HttpGet("{auctionId}")]
    0 references
    public async Task<ActionResult<List<BidDto>>> GetBidsForAuction(string auctionId)
    {
        var bids = await DB.Find<Bid>()
            .Match(a => a.AuctionId == auctionId)
            .Sort(b => b.Descending(a => a.BidTime))
            .ExecuteAsync();

        return bids.Select(_mapper.Map<BidDto>).ToList();
    }
}
```

Εικόνα 25: Δημιουργία Get μεθόδου για τις προσφορές

Στην εικόνα 24 και 25 δημιουργούμε την μέθοδο POST και GET. Με την μέθοδο POST μπορούμε να κάνουμε προσφορά στις δημοπρασίες οι οποίες είναι ενεργές. Για να εγκριθεί όμως η προσφορά κάθε φορά που ο χρήστης κάνει μια περνάει μέσα από μια διαδικασία ελέγχου έτσι ώστε να καλύπτει τα κατάλληλα κριτήρια. Το ίδιο γίνεται και για την μέθοδο GET κάθε φορά που γίνεται η προσφορά εμφανίζει στον ιδιοκτήτη της δημοπρασίας σε ποια δημοπρασία έχει γίνει η προσφορά με τα κατάλληλα στοιχεία.

Τέλος έχουμε δημιουργήσει τον SearchController μέσο του οποίου μπορούμε να αναζητήσουμε τα ονόματα των οχημάτων που βρίσκονται στην εφαρμογή βλέπουμε εικόνα 26.

```
public class SearchController : ControllerBase
{
    [HttpGet]
    0 references
    public async Task<ActionResult<List<Item>>> SearchItems([FromQuery] SearchParams searchParams)
    {
        var query = DB.PagedSearch<Item, Item>();

        if (!string.IsNullOrEmpty(searchParams.SearchTerm))
        {
            query.Match(Search.Full, searchParams.SearchTerm).SortByTextScore();
        }

        query = searchParams.OrderBy switch
        {
            "make" => query.Sort(x => x.Ascending(a => a.Make))
                .Sort(x => x.Ascending(a => a.Model)),
            "new" => query.Sort(x => x.Descending(a => a.CreatedAt)),
            _ => query.Sort(x => x.Ascending(a => a.AuctionEnd))
        };

        query = searchParams.FilterBy switch
        {
            "finished" => query.Match(x => x.AuctionEnd < DateTime.UtcNow),
            "endingSoon" => query.Match(x => x.AuctionEnd < DateTime.UtcNow.AddHours(6)
                && x.AuctionEnd > DateTime.UtcNow),
            _ => query.Match(x => x.AuctionEnd > DateTime.UtcNow)
        };

        if (!string.IsNullOrEmpty(searchParams.Seller))
        {
            query.Match(x => x.Seller == searchParams.Seller);
        }

        if (!string.IsNullOrEmpty(searchParams.Winner))
        {
            query.Match(x => x.Winner == searchParams.Winner);
        }

        query.PageNumber(searchParams.PageNumber);
        query.PageSize(searchParams.PageSize);

        var result = await query.ExecuteAsync();

        return Ok(new
        {
            results = result.Results,
            pageCount = result.PageCount,
            totalCount = result.TotalCount
        });
    }
}
```

Εικόνα 26. Δημιουργία Get για τον SearchController

3.4 Mores

Τα models είναι αντικείμενα ή δομές δεδομένων που αντιπροσωπεύουν τις βασικές οντότητες της εφαρμογής. Κάθε model αντικατοπτρίζει συνήθως έναν πίνακα στη βάση δεδομένων και περιγράφει τις ιδιότητες και τις σχέσεις των δεδομένων που θα χρησιμοποιηθούν από την εφαρμογή. Στο ASP.NET, τα models είναι κλάσεις C# που αντιπροσωπεύουν τις οντότητες της εφαρμογής. Αυτές οι κλάσεις χρησιμοποιούνται για να μοντελοποιήσουν τα δεδομένα που θα αποθηκευτούν στη βάση δεδομένων και θα ανταλλαχθούν μέσω του API.

Σκοπός των models είναι να οργανώνουν και να διαχειρίζονται δεδομένα με έναν δομημένο τρόπο. Παρέχουν ένα ενιαίο σημείο πρόσβασης για την αλληλεπίδραση με τα δεδομένα, είτε πρόκειται για αποθήκευση, ανάκτηση, επικύρωση ή μετασχηματισμό δεδομένων.

Ρόλος των Models σε ένα API Project με ASP.NET

1. Δομή Δεδομένων:

Με τα models καθορίζουμε τη δομή των δεδομένων που θα ανταλλάσσονται μεταξύ της εφαρμογής και των χρηστών μέσω του API. Για παράδειγμα, σε ένα API για ένα ηλεκτρονικό κατάστημα, μπορεί να έχεις ένα model για τα προϊόντα, ένα για τους πελάτες, και ένα για τις παραγγελίες.

2. Επικύρωση (Validation):

Τα models συχνά περιέχουν κανόνες επικύρωσης δεδομένων μέσω data annotations. Αυτοί οι κανόνες διασφαλίζουν ότι τα δεδομένα που εισάγονται είναι έγκυρα πριν αποθηκευτούν στη βάση δεδομένων.

3. Σχέσεις (Relationships):

Επίσης τα models μπορούν να καθορίζουν σχέσεις μεταξύ διαφορετικών οντοτήτων. Για παράδειγμα, ένα μοντέλο παραγγελίας μπορεί να έχει σχέση πολλών-προς-ένα (many-to-one) με ένα μοντέλο πελάτη, και σχέση ένα-προς-πολλά (one-to-many) με ένα μοντέλο προϊόντος.

4. Μετασχηματισμός Δεδομένων (Data Transformation):

Τέλος τα models μπορούν να περιέχουν μεθόδους για τον μετασχηματισμό των δεδομένων, όπως η μορφοποίηση ημερομηνιών ή η υπολογισμός τιμών, που διευκολύνουν την εργασία με τα δεδομένα στο API.

Τα models στο ASP.NET αποτελούν κρίσιμο μέρος της ανάπτυξης ενός API project. Προσφέρουν μια οργανωμένη και δομημένη μέθοδο για την εργασία με τα δεδομένα της εφαρμογής, αντιπροσωπεύοντας τις βασικές οντότητες και παρέχοντας τα απαραίτητα εργαλεία για την επικύρωση, τον χειρισμό και τη διαχείριση των δεδομένων. Παρακάτω θα δούμε μερικά παραδείγματα από τα models της εφαρμογής.

```
namespace AuctionService.Entities;

38 references
public class Auction
{
    25 references
    public Guid Id { get; set; }
    15 references
    public int ReservePrice { get; set; } = 0;
    21 references
    public string Seller { get; set; }
    1 reference
    public string Winner { get; set; }
    2 references
    public int? SoldAmount { get; set; }
    3 references
    public int? CurrentHighBid { get; set; }
    0 references
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    4 references
    public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;
    14 references
    public DateTime AuctionEnd { get; set; }
    14 references
    public Status Status { get; set; }
    35 references
    public Item Item { get; set; }

    2 references
    public bool HasReservePrice() => ReservePrice > 0;
}
```

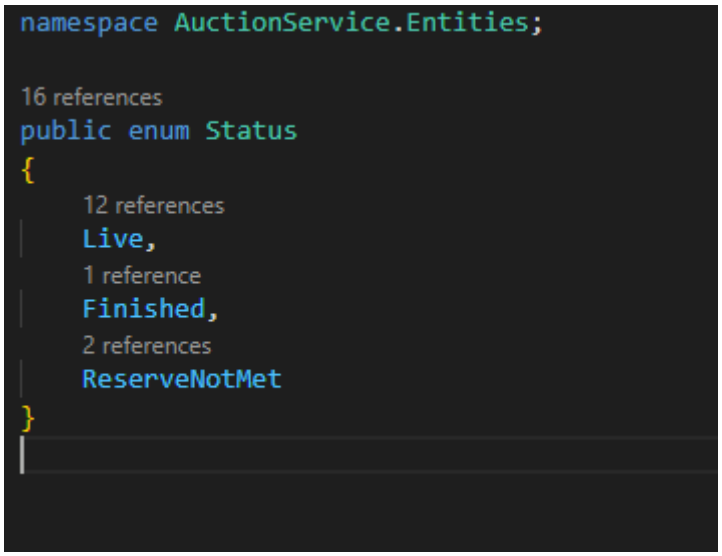
Εικόνα 27: Δημιουργία Auction Model

Στην εικόνα 27 βλέπουμε τα στοιχεία που θέλουμε να έχει η δημοπρασία μέσα στην βάση δεδομένων και πως συνδέονται με τα υπόλοιπα στοιχεία.

```
using System.ComponentModel.DataAnnotations.Schema;
using AuctionService.Entities;

namespace AuctionService;

[Table("Items")]
18 references
public class Item
{
    0 references
    public Guid Id { get; set; }
    16 references
    public string Make { get; set; }
    15 references
    public string Model { get; set; }
    15 references
    public int Year { get; set; }
    15 references
    public string Color { get; set; }
    15 references
    public int Mileage { get; set; }
    13 references
    public string ImageUrl { get; set; }

    // nav properties
    1 reference
    public Auction Auction { get; set; }
    0 references
    public Guid AuctionId { get; set; }
}
```

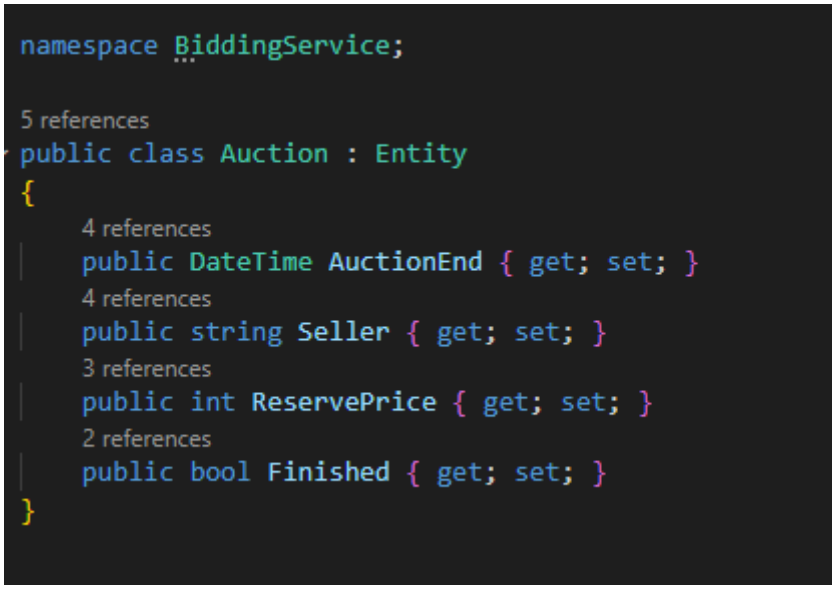
Εικόνα 28: Δημιουργία Item Model

Το ίδιο συμβαίνει και στην εικόνα 28 ονομάζουμε το όνομα του πίνακα Items στην συνέχεια δημιουργούμε την κλάση και προσθέτουμε τα στοιχεία που θέλουμε να έχει το Item. Μέσα στην κλάση προσθέτουμε τα setter και getter και επιπλέον αναφέρουμε ότι το κάθε item συνδέεται με μια μοναδική Auction. Επίσης στην εικόνα 26 φτιάχνουμε ένα model το οποίο σχετίζεται και αυτό με τα Auctions και κάθε φορά που φτιάχνουμε ένα Auction δημιουργήται και το model στην εικόνα 29.



Εικόνα 29: Δημιουργία model Status

Ένα ακόμα service που είναι αρκετά σημαντικό και συνδέεται με το Auction service είναι το Bidding service μέσα από αυτό η κάθε προσφορά που κάνει ο κάθε χρήστης ενημερώνεται και ενημερώνει και τον αντίστοιχο πίνακα.



Εικόνα 30: Δημιουργία και επικοινωνία Auction model

Στην εικόνα 30 δημιουργούμε ένα model Auction το οποίο συνδέεται με το Bidding. Το συγκεκριμένο model περιέχει την ημερομηνία που τελειώνει το Auction, τον πωλητή, το reserve price και αν τελείωσε η δημοπρασία. Το συγκεκριμένο model συνδέεται και με το model Auction στο AuctionService.

```
namespace BiddingService;

6 references
public class Bid : Entity
{
    4 references
    public string AuctionId { get; set; }
    2 references
    public string Bidder { get; set; }
    1 reference
    public DateTime BidTime { get; set; } = DateTime.UtcNow;
    7 references
    public int Amount { get; set; }
    4 references
    public BidStatus BidStatus { get; set; }
}
```

Εικόνα 31: Δημιουργία Bid Model

Στην παραπάνω εικόνα 31 βλέπουμε τα στοιχεία που περιέχει μια προσφορά. Περιέχει το AuctionId στο οποίο γίνεται η προσφορά, τον user που κάνει την προσφορά την ημερομηία και την ώρα που γίνεται, το ποσό αλλά και την κατάσταση της προσφοράς. Το model Bid συνδέεται με τα model Auctions οπότε κάποια από τα στοιχεία συμπληρώνονται αυτόματα.

```
namespace BiddingService;

6 references
public enum BidStatus
{
    2 references
    Accepted,
    1 reference
    AcceptedBelowReserve,
    1 reference
    TooLow,
    1 reference
    Finished
}
```

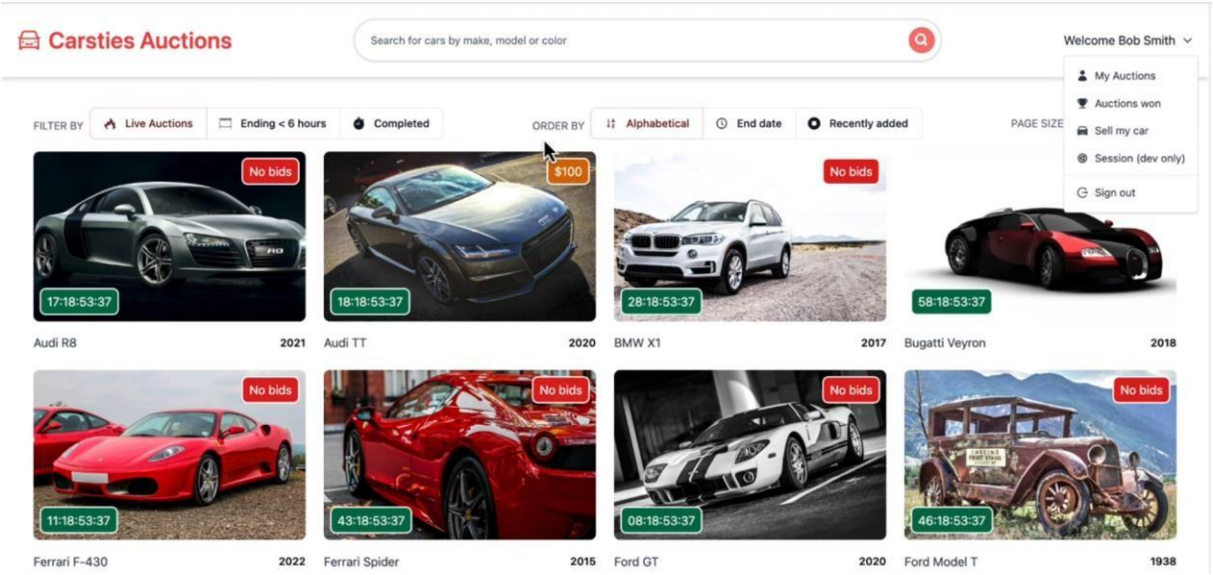
Εικόνα 32: Δημιουργία model BidStatus

Στην εικόνα 32 φτιάχνουμε το model BidStatus μέσο από το οποίο μας δείχνει αν η προσφορά που έχει κάνει ο user έγινε αποδεκτή.

Εικόνα 33: Δημιουργία Item model στο SearchService

Ένα ακόμα πολύ σημαντικό model που πρέπει να προστεθεί στα Services και ποιο συγκεκριμένα στο SearchService είναι το model Item . Μέσα από αυτό ο κάθε user θα μπορεί να αναζητήσει το όχημα που θέλει με βάση το στοιχεία που μπορεί να διαμορφώσει στο Search της εφαρμογής.

3.5 Interface εφαρμογής



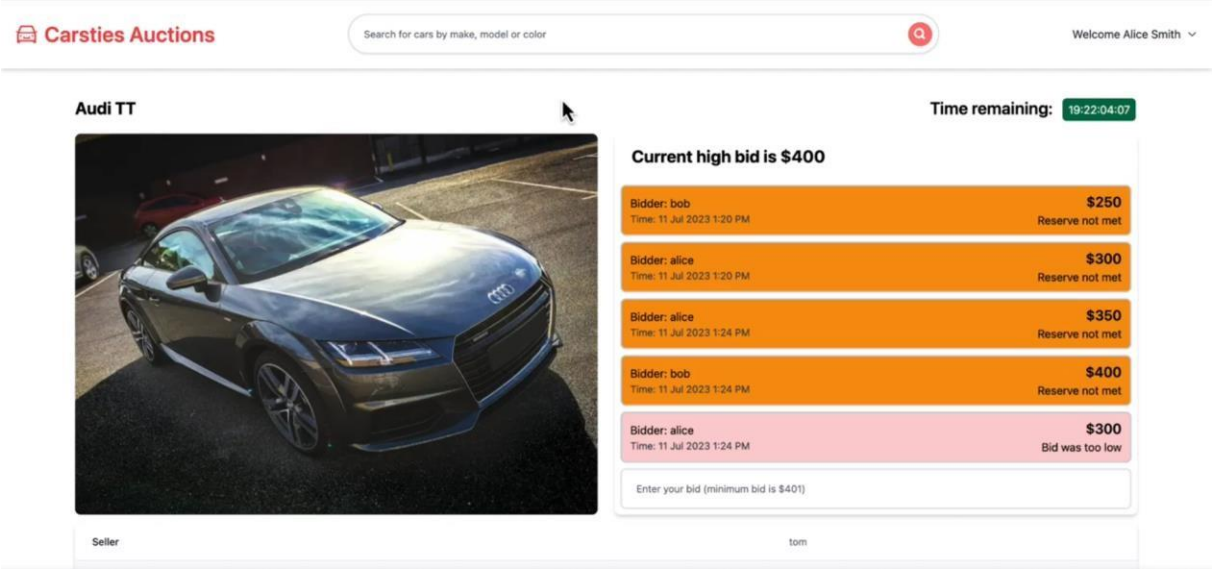
Εικόνα 34: Αρχική σελίδα εφαρμογής

Στην εικόνα 34 βλέπουμε πως είναι η αρχική σελίδα από την στιγμή που συνδεόμαστε στην εφαρμογή. Αρχικά όταν συνδεθούμε παρατηρούμε πως μας εμφανίζει τις δημοπρασίες που είναι ενεργές. Πάνω δεξιά μας καλωσορίζει με το όνομα που έχουμε προσθέσει στην εφαρμογή. Στην συνέχεια αν πατήσουμε στο όνομα μας εμφανίζει κάποια επιπλέον πράγματα που μπορούμε να κάνουμε όπως να δούμε το προφίλ μας, τις δημοπρασίες που έχουμε κερδίσει αλλά και να πουλήσουμε το όχημα μας.

Εικόνα 35: Μπάρα αναζήτησης

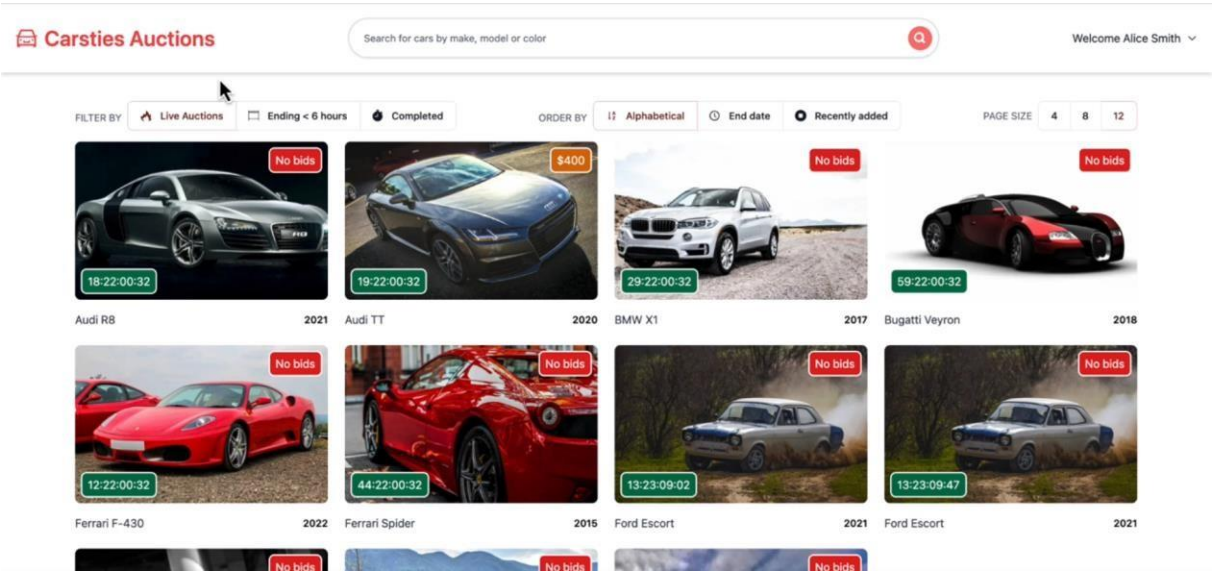
Εικόνα 36: Φίλτρα αναζήτησης

Στις εικόνες 35 και 36 ο χρήστης της εφαρμογής μπορεί να χρησιμοποιήσει την μπάρα αναζήτησης αλλά και τα φίλτρα για να μπορεί να αναζητήσει ποιο εύκολα το όχημα που θέλει.



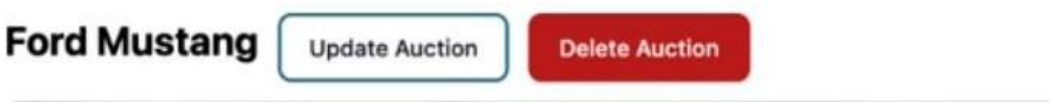
Εικόνα 37: Προσθήκη προσφοράς

Επιπλέον ο κάθε χρήστης μπορεί να μπει στην κάθε δημοπρασία που είναι διαθέσιμη και να κάνει την δική του προσφορά. Στην εικόνα 37 βλέπουμε να έχουν γίνει κάποιες προσφορές σχετικά με το συγκεκριμένο αυτοκίνητο όπου φτάνουν τα 400$. Στην συνέχεια ο χρήστης προσπαθεί να κάνει μία προσφορά η οποία είναι μικρότερη από τις προηγούμενες και παρατηρούμε πως η προσφορά δεν γίνεται δεκτή.

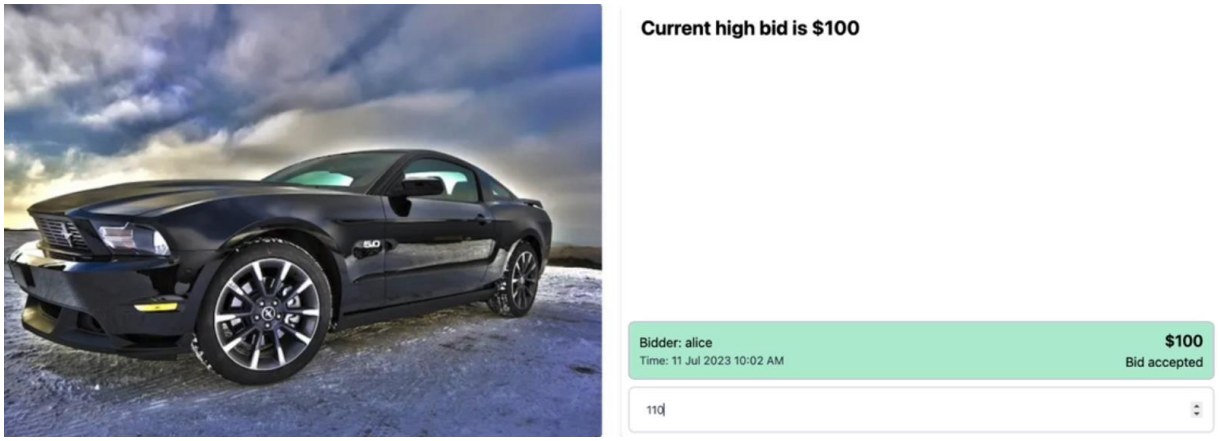


Εικόνα 38: Τωρινή προσφορά στο όχημα

Ένα ακόμα επιπρόσθετο στην εφαρμογή είναι πως ο χρήστης από την στιγμή που μεταβεί στην αρχική σελίδα μπορεί να παρατηρήσει σε ποιες δημοπρασίες έχουν γίνει προσφορές σε όχημα και πως εξελίσσονται οι προσφορές στο συγκεκριμένο όχημα.



Εικόνα 39: Διαγραφή και αναβάθμιση οχήματος

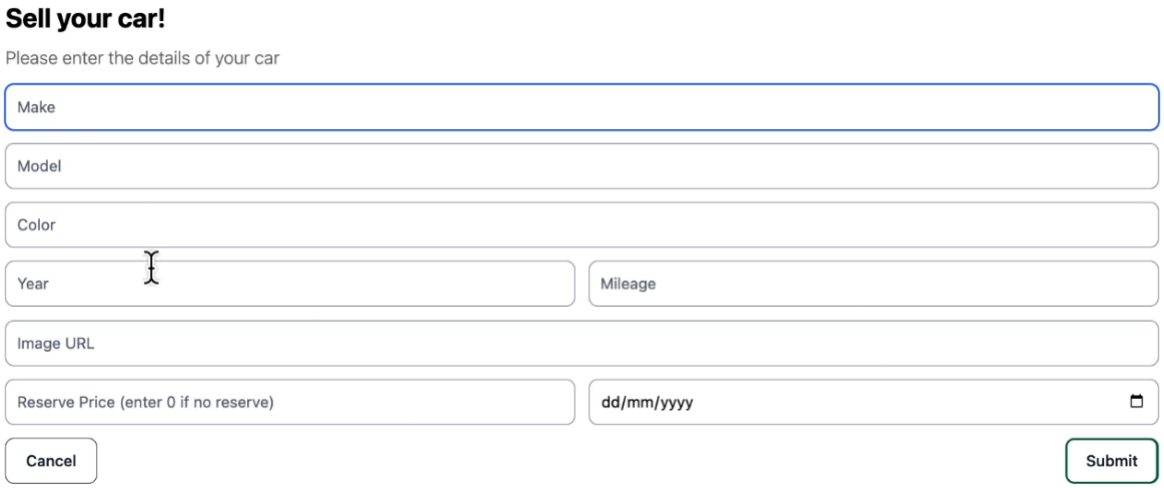


Εικόνα 40: Έγκυρη προσφορά

Επίσης ο χρήστης που έχει προσθέσει το δικό του όχημα για δημοπρασία μπορεί να κάνει αναβάθμιση ή και διαγραφή της δημοπρασίας που έχει στην κατοχή του. Αυτό μας το δείχνει η εικόνα 39. Τέλος ο "ιδιοκτήτης" της δημοπρασίας μπορεί να προσθέσει και ο ίδιος την δική του προσφορά στο όχημα του.

3.6 Δημιουργία Δημοπρασίας

Στις παρακάτω εικόνες θα δούμε πως γίνεται η δημιουργία μιας δημοπρασίας.



Εικόνα 41: Η φόρμα με τα στοιχεία της δημοπρασίας

Στην εικόνα 4` βλέπουμε την φόρμα με τα κενά που πρέπει να συμπληρώσει ο κάθε χρήστης που θέλει να δημιουργήσει μια καινούργια δημοπρασία.

Εικόνα 42: Επιτυχής φόρμα δημοπρασίας

Τέλος στην εικόνα 42 βλέπουμε μια επιτυχημένη συμπλήρωση φόρμας με όλα τα στοιχεία της να είναι επαρκή.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Ζούμε σε μια εποχή όπου οι web εφαρμογές γίνονται ολοένα και πιο περιζήτητες στον χώρο του προγραμματισμού. Γι' αυτό και οι web developers είναι απαραίτητοι για την ανάπτυξη τέτοιων εφαρμογών.

Καθώς υπάρχουν πολλές διαδεδομένες γλώσσες προγραμματισμού που χρησιμοποιούν οι developers, υπήρξε η ανάγκη να δημιουργηθούν εργαλεία που επεκτείνουν τη λειτουργικότητα αυτών των γλωσσών. Παραδείγματα αποτελούν τα frameworks React, Angular και Node.js για τη Javascript. Επίσης, για την Java υπάρχει το framework Spring Boot και για τη C# το framework ASP.NET. Αυτά τα εργαλεία έχουν σκοπό να βοηθήσουν τους προγραμματιστές να δημιουργήσουν εύκολα και γρήγορα web εφαρμογές σε μια γλώσσα που ήδη γνωρίζουν, τόσο στο front-end που αφορά τον σχεδιασμό της σελίδας, όσο και στο back-end που αφορά τη βάση δεδομένων.

Η Microsoft, από την πλευρά της, αποφάσισε να επεκτείνει τη γλώσσα προγραμματισμού C# για να διευκολύνει τη δημιουργία web εφαρμογών. Με το ASP.NET Core, οι προγραμματιστές μπορούν πλέον να αναπτύξουν γρήγορα μια εφαρμογή χρησιμοποιώντας το γνωστό pattern Model View Controller (MVC). Το Core, σε συνδυασμό με άλλα frameworks όπως το Identity Framework για τη διαχείριση συστημάτων μελών και το Entity Framework για τη διαχείριση του back-end, προσφέρει τεράστιες δυνατότητες στους developers για την ανάπτυξη από τις πιο απλές έως και τις πιο περίπλοκες εφαρμογές.

Με την γνώση αυτών των εργαλείων, ένας προγραμματιστής μπορεί να θεωρηθεί full-stack developer, αξιοποιώντας τη δύναμη και την ευελιξία της ισχυρής γλώσσας C#.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] wikipedia. (2024).***C Sharp (programming language)***. Available at: https://en.wikipedia.org/wiki/C_Sharp_(programming_language).

[2] Microsoft, 2024. **Learn C# | Free tutorials, courses, videos, and more**. Available at: https://dotnet.microsoft.com/en-us/learn/csharp .

[3] Dot Net Tutorials, 2024. **C#.NET Tutorials For Beginners and Professionals**. Available at: https://dotnettutorials.net/lesson/csharp-tutorials-for-beginners-and-professionals/

[4] W3Schools, 2024. **C# Tutorial (C Sharp)**. Available at: https://www.w3schools.com/cs/index.php .

[5] GitConnected, 2024. **Learn C# - Best C# Tutorials**. Available at: https://gitconnected.com/learn/csharp .

[6] TutorialsTeacher, 2024. **C# Tutorial**. Available at: https://www.tutorialsteacher.com/csharp

[7] wikipedia (2024)..***Net Framework.*** Available at: https://en.wikipedia.org/wiki/.NET_Framework.

[8] Dot Net Tutorials (2024) **ASP.NET Core Tutorials for Beginners**. Available at: https://dotnettutorials.net/course/asp-net-core-tutorials/

[9] HowtoASP.NET (2024) **ASP.NET Tutorials**. Available at: http://www.howtoasp.net/

[10] TutorialsTeacher (2024) **Learn ASP.NET Core**. Available at: https://www.tutorialsteacher.com/core .

[11] TutorialsPoint (2024) **ASP.NET Tutorial**. Available at: https://www.tutorialspoint.com/asp.net/index.htm .

[12] Microsoft Docs (2024) **ASP.NET Core Documentation**. Available at: https://learn.microsoft.com/en-us/aspnet/core/ .

[13] wikipedia (2024). ***Next.Js.*** Available at: https://en.wikipedia.org/wiki/Next.js.

[14] Next.js Documentation (2024) **Create a Next.js App**. Available at:

https://nextjs.org/learn/basics/create-nextjs-app

[15] Ikius (2024) From Zero to Hero: **11 Best Next.js Tutorials for 2024**. Available at: https://www.ikius.com/blog/nextjs-tutorials

[16] freeCodeCamp (2024) **The Next.js Handbook – Learn Next.js for Beginners**. Available at: https://www.freecodecamp.org/news/the-next-js-handbook/

[17] DEV Community (2024) **A Beginner's Guide to Building Your First Next.js Application: Step-by-Step Tutoria**l. Available at: https://dev.to/franciscomendes10866/a-beginner-s-guide-to-building-your-first-next-js-application-step-by-step-tutorial-2gdh

[18] Net Ninja (2024) **Next.js Tutorial for Beginners**. Available at: https://www.youtube.com/watch?v=A63UxsQsEbU

[19] wikipedia (2024). ***MongoDB.*** Available at:: https://en.wikipedia.org/wiki/MongoDB

[20] MongoDB (2024) **Get Started with MongoDB**. Available at: https://www.mongodb.com/get-started

[21] MongoDB University (2024) **MongoDB Basics**. Available at: https://university.mongodb.com/courses/M001/about

[22] DigitalOcean (2024) **How To Use MongoDB**. Available at: https://www.digitalocean.com/community/tutorial_series/how-to-use-mongodb

[23]freeCodeCamp (2024) **Introduction to MongoDB**. Available at: https://www.freecodecamp.org/news/introduction-to-mongodb/

[24] W3Schools (2024) **MongoDB Tutorial. Available at**: https://www.w3schools.com/mongodb/

[25] wikipedia (2024). ***PostgreSQL*** Available at: https://en.wikipedia.org/wiki/PostgreSQL

[26] PostgreSQL (2024) **Documentation**. Available at: https://www.postgresql.org/docs/ (Accessed: 12 July 2024).
freeCodeCamp (2024) **PostgreSQL Tutorial**. Available at: https://www.freecodecamp.org/news/learn-postgresql-introduction/ (Accessed: 12 July 2024).

[27] DigitalOcean (2024) **How To Use PostgreSQL**. Available at: https://www.digitalocean.com/community/tutorial_series/how-to-use-postgresql

[28] Guru99 (2024) **PostgreSQL Tutorial for Beginners**. Available at: https://www.guru99.com/postgresql-tutorial.html

[29] TutorialsPoint (2024) **PostgreSQL Tutorial**. Available at: https://www.tutorialspoint.com/postgresql/index.htm

[30] wikipedia (2024). ***RabbitMQ***. Available at: https://en.wikipedia.org/wiki/RabbitMQ

[31] RabbitMQ (2024) **Getting Started with RabbitMQ**. Available at: https://www.rabbitmq.com/getstarted.html

[32] DigitalOcean (2024) **How To Use RabbitMQ**. Available at: https://www.digitalocean.com/community/tutorial_series/how-to-use-rabbitmq

[33]CloudAMQP (2024) **RabbitMQ Tutorials**. Available at: https://www.cloudamqp.com/blog.html

[34] freeCodeCamp (2024) **Introduction to RabbitMQ**. Available at: https://www.freecodecamp.org/news/an-introduction-to-rabbitmq/

[35]TutorialsPoint (2024) **RabbitMQ Tutorial**. Available at: https://www.tutorialspoint.com/rabbitmq/index.htm

[36] wikipedia (2024). ***Identity Provider***. Available at: https://en.wikipedia.org/wiki/Identity_provider

[37] Auth0 (2024) **What is an Identity Provider?** Available at: https://auth0.com/docs/authentication/identity-providers

[38] Okta (2024) **Identity Providers and Federation**. Available at: https://www.okta.com/identity-provider/

[39] Ping Identity (2024) **What is an Identity Provider?** Available at: https://www.pingidentity.com/en/company/blog/posts/2020/what-is-an-identity-provider.html

[40] OneLogin (2024) **Identity Provider Explained**. Available at: https://www.onelogin.com/learn/identity-provider

[41] Forgerock (2024) **Identity Providers. Available** at: https://www.forgerock.com/what-we-offer/identity-providers

[42] wikipedia (2024). ***Managment Systerm***. Available at: https://en.wikipedia.org/wiki/#Management_systems

[43] Docker (2024) Get Started with Docker. Available at: https://www.docker.com/get-started

[44] freeCodeCamp (2024) **Docker for Beginners**. Available at: https://www.freecodecamp.org/news/docker-tutorial/

[45] DigitalOcean (2024) **How To Use Docker**. Available at: https://www.digitalocean.com/community/tutorial_series/how-to-use-docker

[46] Kubernetes (2024) **Docker Tutorials**. Available at: https://kubernetes.io

[47] Microsoft, 2023. **ASP.NET Core Documentation**. Available at: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0

[48] Pluralsight, 2023. **ASP.NET Core Fundamentals**. Available at: https://www.pluralsight.com/courses/asp-dot-net-core-fundamentals

[49] Udemy, 2023. **The Complete ASP.NET MVC 5 Course**. Available at: https://www.udemy.com/course/the-complete-aspnet-mvc-5-course/

[50] Docker, 2023. **Docker Documentation**. Available at: https://docs.docker.com/

[51] Udemy, 2023. **Docker Mastery: with Kubernetes +Swarm from a Docker Captain**. Available at: https://www.udemy.com/course/docker-mastery/

[52] Pluralsight, 2023. **Docker Deep Dive**. Available at: https://www.pluralsight.com/courses/docker-deep-dive

[53] RabbitMQ, 2023. **RabbitMQ Documentation**. Available at: https://www.rabbitmq.com/documentation.html

[54] Udemy, 2023. **RabbitMQ: Learn Messaging with RabbitMQ**. Available at: https://www.udemy.com/course/rabbitmq/

[55] Pluralsight, 2023. **RabbitMQ by Example**. Available at: https://www.pluralsight.com/courses/rabbitmq-by-example

[56] PostgreSQL, 2023. **PostgreSQL Documentation**. Available at: https://www.postgresql.org/docs/

[57] Udemy, 2023. **The Complete SQL Bootcamp 2022: Go from Zero to Hero**. Available at: https://www.udemy.com/course/the-complete-sql-bootcamp/

[58] Pluralsight, 2023. **PostgreSQL: Getting Started**. Available at: https://www.pluralsight.com/courses/postgresql-getting-started

[59] MongoDB, 2023. **MongoDB Documentation**. Available at: https://docs.mongodb.com/

[60] Udemy, 2023. **MongoDB - The Complete Developer's Guide 2022**. Available at: https://www.udemy.com/course/mongodb-the-complete-developers-guide/

[61] Pluralsight, 2023. **Introduction to MongoDB**. Available at: https://www.pluralsight.com/courses/introduction-to-mongodb

[Οπισθόφυλλο. Κενή σελίδα]