ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή Android - Service Book

Του φοιτητή
Πομπόρτση Γεώργιου
Αρ. Μητρώου: 154526

Επιβλέπων
Κεραμόπουλος Ευκλείδης
Αναπληρωτής Καθηγητής

Ημερομηνία 15/06/2021

Τίτλος Δ.Ε. Service Book
Κωδικός Δ.Ε. 20188
Ονοματεπώνυμο φοιτητή Πομπόρτσης Γιώργος.
Ονοματεπώνυμο εισηγητή Κεραμόπουλος Ευκλείδης
Ημερομηνία ανάληψης Δ.Ε. 17/09/2020
Ημερομηνία περάτωσης Δ.Ε.15/06/2020

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

Στη μνήμη

του πατέρα μου Ανδρέα

Πρόλογος

Αντικείμενο της παρούσας εργασίας είναι μια εφαρμογή android μέσω της οποίας θα καταγράφονται οι αλλαγές που έχουν πραγματοποιηθεί σε ένα αυτοκίνητο. Η ψηφιοποίηση των εγγράφων και η απλοποίηση των διαδικασιών είναι βασικοί στόχοι της εποχής μας. Ταυτόχρονα όμως έχουν δημιουργηθεί αρκετές ανησυχίες για την ασφάλεια των ψηφιοποιημένων εγγράφων σχετικά με την ακεραιότητα και τη διαθεσιμότητα τους.

Η συγκεκριμένη εφαρμογή είχε ως στόχο να αποτελεί ένα εύχρηστο και ταυτόχρονα χρήσιμο εργαλείο για τους απλούς χρήστης/ιδιοκτήτες αυτοκινήτων, για τις εταιρείες που κατέχουν πληθώρα αυτοκινήτων αλλά και για τις επιχειρήσεις που πραγματοποιούν αλλαγές στα αυτοκίνητα αυτά. Ο χρήστης θα έχει τη δυνατότητα να λάβει στοιχεία σχετικά για το όχημα, να προσπελάσει το ιστορικό των αλλαγών που έχουν πραγματοποιηθεί αλλά και να προσθέσει νέες αλλαγές. Οι πληροφορίες είναι ασφαλές ως προς την ακεραιότητα καθώς μονάχα ο ιδιοκτήτης του οχήματος έχει το δικαίωμα να επεξεργαστεί τα στοιχεία ή να αφαιρέσει αλλαγές. Επιπλέον, όλα τα δεδομένα είναι αποθηκευμένα σε cloud database επομένως δεν υπάρχει κίνδυνος να χαθούν με την απώλεια ή καταστροφή της συσκευής στην οποία είναι η εφαρμογή.

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η Dart και έγινε χρήση της Flutter, η οποία είναι open-source UI software development kit που δημιουργήθηκε και υποστηρίζεται από την Google.

Επέλεξα την ανάπτυξη της εφαρμογής με την χρήση της Flutter καθώς επιθυμούσα να αποκτήσω περισσότερη εμπειρία και να αναπτύξω μια πλήρες λειτουργική εφαρμογή. Το πρώτο stable version release της Flutter έγινε πρόσφατα, το Δεκέμβριο του 2018, και έκτοτε αναφέρεται όλο και περισσότερο στο κομμάτι του application development και χρησιμοποιείται ακόμα και από γνωστές μεγάλες εταιρείες (όπως για παράδειγμα η BMW, το ebay κ.α.). Εν κατακλείδι, επέλεξα την ανάπτυξη της εφαρμογής με τη χρήση της Flutter προκειμένου να εξοικειωθώ με αυτή και αποκτήσω τις νέες δεξιότητες και γνώσεις.

Περίληψη

Η πτυχιακή εργασία είναι μια εφαρμογή android, η οποία κρατάει το ιστορικό των αλλαγών που πραγματοποιούνται σε αυτοκίνητα. Η γνησιότητα του χρήστη πετυχαίνεται με τη σύνδεση με τον αριθμό τηλεφώνου του χρήστη και την εισαγωγή του κωδικού που αποστάλθηκε στο συγκεκριμένο αριθμό με τη μορφή μηνύματος. Ο χρήστης μπορεί να διαχειριστεί όσα αυτοκίνητα επιθυμεί. Αρχικά, χρειάζεται να γίνει η εγγραφή του αυτοκινητου, η οποία γίνεται είτε εισάγωντας χειροκίνητα τις απαραίτητες πληροφορίες, είτε αποκωδικοποιώντας την πληροφορία από έναν κωδικό ταχείας απόκρισης (QR code) που δημιουργήθηκε από άλλον χρήστη μέσα από την εφαρμογή. Μπορεί να πραγματοποιηθεί αναζήτηση με βάση τον αριθμό πινακίδας ή το μοντέλο του αυτοκινήτου, σε περίπτωση που υπάρχουν πολλά οχήματα. Για κάθε αυτοκίνητο υπάρχουν 6 βασικές κατηγορίες αναβαθμίσεων που μπορούν να πραγματοποιηθούν (έλεγχος, αλλαγή λαδιών, αλλαγή ελαστικών, αλλαγή φρένων, αλλαγή υαλοκαθαριστήρων και αλλαγή μπαταρίας) αλλά υπάρχει και η δυνατότητα εισαγωγής σχολίου σε περίπτωση αναβάθμισης κάποιου εξαρτήματος που δεν ανήκει σε κάποια από τις παραπάνω κατηγορίες. Η βάση δεδομένων είναι στο διαδίκτυο και σε άμεση επικοινωνία. Με αυτόν τον τρόπο, σε περίπτωση κλοπής, βλάβης ή απώλειας της συσκευής, το ιστορικό των αλλαγών δε χάνεται. Σχετικά με το κωδικό ταχείας απόκρισης (QR Code), τα δεδομένα του κρυπτογραφούνται με κλειδί 32 bytes και αποκρυπτογραφούνται κατά τη σάρωση του. Η συσκευή η οποία σαρώνει τον κωδικό ταχείας απόκρισης, έχει πρόσβαση στο ιστορικό των αλλαγών, μπορεί να πραγματοποιήσει καινούργιες αλλαγές αλλά για λόγους ασφαλείας δε μπορεί να διαγράψει καμία αλλαγή που έχει ήδη πραγματοποιηθεί.

«Service Book»

«Pomportsis A. Georgios»

Abstract

The thesis project is an android application regarding cars and tracking changes that have been made on them. The authentication is accomplished by logging in the application with the phone number and after that by inserting the SMS code which has been sent to the phone. Every user can register as many cars as he wants. Cars can be added either manually by inserting the information which is needed, or by scanning the QR code of a car which is already registered on the app by a different device. In case that there are a lot of cars registered, search can be made through the car's plate of the cars that we desire. For every car the user can add changes that have been made, like changes of Tires, Oils, Wipers etc. and track their history. The application keeps track of changes of Services, Tires, Oils, Pads, Wipers, Battery and there are extra notes that can be made. The database of the application is online and real time so the history of changes won't be lost if the device is stolen, lost or damaged. Regarding the QR code, the message is encrypted with a 32 bytes key. The device which scans the QR code does not have access to delete anything, but it can access the whole history and add new updates that have been made on the car.

Ευχαριστίες

Πριν την παρουσίαση της παρούσας εργασίας, αισθάνομαι την υποχρέωση να ευχαριστήσω ορισμένους από τους ανθρώπους που γνώρισα, συνεργάστηκα μαζί τους και έπαιξαν πολύ σημαντικό ρόλο στην πραγματοποίησή της.

Η παρούσα εργασία αποτελεί τη Διπλωματική Εργασία στο πλαίσιο των σπουδών μου στο τμήμα μηχανικών πληροφορικής και ηλεκτρονικών συστημάτων του ΔΙΠΑΕ υπό την επίβλεψη του αναπληρωτή καθηγητή Κεραμόπουλου Ευκλείδη, στον οποίο οφείλω ιδιαίτερες ευχαριστίες τόσο για την ανάθεση της εργασίας όσο και για την εμπιστοσύνη που μου έδειξε και τη γενικότερη συμβολή του στη μέχρι τώρα σταδιοδρομία.

Τέλος, θα ήθελα να ευχαριστήσω θερμά για τις πολύτιμες συμβουλές και για την καθοδήγηση τους φίλους μου, τους συμφοιτητές μου αλλά και τον συνεργάτη μου και υπεύθυνο μου κατά την πρακτική μου άσκηση Γαβριήλ.

Περιεχόμενα

Πρόλογος ……..v

Περίληψη ……..……vi

Abstract……..……vii

Ευχαριστίες ……...viii

Περιεχόμενα ……..……..ix

Κατάλογος Σχημάτων ……xi

Κατάλογος Διαγραμμάτων ……..xi

Κατάλογος Πινάκων ……..……...xi

Συντομογραφίες……..xii

Κεφάλαιο 1ο: Εισαγωγή ……1

Κεφάλαιο 2ο: Τεχνολογίες που χρησιμοποιήθηκαν ……...3

2.1 Εισαγωγή……...3

2.2 Flutter ……....3

2.2.1 Εισαγωγή στην Flutter……3

2.2.2 Flutter - Cross-platform framework……4

2.2.3 Hot-Reload……...6

2.2.4 Widgets……..……...6

2.2.5 Routing ……....8

2.3 Dart Programming Language……....9

2.3.1 Εισαγωγή ……9

2.3.2 Compile……..……..9

2.3.3 JIT and AOT……..10

2.3.4 Garbage Collector……..11

2.4 Google Firebase Services……....12

2.4.1 Αυθεντικοποίηση χρήστη και Firebase Authentication……13

2.4.2 Cloud Firestore ……14

2.4.2.1 NoSQL Database……..15

2.4.2.2 Firestore ως Cloud-Hosted Database ……...16

2.4.2.3 Security of Firestore……16

2.4.3 Cloud Storage ……....17

2.4.3.1 Εισαγωγή……..17

2.4.3.2 Πλεονεκτήματα χρήσης Cloud Storage ……...17

2.4.3.3 Monitor Cloud Storage Activity……...18
2.4.4 Firebase Cloud Messaging - Push Notifications……....19
2.4.4.1 Εισαγωγή στα Push-Notifications……....19
2.4.4.2 Firebase Cloud Messaging (FCM) ……...19
2.4.4.3 Αρχιτεκτονική Firebase Cloud Messaging ……..20
2.5 QR code……...21
2.6 Adobe ……..23
2.7 Επίλογος……..24
Κεφάλαιο 3ο: Service Book App……..25
3.1 Εισαγωγή……25
3.2 Είσοδος χρήστη ……...25
3.2.1 SharedPreferences & stayLoggedIn ……..29
3.3 Κυρίως Μενού / Αρχική Σελίδα……..31
3.3.1 Αναζήτηση……32
3.3.2 Εισαγωγή οχήματος……...33
3.3.2.1 Εισαγωγή οχήματος χειροκίνητα ……35
3.3.2.2 Προσπέλαση οχήματος με QR code……37
3.3.3 Επεξεργασία οχήματος ……..40
3.3.4 Διαγραφή οχήματος……...43
3.4 Μενού χρήστη ……45
3.5 Διαχείριση οχήματος……...49
3.5.1 Εμφάνιση αναβαθμίσεων……...51
3.5.2 Διαγραφή αναβάθμισης ……54
3.5.3 Προσθήκη αναβάθμισης……....55
3.5.4 Δημιουργία QR code ……56
3.5.5 Πλήρες ιστορικό αλλαγών……58
3.6 Επίλογος……..60
Κεφάλαιο 4ο: Συμπεράσματα ή/και προτάσεις βελτίωσης ……...……...…63
ΒΙΒΛΙΟΓΡΑΦΙΑ ……65

Κατάλογος Σχημάτων

Εικόνα 2.1: Αρχιτεκτονική των widget στην Flutter……8
Εικόνα 2.2: Εναλλαγή σε route και επιστροφή ……...…….9
Εικόνα 2.3: Δημιουργία εφαρμογών με Dart……...10
Εικόνα 2.4: χρήση JIT & AOT……..……..11
Εικόνα 2.5: Young Space Scavenger……..……12
Εικόνα 2.6: Δεδομένα εφαρμογής αποθηκευμένα στην Firestore……..15
Εικόνα 2.7: Σύγκριση μεταξύ Firebase και RDBMS ……...……..16
Εικόνα 2.8: Μετρήσεις αιτημάτων στο Cloud Storage……...18
Εικόνα 2.9: Αρχιτεκτονική Firebase Cloud Messaging……..21
Εικόνα 2.10: Παραδοσιακός γραμμικός κώδικας……...22
Εικόνα 2.11: Κώδικας ταχείας απόκρισης……..22
Εικόνα 2.12: Εικονίδιο εφαρμογής……24
Εικόνα 2.13: Γραφικό στοιχείο εντός της εφαρμογής……...24
Εικόνα 3.1: Εισαγωγή τηλεφωνικού αριθμού ……...…….28
Εικόνα 3.2: Εισαγωγή μοναδικού κωδικού ……...…….28
Εικόνα 3.3: Κυρίως μενού της εφαρμογής……32
Εικόνα 3.4: Αναζήτηση οχήματος βάση αριθμού κυκλοφορίας ……...…….33
Εικόνα 3.5: Αναζήτηση οχήματος βάση μάρκας και μοντέλου ……...……..33
Εικόνα 3.6: Διεπαφή προσθήκης οχήματος……....36
Εικόνα 3.7: Σκανάρισμα λανθασμένου QR code ……...40
Εικόνα 3.8: Επεξεργασία στοιχείων οχήματος……...41
Εικόνα 3.9: Επιλογή εικόνας οχήματος……..……42
Εικόνα 3.10: Μενού χρήστη ……..……47
Εικόνα 3.11: Αναδυόμενο παράθυρο About ……...……...48
Εικόνα 3.12: Manual εφαρμογής……....49
Εικόνα 3.13: Διαχείριση οχήματος από τον ιδιοκτήτη……...51
Εικόνα 3.14: Διαχείριση οχήματος από τρίτο χρήστη……....51
Εικόνα 3.15: Ιστορικό αλλαγών συγκεκριμένης κατηγορίας (ελαστικών) ……...……52
Εικόνα 3.16: Προσθήκη αναβάθμισης……....56
Εικόνα 3.17: προεπισκόπηση PDF με το QR code……58
Εικόνα 3.18: Χρονοδιάγραμμα αλλαγών……....59

Κατάλογος Διαγραμμάτων

Διάγραμμα 2.1: Χρήση φορητών συσκευών και σταθερού υπολογιστή……..4
Διάγραμμα 2.2: Διαθέσιμες εφαρμογές στο Google Play Store & App Store……..5
Διάγραμμα 2.3: Πωλήσεις εφαρμογών στο Play Store ……5
Διάγραμμα 2.4: Πωλήσεις εφαρμογών στο App Store ……....5

Κατάλογος Πινάκων

Πίνακας 2.1: Επίπεδα διόρθωσης QR Code ……...……....23

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
QR code	Quick Response code
URL	Uniform Resource Locator
SMS	Short message service
PDF	Portable Document Format
GSM	Global System for Mobile communications
SDK	Software Development Kit
KB	Kilobyte. Μονάδα μνήμης που ισούται με 1024 bytes.
ARGB	Alpha, Red, Green, Blue - Σύστημα αναπαράστασης χρωμάτων με πολλαπλά επίπεδα διαφάνειας.
VM	Virtual Machine. Η εικονική συσκευή στην οποία εκτελείται ο κώδικας κατα το στάδιο της ανάπτυξης.
HTTP	Hypertext Transfer Protocol. Πρωτόκολλα εφαρμογής που επιτρέπει την επικοινωνία χρηστών στον παγκόσμιο ιστό.
XMPP	Extensible Messaging and Presence Protocol. Πρωτόκολλο επικοινωνίας σε πραγματικό χρόνο
APIs	Application Programming Interface. Λογισμικό που επιτρέπει σε δυο εφαρμογές να επικοινωνούν μέσω αυτού.
SIM card	Subscriber Identity Module card. Κάρτα για κινητά τηλέφωνα ή άλλες φορητές συσκευές, η οποία συνδέει την συσκευή με ένα συγκεκριμένο τηλεφωνικό δίκτυο
AES	Advanced Encryption Standard. Προδιαγραφή για την κρυπτογράφηση ηλεκτρονικών δεδομένων.

Κεφάλαιο 1ο: Εισαγωγή

Η εκπόνηση της παρούσας πτυχιακής εργασίας έχει ως στόχο το σχεδιασμό και τη δημιουργία εφαρμογής για φορητές συσκευές με λειτουργικό σύστημα Android αλλά και την ανάλυση των κλάσεων και των μεθόδων που χρησιμοποιήθηκαν στον κώδικα της. Με το παρόν κείμενο, προσφέρεται μια αναλυτική περιγραφή τόσο των λειτουργιών που υποστηρίζει η εφαρμογή όσο και του τρόπου με τον οποίον αυτές αναπτύχθηκαν και πήραν την τελική του μορφή. Αυτό έχει ως αποτέλεσμα, το κείμενο να χαρίζει στον αναγνώστη όχι μόνο ένα αναλυτικό εγχειρίδιο της λειτουργίας της εφαρμογής αλλά και μια πρώτη επαφή με καινούργιες τεχνολογίες και καινοτόμες ιδέες που θα μπορεί να χρησιμοποιήσει για την ανάπτυξη δικών του εφαρμογών.

Κύριος στόχος και αντικείμενο της εφαρμογής είναι η ψηφιοποίηση του βιβλιαρίου αναβαθμίσεων του αυτοκινήτου. Με αυτό το τρόπο, επιτυγχάνεται η διασφάλιση της διαθεσιμότητας των δεδομένων και η πρόσβαση σε αυτά από οποιαδήποτε φορητή συσκευή. Εν' ολίγοις, επιτυγχάνεται ο εκσυγχρονισμός της καταγραφής των δεδομένων ως προς τη διαδικασία συντήρησης των οχημάτων.

Στο 2ο κεφάλαιο, θα γίνει λεπτομερής περιγραφή των τεχνολογιών που χρησιμοποιήθηκαν για τη δημιουργία της παρούσας εφαρμογής. Εντός του συγκεκριμένου κεφαλαίου θα περιγραφούν όλα τα βασικά στοιχεία που χρειάζεται να γνωρίζει ένας προγραμματιστής προκειμένου να είναι σε θέση να κατανοήσει πλήρως το τρόπο λειτουργίας και τα χαρακτηριστικά της εφαρμογής.

Αρχικά, γίνεται αναφορά στη γλώσσα προγραμματισμού που επιλέχθηκε για την ανάπτυξη της εφαρμογής, στο κιτ ανάπτυξης διεπαφής που χρησιμοποιήθηκε αλλά και αναλυτική περιγραφή των πλεονεκτημάτων του καθενός και των λόγων για τους οποίους επιλέχθηκαν και χρησιμοποιήθηκαν. Στη συνέχεια, απαριθμούνται και περιγράφονται οι υπηρεσίες που χρησιμοποιήθηκαν κατα την ανάπτυξη της εφαρμογής. Οι υπηρεσίες που επιλέχθηκαν, παρέχονται και υποστηρίζονται από την Google και χρησιμοποιήθηκαν για τη εξυπηρέτηση διαφόρων λειτουργιών, όπως για παράδειγμα για την αυθεντικοποίηση του χρήστη, την αποθήκευση δεδομένων κ.α.. Επιπλέον, περιγράφονται οι τεχνολογίες που προορίζονται για τη διευκόλυνση του χρήστη και την αύξηση της αποδοτικότητας της εν λόγω εφαρμογής, όπως για παράδειγμα η δημιουργία και χρήση κωδικών ταχείας πρόσβασης (QR code), η κρυπτογράφηση δεδομένων κ.α.. Τέλος, γίνεται αναφορά στα προγράμματα τα οποία χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής στην τελική της μορφή, όπως για παράδειγμα στο πρόγραμμα επεξεργασίας και σχεδιασμού εικονιδίων που χρησιμοποιήθηκε για τη δημιουργία των γραφικών στοιχείων της εφαρμογής.

Στο 3ο κεφάλαιο αναλύεται ο τρόπος λειτουργίας της εφαρμογής και τα βασικά χαρακτηριστικά της. Περιγράφονται διεξοδικά οι λειτουργίες της και ο σκοπός που κάθε μια από αυτές επιτελεί, δίνοντας ταυτόχρονα εικόνες από τη διεπαφή του χρήστη, ώστε με αυτό τον τρόπο να επιτυγχάνεται και η ουσιαστικότερη κατανόηση της δομής και της λειτουργίας της. Επιπλέον γίνεται επεξήγηση του κώδικα που χρησιμοποιήθηκε για τη δημιουργία της εφαρμογής και παρατίθενται τα σημαντικότερα τμήματά του.

Τέλος, στο 4ο κεφάλαιο γίνεται ανακεφαλαίωση των σημαντικότερων λειτουργιών και ανασκόπηση των αρχικών στόχων και του κατά πόσο επιτεύχθηκαν. Επίσης, γίνονται προτάσεις βελτίωσης της εφαρμογής.

Κεφάλαιο 2ο: Τεχνολογίες που χρησιμοποιήθηκαν

2.1 Εισαγωγή

Σε αυτο το κεφάλαιο θα περιγραφούν οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής, τα κύρια χαρακτηριστικά τους αλλά και τα κύρια πλεονεκτήματα τους, τα οποία οδήγησαν στην επιλογή αυτών των τεχνολογιών. Στην πρώτη ενότητα, περιγράφεται το ανοιχτού κώδικα UI framework για τη δημιουργία εφαρμογων για iOS και Android η Flutter, η οποία υποστηρίζεται από την Google [1]. Έπειτα αναφέρεται η γλώσσα προγραμματισμού που χρησιμοποιεί η Flutter και αυτή είναι η Dart [1]. Με τη χρήση της Flutter και της Dart αναπτύχθηκε η συγκεκριμένη εφαρμογή. Στη συνέχεια περιγράφεται η πλατφόρμα Google Firebase [2] και οι υπηρεσίες της οποίας χρησιμοποιήθηκαν. Με την Firebase γίνεται χειρισμός του μεγαλύτερου μέρους της εργασίας από την πλευρά του διακομιστή όσον αφορά την ανάπτυξη εφαρμογών. Τέλος περιγράφεται το λογισμικό επεξεργασίας εικόνων, το οποίο χρησιμοποιήθηκε για το σχεδιασμό των γραφιστικών της εφαρμογής. Το πρόγραμμα αυτό είναι το Adobe Photoshop.

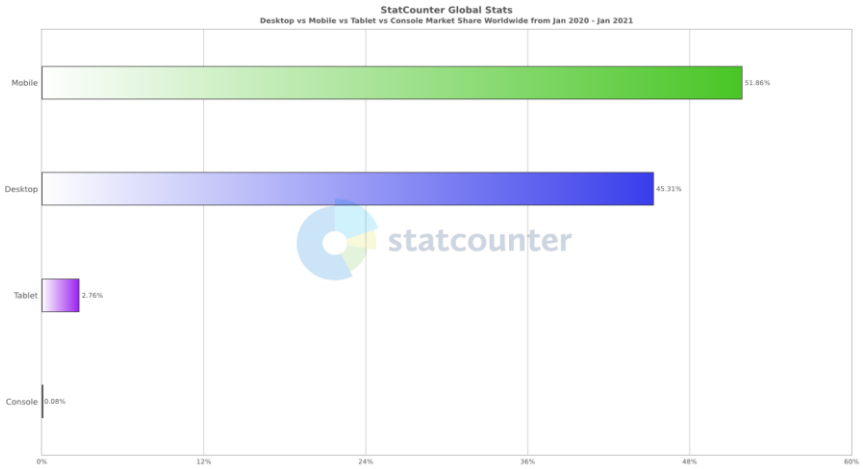
2.2 Flutter

2.2.1 Εισαγωγή στην Flutter

Η Flutter είναι open source cross-platform framework, το οποίο αναπτύχθηκε και υποστηρίζεται από την Google. Αρχικά ξεκίνησε με την ονομασία “Sky” το 2016 και λειτουργούσε μονάχα για εφαρμογές στο λειτουργικό σύστημα Android. Η πρώτη σταθερή έκδοση της Flutter έγινε διαθέσιμη το Δεκέμβριο του 2018. Ο αρχικός στόχος της Google ήταν η δυνατότητα γρήγορου σχεδιασμού εφαρμογής η οποία θα μπορεί να εκτελεστεί σε όλες τις συσκευές ανεξαρτήτως λειτουργικού συστήματος και θα είχε συνεχόμενη απόδοση σταθερών 60fps προκειμένου να ναι φιλική προς τον χρήστη.

Παρόλο που η flutter είναι νέα τεχνολογία (έχουν περάσει σχεδόν δυο χρόνια από το πρώτο stable release), έχει πολλά πλεονεκτήματα και διαφοροποιήσεις σε σχέση με τις υπόλοιπες επιλογές που είναι διαθέσιμες για ανάπτυξη εφαρμογών για κινητά. Για αυτόν τον λόγο η χρήση της αυξάνεται ολοένα και περισσότερο και χρησιμοποιείται ακόμα και από επιχειρήσεις με τεράστιο κοινό όπως για παράδειγμα η BMW, το Ebay, The New York Times κ.α. [3].

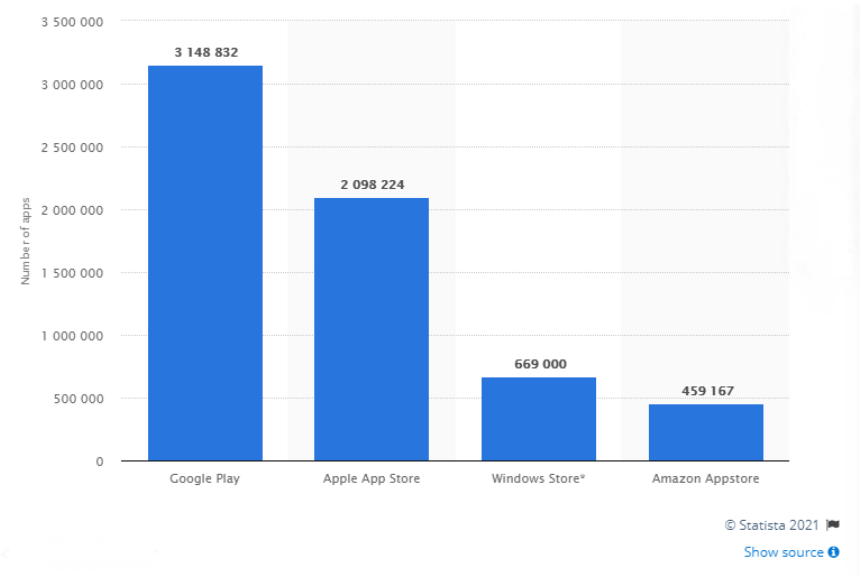
Παρόλο την εκτεταμένη χρήση της για εφαρμογές κινητών (android, iOS, Fuschia), μπορεί να χρησιμοποιηθεί και για ανάπτυξη desktop εφαρμογών σε Windows, macOS και Linux λειτουργικό σύστημα. Επιπλέον, μπορεί να χρησιμοποιηθεί για την ανάπτυξη web εφαρμογών. Οι εφαρμογές κινητών τηλεφώνων χρησιμοποιούνται ολοένα και περισσότερο και πλέον αποτελούν σημαντικό μέρος της καθημερινότητας μας. Παρ’όλα αυτά η χρήση εφαρμογών σε σταθερό υπολογιστή συνεχίζει να έχει πολλούς χρήστες. Βάση στατιστικής έρευνας, η οποία πραγματοποιήθηκε σε παγκόσμιο επίπεδο κατα τη διάρκεια του έτους 2020, υπήρξε μεγαλύτερη κίνηση στο διαδίκτυο μέσω τον κινητών τηλεφώνων (54.18%) σε σχέση με τους ηλεκτρονικούς υπολογιστές (42.9%) (Διάγραμμα 2.1) [4]. Επομένως, είναι σημαντικό να υπάρχει η δυνατότητα ανάπτυξης εφαρμογών που μπορούν να εκτελεστούν σε όλες τις συσκευές είτε είναι φορητές συσκευές είτε σταθεροί υπολογιστές.



Διάγραμμα 2.1: Χρήση φορητών συσκευών και σταθερού υπολογιστή

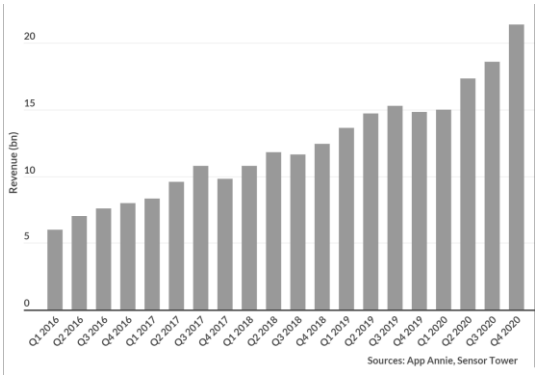
2.2.2 Flutter - Cross-platform framework

Τη σημερινή εποχή, τα κινητά τηλέφωνα χρησιμοποιούν κυρίως δύο διαφορετικά λειτουργικά συστήματα, το Android [5] και το iOS [6], τα οποία διατίθενται από την Google και την Apple αντίστοιχα. Δεδομένου ότι υπάρχουν πληθώρα διαφορών μεταξύ των δύο αυτών λειτουργικών συστημάτων, για να αναπτυχθούν εφαρμογές απαιτούνται διαφορετικές γνώσεις προγραμματισμού, όπως για παράδειγμα Java/Kotlin για το λειτουργικό σύστημα Android και Object-C/Swift για το iOS [7]. Αυτό το δίλημμα αποτελεί μία από τις μεγαλύτερες δυσκολίες των προγραμματιστών καθώς το λειτουργικό σύστημα Android χρησιμοποιείται περισσότερο σε σχέση με το iOS αλλά οι χρήστες του iOS είναι διατεθειμένοι να πληρώσουν για τις εφαρμογές τους. Στο παρακάτω διάγραμμα (Διάγραμμα 2.2) παρατηρούμε ότι οι διαθέσιμες εφαρμογές για Android είναι περίπου 30% περισσότερες σε σχέση με αυτές για iOS. Πιο συγκεκριμένα το τελευταίο τετράμηνο του 2020 υπήρχαν 3.148.832 διαθέσιμες εφαρμογές στο Google Play Store για τα κινητά με λειτουργικό σύστημα Android. Ενώ, την ίδια περίοδο υπήρχαν 2.098.224 διαθέσιμες εφαρμογές στο App Store για κινητά με λειτουργικό σύστημα iOS [8].

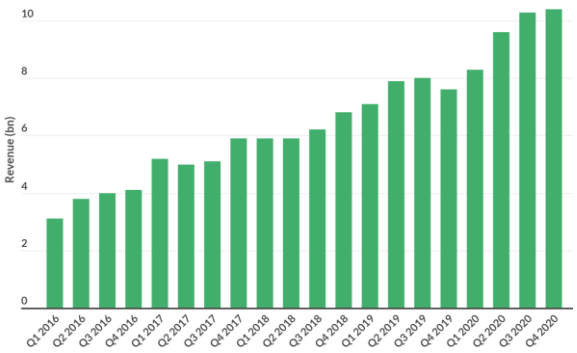


Διάγραμμα 2.2: Διαθέσιμες εφαρμογές στο Google Play Store & App Store

Απο την άλλη, οι χρήστες που διαθέτουν λειτουργικό σύστημα iOS είναι πιο δεκτικοί στο να πληρώσουν ώστε να αποκτήσουν μια εφαρμογή. Βάση στατιστικών, σε παγκόσμιο επίπεδο, οι πωλήσεις που γίνονται από το App Store της Apple (Διάγραμμα 2.4) είναι σχεδόν διπλάσιες σε σχέση με αυτές που γίνονται στο Google Play Store (Διάγραμμα 2.3) [9]. Η έρευνα αφορούσε τα τετράμηνα απο το 2016 έως το 2020. Ενδεικτικά παρατηρούμε ότι οι πωλήσεις στο τελευταίο τετράμηνο του 2020 στο App Store ηταν 21,4 δισεκατομμύρια δολάρια ενώ στο Play Store 10,4 δισεκατομμύρια δολάρια.



Διάγραμμα 2.3: Πωλήσεις στο Play Store



Διάγραμμα 2.4: Πωλήσεις στο App Store

Με τη χρήση της Flutter μπορεί να λυθεί αυτό το δίλημμα καθώς πρόκειται για cross-platform framework. Αυτό σημαίνει ότι ο κώδικας της μπορεί να εκτελεστεί σε συσκευές με λειτουργικό σύστημα είτε Android είτε iOS αλλά ακόμη και στο Fuschia, το λειτουργικό σύστημα της Google. Επιπλέον, με τη χρήση της Flutter μπορούν να δημιουργηθούν Desktop αλλά και Web εφαρμογές. Αυτό οφείλεται στο οτι δεν χρησιμοποιεί native platform components άρα το περιεχόμενο της εφαρμογής σχεδιάζεται απο την Flutter και οχι απο το λειτουργικό σύστημα. Επίσης σημαντικό είναι οτι τα γραφικά στοιχεία της διεπαφής προστίθενται γρήγορα και εύκολα ακόμα και αν δεν τα υποστηρίζει το λειτουργικό σύστημα.

2.2.3 Hot-Reload

Ένα ακόμα πλεονέκτημα που προσφέρει η flutter στον προγραμματιστή, είναι το λεγόμενο stateful Hot-Reload κατα τη διάρκεια της ανάπτυξης του κώδικα [1], [7], [10], [11]. Το Hot-Reload λειτουργεί μονάχα σε debug mode. Κατα τη διάρκεια του debug mode, η εφαρμογή εκτελείται είτε σε εικονική συσκευή είτε σε πραγματική συσκευή στην οποία έχουμε δώσει τα κατάλληλα δικαιώματα.

Όταν αποθηκεύονται τα τροποποιημένα αρχεία, η συσκευή ενημερώνει μονάχα τις κλάσεις που έχουν αλλαχτεί. Στη συνέχεια, η flutter ξεκινάει να σχεδιάζει τα widget και οι αλλαγές αντικατροπτίζονται αυτόματα εντός ολίγον δευτερολέπτων. Αυτό έχει σαν αποτέλεσμα, ο χρήστης να μπορεί να πραγματοποιήσει αλλαγές και να της εφαρμόσει στην εικονική μηχανή γρήγορα και εύκολα. Με την χρήση του Hot-Reload, εισάγεται και εκτελείται ο καινούργιος-τροποποιημένος πηγαίος κώδικας στην τρέχουσα εικονική μηχανή που τρέχει χωρίς όμως να αλλάξει η εσωτερική δομή της εφαρμογής. Εν συντομία, αντί να εκτελεστεί όλος ο κώδικας από την αρχή, εκτελούνται μονάχα τα κομμάτια του κώδικα τα οποία είναι διαφορετικά σε σχέση με την εκτέλεση που ήδη τρέχει. Ταυτόχρονα όμως διατηρείται και η τρέχουσα κατάσταση της εφαρμογής. Άρα δεν απαιτείται να γίνει πλήρης επανεκκίνηση της εφαρμογής. Για παράδειγμα, εάν πατήσουμε ένα κουμπί ορισμένες φορές και έπειτα πραγματοποιήσουμε αλλαγές στον τίτλο και κάνουμε Hot-Reload, τότε ο αριθμός των φορών που πατήθηκε το κουμπί θα διατηρηθεί ακόμα και μετά την επαναφόρτιση της εφαρμογής. Στην περίπτωση που δεν επιθυμούμε να διατηρηθεί η κατάσταση της εφαρμογής, μπορούμε να πραγματοποιήσουμε Hot-Restart αντί για Hot-Reload. Σε αντίθεση με το Hot-Reload το οποίο πραγματοποιείται κάθε φορά που αποθηκεύεται αρχείο με αλλαγές, το Hot-Restart χρειάζεται να το επιλέξουμε είτε από το μενού είτε με τη συντόμευση απο το πληκτρολόγιο με βάση το πρόγραμμα που χρησιμοποιείται. Η δυνατότητα του Hot-Restart απαιτεί σχεδόν τον ίδιο χρόνο με το Hot-Reload.

Εν κατακλείδι, το Hot-Reload βοηθάει δραματικά στην ανάπτυξη της εφαρμογής καθώς καθιστά την όλη διαδικασία ταχύτερη και απλούστερη. Ως μειονέκτημα του Hot-Reload μπορεί να θεωρεί ότι η εφαρμογή θα ναι πιο αργή καθώς θα βρίσκεται σε debug mode.

2.2.4 Widgets

Τα γραφικά στοιχεία της διεπαφής, όπως για παράδειγμα ένα κουμπί, στην Flutter αναφέρονται ως widget. Εν συντομία, τα widget είναι τα βασικά δομικά στοιχεία της διεπαφής της εφαρμογής και αποτελούν ένα από τα πιο σημαντικά στοιχεία της. Χρειάζεται να είναι ελκυστικά στο χρήστη αλλά ταυτόχρονα να δηλώνουν ξεκάθαρα τη χρήση τους χωρίς να αφήνουν κενά για παρερμηνείες απο το χρήστη. Επιπλέον, τα widget πέραν του ότι είναι υπεύθυνα για την προβολή των γραφικών στοιχείων στην εφαρμογή και την εμφάνιση της, είναι επίσης υπεύθυνα και για τον τρόπο με τον οποίο θα ανταποκριθούν στις διάφορες ενέργειες του χρήστη (όπως για παράδειγμα στο παρατεταμένο πάτημα). Τα widget σχεδιάζουν τη διεπαφή του χρήστη φτιάχνοντας δομή τύπου tree, η οποία ονομάζεται widget tree. Επίσης, τα widget πρέπει να είναι όσο το δυνατόν γρηγορότερα προκειμένου να παρέχεται συνεχόμενη απόδοση σταθερών 60fps στη διεπαφή. Η δημιουργία του widget tree, η αρχικοποίηση των εμφωλευμένων widget, η παρουσίαση τους, η απόδοση των κινούμενων γραφικών και ότι άλλο προβάλλεται στη διεπαφή χρειάζονται ένα χρονικό διάστημα το οποίο πρέπει να είναι όσο πιο σύντομο γίνεται.

Όποτε υπάρχει αλλαγή στην κατάσταση ενός widget, η flutter προσαρμόζει τη διεπαφή του χρήστη εφαρμόζοντας αλλαγές στο widget tree μονάχα για τα widget στα οποία υπάρχουν αλλαγές. Με αυτόν τον τρόπο, ο επανασχεδιασμός της διεπαφής του χρήστη επιτυγχάνεται αποτελεσματικά.

Σαν Widget μπορεί να οριστούν:

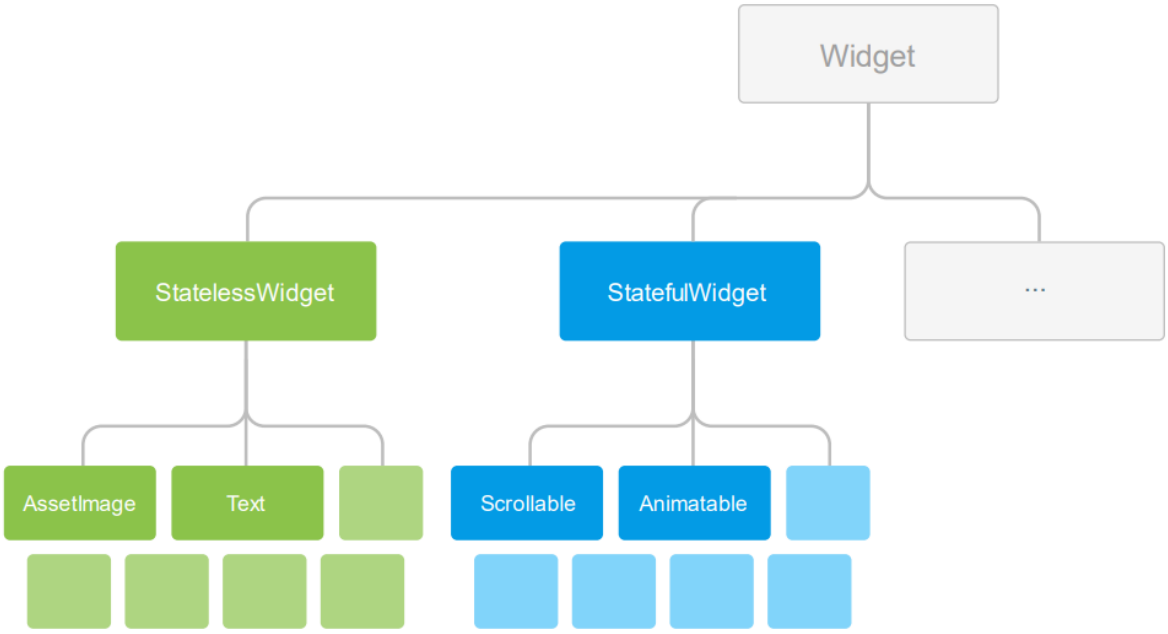
- Τα δομικά στοιχεία, όπως για παράδειγμα ένα κουμπί ή το μενού.
- Τα στοιχεία που προσδιορίζουν το στυλ της εφαρμογής, όπως για παράδειγμα ο ορισμός χρωμάτων ή γραμματοσειρών
- Στοιχεία τα οποία είναι υπεύθυνα για τη διάταξη της εφαρμογής, όπως για παράδειγμα το κενό που υπάρχει μεταξύ των widget ή από τις πλευρές της συσκευής.
- Οποιοδήποτε widget με εμφωλευμένα widget σε αυτό θεωρείται σαν ένα widget.

Τα widget χωρίζονται σε δύο κατηγορίες ως προς την κατάστασή τους κατα τη διάρκεια της εφαρμογής: τα Stateful Widget και τα Stateless Widget. Ως State ορίζονται οι πληροφορίες του widget οι οποίες μπορούν να αλλαχτούν κατα τη διάρκεια ζωής του, όπως για παράδειγμα η τιμή της μεταβλητής που εμφανίζεται [12]. Τα Stateless Widget είναι widget των οποίο το state δεν αλλάζει κατα τη διάρκεια της εφαρμογής. Σε αυτήν την κατηγορία ανήκουν widget όπως το Text, Icon κ.α.. Αντιθέτως, τα Stateful Widget είναι δυναμικά widget που σημαίνει ότι μπορεί να αλλαχτεί το state τους κατα τη διάρκεια της εφαρμογής. Σε αυτήν την κατηγορία περιλαμβάνονται widget όπως το Checkbox, Radio Button, Form, το TextField κ.α..

Η Flutter παρέχει μια ευρεία ποικιλία widget τα οποία είναι έτοιμα προς χρήση (ready-to-use). Τα widget έχουν τη δυνατότητα να προσαρμοστούν με βάση τις απαιτήσεις του χρήστη και παρέχονται δύο κατηγορίες widget προκειμένου να μιμηθούν τέλεια την εκάστοτε γλώσσα σχεδιασμού. Η πρώτη κατηγορία είναι τα Material Widget τα οποία χρησιμοποιούνται στις εφαρμογές Android. Ενώ στη δεύτερη κατηγορία βρίσκονται τα Cupertino Widgets τα οποία χρησιμοποιούνται για τις εφαρμογές με λειτουργικό σύστημα iOS της Apple. Τα widget που παρέχονται μπορούν να προσαρμοστούν πλήρως από τον προγραμματιστή και είναι και επεκτάσιμα. Έτσι, οι προγραμματιστές μπορούν να προσθέσουν καινούργια widget που επιθυμούν ή και να προσαρμόσουν τα ήδη υπάρχοντα που διατίθενται από την Flutter ώστε να ταιριάζουν με το στυλ που ακολουθεί η εταιρεία ή η προσωπική εφαρμογή.

Το βασικό πλεονέκτημα των widget της Flutter είναι ότι δεν χρησιμοποιεί τα γραφικά στοιχεία διεπαφής του συστήματος αλλά παρέχει τα δικά της. Για αυτόν τον λόγο, η διεπαφή μπορεί να σχεδιαστεί σε οποιαδήποτε συσκευή ανεξαρτήτως τα γραφικά στοιχεία που παρέχει.

Στην Flutter σχεδόν τα πάντα είναι widget, επομένως στο τέλος ο κώδικας καταλήγει να είναι μια ιεραρχία των εμφωλευμένων widget. Υπάρχουν widget, όπως για παράδειγμα το container, που μπορούν να έχουν πολλαπλά widget εμφωλευμένα μέσα τους. Επίσης, υπάρχουν widget τα οποία επιτρέπεται να έχουν μονάχα ένα widget εμφωλευμένο αλλά στη συνέχεια αυτό μπορεί να έχει είτε ένα είτε περισσότερα. Η αρχιτεκτονική που ακολουθεί η flutter για τα widget της παρουσιάζεται παρακάτω (Εικόνα 2.1).



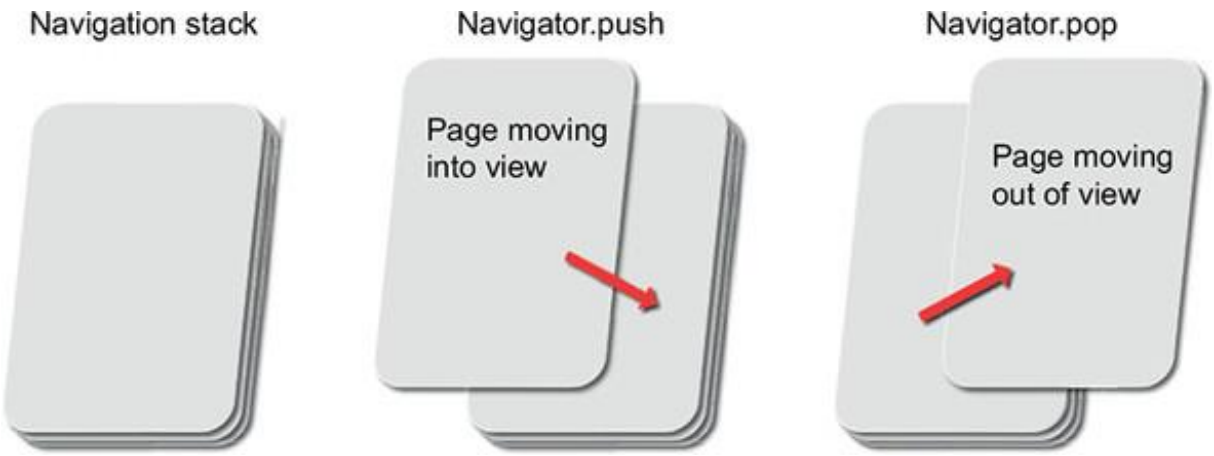
Εικόνα 2.1: Αρχιτεκτονική των widget στην Flutter

2.2.5 Routing

Η εναλλαγή των παραθύρων της εφαρμογής αποτελεί μια από τις σημαντικότερες προδιαγραφές με βάση τις οποίες κρίνεται η δομή της εφαρμογής. Αναπτύσσοντας διαφορετικά παράθυρα, οι προγραμματιστές μπορούν να απλοποιήσουν τη δομή της εφαρμογής αφου έτσι τη χωρίζουν σε υποκατηγορίες και αποσαφηνοποιούν τη λογική της πλοήγησης. Οι περισσότερες εφαρμογές περιέχουν πολλά διαφορετικά παράθυρα προκειμένου να εμφανίζουν διαφορετικά στοιχεία ή και να εκτελούν διαφορετικές ενέργειες. Για παράδειγμα, μια εφαρμογή μπορεί να αποτελείται από την οθόνη στην οποία συνδέεται ο χρήσης και έπειτα να γίνεται μετάβαση σε άλλη οθόνη.

Στις εφαρμογές Android η μετάβαση από ένα παράθυρο σε άλλο ισοδυναμεί με ενα Activity, ενώ στις εφαρμογές για iOS με ένα ViewController. Στην Flutter υπάρχει το widget Navigator και επιτρέπει την περιήγηση απο την μια οθόνη στην άλλη. Τα widget τα οποία καταλαμβάνουν πλήρως την οθόνη (full-screen) και μπορεί να γίνει περιήγηση σε αυτά αναφέρονται ως “route” και όχι ως παράθυρο ή οθόνη. Επομένως ο Navigator είναι το widget που είναι υπεύθυνο για την εναλλαγή μεταξύ routes.

Ο Navigator λειτουργεί με την χρήση της στοίβας άρα όταν γίνεται μετάβαση σε ένα route, το route προστίθεται στη στοίβα και έπειτα μπορεί να γίνει επιστροφή στην αρχικη σελίδα διώχνοντας το τελευταίο route απο τη στοίβα. Η εναλλαγή σε ένα route αναφέρεται ως “push” ενώ η επιστροφή στην αρχική ως “pop” (Εικόνα 2.2).



Εικόνα 2.2: Εναλλαγή σε route και επιστροφή

2.3 Dart Programming Language

2.3.1 Εισαγωγή

Κατα τη δημιουργία της Flutter βασικός στόχος ήταν να χρησιμοποιηθεί μια γλώσσα προγραμματισμού η οποία θα είναι οικεία στους έμπειρους προγραμματιστές και εύκολη να τη μάθουν οι καινούργιοι προγραμματιστές. Έτσι, επιλέχτηκε η γλώσσα προγραμματισμού Dart η οποία επίσης αναπτύχθηκε και υποστηρίζεται από την Google [1], [7], [10], [11]. Αρκετές δηλώσεις και φράσεις της σύνταξης της είναι όμοιες κυρίως με αυτές της C αλλά έχει και ορισμένες κοινές φράσεις με την Java, JavaScript και άλλες διαδεδομένες γλώσσες (όπως για παράδειγμα οι λέξεις κλειδιά “async” και “await”). Η ανάπτυξη της ξεκίνησε το 2011 και η πρώτη επίσημη σταθερή έκδοση έγινε διαθέσιμη το Νοέμβριο του 2013. Χρησιμοποιείται για την ανάπτυξη πληθώρων εφαρμογών όπως για παράδειγμα εφαρμογές IoT, web εφαρμογές, command-line script κ.α. Παρ’όλα αυτά, η χρήση της αυξήθηκε τα τελευταία χρόνια που χρησιμοποιείται ως κύρια γλώσσα προγραμματισμού απο την Flutter.

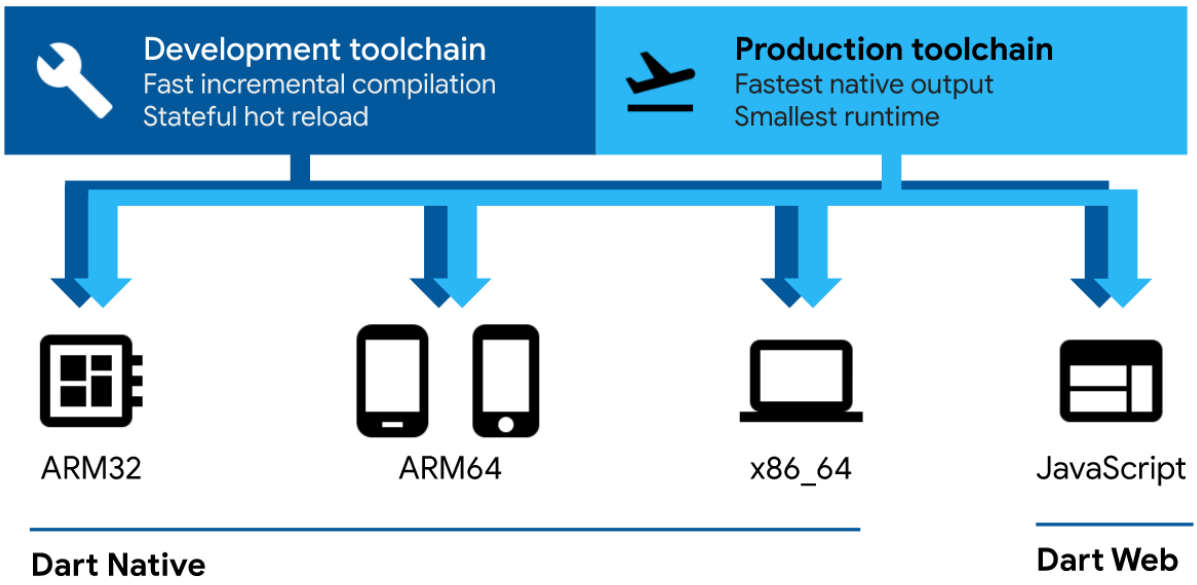
Πέρα από το γεγονός ότι πρόκειται για μια οικεία και εύκολη γλώσσα προγραμματισμού, υπήρχαν και άλλοι λόγοι που οδήγησαν στην επιλογή της Dart ως κύρια γλώσσα προγραμματισμού στην Flutter. Αρχικά, η Dart είναι αντικειμενοστραφής γλώσσα προγραμματισμού με σύνταξη όμοια της C. Στην Dart όλα είναι αντικείμενα, επομένως όλα τα δεδομένα χειρίζονται ομοιόμορφα. Επίσης, είναι class-based γλώσσα προγραμματισμού επομένως χρησιμοποιούνται abstract classes, interfaces και reified generics. Κάθε αντικείμενο αποτελεί instance της παρούσας κλάσης στην οποία ορίζεται και κάθε interface περιγράφει τις μεθόδους που είναι διαθέσιμες.

2.3.2 Compile

Η Dart μπορεί να μεταγλωτίζει τον κώδικα σε ARM και x86 επομένως οι εφαρμογές που είναι γραμμένες σε Dart μπορούν να εκτελεστούν και σε Android και σε iOS αλλά και σε άλλα λειτουργικά συστήματα όπως για παράδειγμα στο Fuschia, το λειτουργικό σύστημα της Google [13]. Για τη

δημιουργία των mobile αλλα και των desktop εφαρμογών χρησιμοποιείται και η εικονική μηχανή με Justi-In-Time compilation αλλά και Ahead-Of-Time compiler που θα τα δούμε αναλυτικά στη συνέχεια. Επίσης, μπορούν να δημιουργηθούν και web εφαρμογές καθώς μεταγλωτίζεται και σε JavaScript. Αυτό πετυχαίνεται με τη χρήση του dartdevc για την ανάπτυξη και με το dart2js για την παραγωγή. Στην παρακάτω εικόνα (Εικόνα 2.3) παρατηρούμε τις συσκευές για τις οποίες μπορεί να εκτελεστεί ο κώδικας Dart.

Εικόνα 2.3: Δημιουργία εφαρμογών με Dart



2.3.3 JIT and AOT

Καθώς χρησιμοποιείται η Dart για ανάπτυξη εφαρμογών κινητών τηλεφώνων ή desktop εφαρμογών, η Dart προσφέρει και το Dart VM με Just In Time (JIT) compiler του κώδικα αλλά και compiler που είναι Ahead Of Time (AOH) για την εκτέλεση κώδικα μηχανής [13]. Κατα τη διάρκεια του Native development η Dart χρησιμοποιεί το JIT και Virtual Machine προκειμένου να εκτελέσει τον κώδικα σε ARM και x64 κώδικα μηχανής. Οι μεταγλωττιστές JIT είναι εξαιρετικά γρήγοροι καθώς μεταγλωτίζουν τον κώδικα κατα τη διάρκεια της εκτέλεσης. Αυτό έχει σαν αποτέλεσμα, να παρέχονται περισσότερες λειτουργίες για ανάπτυξη στον χρήστη οι οποίες είναι πιο φιλικές. Ένα παράδειγμα που είδαμε νωρίτερα είναι το Hot-Reload που παρέχεται από την Flutter, το οποίο επιτυγχάνεται μέσω του JIT compiler. Το Hot-Reload εμφανίζει σχεδόν άμεσα τις αλλαγές που πραγματοποιήθηκαν στην εικονική μηχανή που εκτελείται ο κώδικας, διατηρώντας ταυτόχρονα την κατάσταση της εφαρμογής. Αυτή η φιλική προς τον χρήστη λειτουργία βοηθάει τον προγραμματιστή να παρατηρεί άμεσα τις αλλαγές που πραγματοποιεί και βελτιώνει σημαντικά τον κύκλο ανάπτυξης της εφαρμογής αλλά και το χρόνο που απαιτείται. Παρολο αυτά, το JIT δεν είναι ιδανικό για ανάπτυξη στην παραγωγή λόγω της αργής και τμηματικής εκτέλεσης. Για αυτό, όταν η εφαρμογή είναι έτοιμη για την παραγωγή χρησιμοποιείται ο μεταγλωττιστής AOT ο οποίος επιτρέπει τη σύνταξη σε native code ARM αλλά και x86. Ο κώδικας μεταγλωτίζεται εκ των προτέρων και έτσι αυξάνεται σημαντικά ο χρόνος ανάπτυξης. Εν κατακλείδι, ο JIT compiler χρησιμοποιείται κατα την ανάπτυξη της εφαρμογής σε debug mode και ο AOT compiler όταν επιθυμούμε να μεταγλωτήσουμε την εφαρμογή ώστε να γίνει διαθέσιμη στην

παραγωγή (Εικόνα 2.3). Η χρήση και των δύο τύπων μεταγλώττισης έχει ως αποτέλεσμα τις καλύτερες επιδόσεις και την εξοικονόμηση χρόνου τόσο σε επίπεδο παραγωγής όσο και στους κύκλους ανάπτυξης.

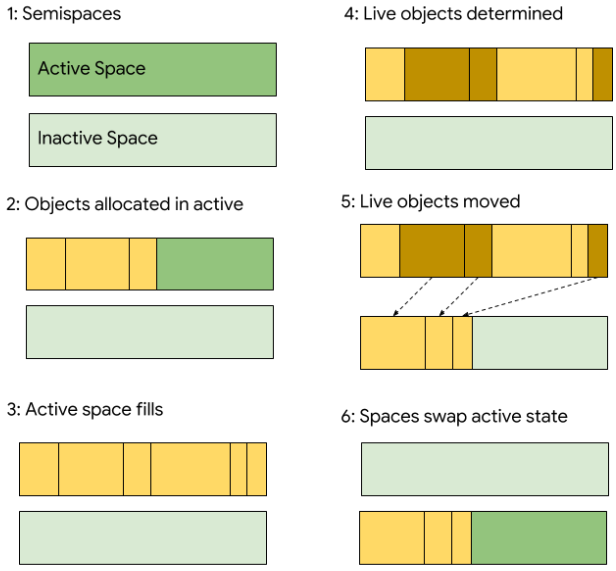
Εικόνα 2.4: χρήση JIT και AOT

2.3.4 Garbage Collector

Ακόμα ένα πλεονέκτημα της Dart είναι ότι πρόκειται για "garbage collector" γλώσσα προγραμματισμού. Έτσι, δεν υπάρχει λόγος ανησυχίας για την κατανομή της μνήμης καθώς η μνήμη η οποία δεν χρησιμοποιείται πλεον απο αντικείμενα θα απελευθερώνεται. Ο garbage collector είναι ένα βασικό συστατικό για τον χειρισμό της μνήμης καθώς δημιουργούνται πάρα πολλά αντικείμενα που παύουν να χρησιμοποιούνται αλλά καταλαμβάνουν χώρο στη μνήμη. Σε μία εφαρμογή με μέτριο επίπεδο πολυπλοκότητας της διεπαφής του χρήστη, χρησιμοποιούνται χιλιάδες stateless Widgets. Τα περισσότερα απο αυτά δημιουργούνται καθώς σχεδιάζεται η διεπαφή του χρήστη και έπειτα καταστρέφονται ή ανακατασκευάζονται εφόσον αλλάζει η κατάσταση της εφαρμογής ή πάψουν να είναι ορατά. Επομένως τα περισσότερα widget έχουν μικρή διάρκεια ζωής.

Προκειμένου να μην επηρεάζει την απόδοση της εφαρμογής, ο garbage collector εκτελεί τις ενέργειες του όταν η εφαρμογή είναι αδρανής ή όταν δεν υπάρχει αλληλεπίδραση μεταξύ του χρήστη και της διεπαφής. Ο garbage collector αποτελείται από δύο φάσεις: τον Young Space Scavenger και τον Parallel Mark Sweep Collectors [11].

Κατα την φάση Young Space Scavenger καταστρέφονται αντικείμενα με σύντομη διάρκεια ζωής, όπως για παράδειγμα τα Stateless Widget. Η μνήμη στην οποία κατανέμονται τα αντικείμενα αποτελείται από δύο τμήματα: το ενεργό και το ανενεργό. Σε οποιαδήποτε στιγμή χρησιμοποιείται μονάχα το ένα τμήμα. Τα νέα αντικείμενα τοποθετούνται στο ενεργό τμήμα της μνήμης έως ότου να γεμίσει. Όταν γεμίσει το ενεργό τμήμα, τότε τα αντικείμενα που εξακολουθούν να χρησιμοποιούνται μεταφέρονται στο ανενεργό τμήμα. Τα αντικείμενα για τα οποία δεν υπάρχουν αναφορές, άρα έχουν πάψει να χρησιμοποιούνται, παραμένουν στο ίδιο τμήμα μνήμης έως ότου να γραφτούν νέα αντικείμενα πάνω τους κατα τους επόμενους κύκλους της διαδικασίας. Έπειτα το ανενεργό τμήμα γίνεται το ενεργό και συνεχίζεται η διαδικασία όπου προστίθενται τα νέα αντικείμενα σε αυτό. Όμοια το τμήμα που ήταν ενεργό και γέμισε τώρα περιέχει μονάχα τα αντικείμενα που δεν χρησιμοποιούνται στα οποία θα γίνει overwrite και μετατράπηκε σε ανενεργό. Στην παρακάτω εικόνα (Εικόνα 2.5) παρατηρούμε την φάση young space scavenger βήμα προς βήμα.



Εικόνα 2.5: Young Space Scavenger

Η δεύτερη φάση του garbage collector είναι η Mark-Sweep και είναι υπεύθυνη για τα αντικείμενα με μεγάλη διάρκεια ζωής τα οποία όμως είναι πιο σπάνια σε μια εφαρμογή Flutter. Αυτό οφείλεται κυρίως στον τρόπο λειτουργίας των widget. Επομένως αυτή η φάση είναι πιο σπάνια σε σχέση με την young space scavenger. Κατα τη διάρκεια της mark-sweep φάσης χρειάζεται να σταματήσει προσωρινά η εκτέλεση της εφαρμογής. Παρόλο αυτά, η Flutter μπορεί να προγραμματίσει αυτήν την φάση σε περιόδους όπου ο χρήστης δεν αλληλεπιδρά με την εφαρμογή προκειμένου να μη γίνει αισθητή η παύση της εφαρμογής. Η φάση αυτή, αποτελείται επίσης από τρια στάδια. Αρχικά επισημαίνονται όλα τα αντικείμενα που χρησιμοποιούνται με flags. Έπειτα σαρώνεται όλη η μνήμη και διαγράφονται όσα αντικείμενα βρεθούν τα οποία δεν έχουν επισημανθεί. Τέλος, διαγράφονται όλα τα flag προκειμένου να επανέλθει στην αρχική κατάσταση και να μπορεί να ξανα εκτελεστεί η mark-sweep φάση του garbage collector.

2.4 Google Firebase Services

Η Firebase είναι πλατφόρμα εφαρμογών ιστού από τους Andrew Lee και James Tamplin το 2011 με κύριο στόχο την παροχή βάσης δεδομένων σε πραγματικό χρόνο [14]. Ωστόσο, η Google εξαγόρασε την Firebase το 2014 και πρόσθεσε πληθώρα λειτουργιών και υποστηρικτικών εργαλείων. Πλέον αποτελεί ένα εργαλείο υποστήριξης για την ανάπτυξη εφαρμογών υψηλής απόδοσης και εύκολα επεκτάσιμες με χαμηλό κόστος και χρόνο. Χρησιμοποιείται σε εκατομμύρια εφαρμογές αλλά κυρίως σε web και mobile εφαρμογές στις οποίες απαιτείται ανταλλαγή δεδομένων σε πραγματικό χρόνο. Παρακάτω γίνεται αναφορά των εργαλείων της Firebase, τα οποία χρησιμοποιήθηκαν στη συγκεκριμένη εφαρμογή.

2.4.1 Αυθεντικοποίηση χρήστη και Firebase Authentication

Η αυθεντικοποίηση του χρήστη είναι η διαδικασία της θετικής επαλήθευσης της ταυτότητας ενός χρήστη, μιας συσκευής ή άλλης οντότητας σε ένα υπολογιστικό σύστημα προκειμένου να αποκτήσει πρόσβαση στους πόρους του συστήματος [15]. Εν' συντομία, η αυθεντικοποίηση είναι η διαβεβαίωση ότι η οντότητα επικοινωνίας είναι αυτη που ισχυρίζεται ότι είναι.

Οι περισσότερες εφαρμογές χρειάζεται να χρησιμοποιήσουν κάποια μέθοδο αυθεντικοποίησης του χρήστη. Έτσι, μπορούν να προσδιορίσουν το χρήστη και το ρόλο του, να αποθηκεύσουν δεδομένα για το συγκεκριμένο χρήστη και να παρέχουν εξατομικευμένες λειτουργίες με βάση τις προτιμήσεις του χρήστη (όπως για παράδειγμα εξατομικευμένες προτάσεις προϊόντων). Η αυθεντικοποίηση του χρήστη επιτυγχάνεται αντιστοιχίζοντας κάποια ένδειξη ταυτότητας. Υπάρχουν πολλοί διαφορετικοί τρόποι αυθεντικοποίησης, οι οποίοι παρέχουν διαφορετικά επίπεδα ασφαλείας μεταξύ τους. Ο πιο συνήθης και διαδεδομένος τρόπος αυθεντικοποίησης είναι η εισαγωγή ενός συνθηματικού το οποίο έχει οριστεί κατα την εγγραφή του χρήστη. Άλλοι μέθοδοι αυθεντικοποίησης είναι η επιβεβαίωση του χρήστη μέσω του κινητού του τηλεφώνου, η είσοδος με την αυθεντικοποίηση από τρίτους έμπιστους λογαριασμούς, η χρήση βιομετρικών στοιχείων κ.α.. Παρόλο που η αυθεντικοποίηση του χρήστη είναι μια εύκολη διαδικασία από την πλευρά του χρήστη, καθώς αποτελείται από ελάχιστα βήματα, είναι μια αρκετά περίπλοκη και χρονοβόρα διαδικασία για το πληροφοριακό σύστημα. Αυτο συμβαίνει διότι χρειάζεται να παρέχεται η δυνατότητα εγγραφής νέων χρηστών, σύνδεσης των ήδη υπάρχων χρηστών, διαχείρισης των στοιχείων των χρηστών και το πιο σημαντικό είναι πως πρέπει τα δεδομένα του κάθε χρήστη να διατηρούνται ασφαλής.

Η Firebase διαθέτει το δικό της σύστημα αυθεντικοποίησης του χρήστη, το Firebase Auth, το οποίο χρησιμοποιήθηκε και στη συγκεκριμένη εφαρμογή. Αποτελεί μέρος της βιβλιοθήκης ανοικτού κώδικα FirebaseUI και παρέχει υπηρεσίες backend, εύχρηστα και έτοιμα προς χρήση SDKs και έτοιμες βιβλιοθήκες για τον έλεγχο και την αυθεντικοποίηση του χρήστη. Επιπλέον, είναι υπηρεσία επί πληρωμή με βάση τις ανάγκες και τις απαιτήσεις του συστήματος. Υποστηρίζει πληθώρα διαφορετικών μεθόδων αυθεντικοποίησης του χρήστη, τις οποίες θα δούμε στη συνέχεια. Στόχος του Firebase Auth είναι η διαχείριση πολύπλοκων ροών του χρήστη κατα τη διάρκεια της σύνδεσης και της εγγραφής. Για να εγγυηθεί την ασφάλεια των στοιχείων των χρηστών, η Firebase Auth εφαρμόζει τις ίδιες μεθόδους και τεχνογνωσίες με αυτές που χρησιμοποιούνται στο Chrome Password Manager, στο Google Sign-in και στο Smart Lock μιας και όλα τα προαναφερθέντα αναπτύχθηκαν από την ίδια ομάδα, δηλαδή απο αυτην της Google.

Η διαδικασία της αυθεντικοποίησης απλοποιείται με την χρήση του Firebase Auth. Αρχικά, απαιτούνται τα διαπιστευτήρια του χρήστη. Τα διαπιστευτήρια μπορεί να είναι η διεύθυνση ηλεκτρονικού ταχυδρομείου και ο κωδικός πρόσβασης του χρήστη, ένα token που έχει εκδοθεί από έναν τρίτο έμπιστο πάροχο, ένας κωδικός μιας χρήσης ο οποίος στάλθηκε στο κινητό τηλέφωνο του χρήστη κ.α.. Έπειτα στέλνονται τα διαπιστευτήρια στο Firebase Authentication SDK, το οποίο με τη σειρά του πραγματοποιεί έλεγχο ορθότητας τους. Τέλος, το Firebase Authentication επιστρέφει μια απάντηση στην εφαρμογή, η οποία μπορεί να είναι ότι ο χρήστης επαληθευθηκε επιτυχως, οτι ο χρήστης δεν βρέθηκε, οτι τα διαπιστευτήρια δεν είναι σωστά και άλλα τυχόν προβλήματα που μπορεί να προέκυψαν κατα τη διάρκεια της επαλήθευσης. Μετά από την επιτυχημένη πρόσβαση του χρήστη, η εφαρμογή μπορεί να αποκτήσει πρόσβαση στις πληροφορίες προφίλ του χρήστη μιας και ο χρήστης επιστρέφεται

ως ένα αντικείμενο. Επιπλέον, μπορεί να γίνει έλεγχος της πρόσβασης του χρήστη σε δεδομένα που είναι αποθηκευμένα σε άλλες υπηρεσίες της Firebase, όπως για παράδειγμα στην online βάση δεδομένων την Firestore. Οι επικυρωμένοι χρήστες από προεπιλογή μπορούν να διαβάζουν και να γράφουν δεδομένα στο Firebase Realtime Database και στο Cloud Storage. Παρ'όλα αυτά τα δικαιώματα πρόσβασης των χρηστών μπορούν να αλλαχτούν τροποποιώντας τους κανόνες ασφαλείας Firebase Realtime Database και Cloud Storage.

Οι μέθοδοι αυθεντικοποίησης του χρήστη που υποστηρίζει το Firebase Auth είναι οι εξής:

- Χρήση διεύθυνσης ηλεκτρονικού ταχυδρομείου και κωδικού πρόσβασης
- Αποστολή κωδικού μιας χρήσης στο κινητό τηλέφωνο του χρήστη
- Third-Party Authentication δηλαδή η επιβεβαίωση του χρήστη από τρίτες έμπιστες υπηρεσίες. Πιο συγκεκριμένα, αυθεντικοποιείται μέσω του λογαριασμού που διαθέτει σε τουλάχιστον μια απο τις ακόλουθες υπηρεσίες: Google, Play Games, Game Center, Facebook, Twitter Github, Yahoo, Microsoft και Apple.
- Ανώνυμος χρήστης στον οποίο δίνεται η δυνατότητα να περιηγηθεί στην εφαρμογή με προσωρινό λογαριασμό και αν επιθυμεί μπορεί να εγγραφεί προκειμένου να μην χάσει τα δεδομένα που έχει μέχρι στιγμής.

Επιπλέον, υπάρχει η δυνατότητα να συσχετιστούν περισσότερες από μια μεθόδους αυθεντικοποίησης σε ένα χρήστη. Για παράδειγμα, ένας χρήστης μπορεί να συνδεθεί στο λογαριασμό του χρησιμοποιώντας το κινητό του τηλέφωνο για επιβεβαίωση ή χρησιμοποιώντας σύνδεση μέσω λογαριασμού Google.

Τέλος, το FirebaseUI είναι drop-in UI βιβλιοθήκη η οποία εφαρμόζει εύκολα και γρήγορα ολοκληρωμένες ροές χρηστών για όλες τις υποστηριζόμενες μεθόδους σύνδεσης του Firebase Authentication. Σε αρκετές περιπτώσεις οι εφαρμογές διαθέτουν τη δικιά τους ροή σύνδεσης του χρήστη προκειμένου να υπάρχει μεγαλύτερος έλεγχος. Σε τέτοιες περιπτώσεις μπορεί να γίνει χρήση του Custom Authentication της Firebase, το οποίο είναι μια προσαρμοσμένη λύση και με την χρήση του μπορούν να χρησιμοποιηθούν τμήματα του Firebase Authentication. Για παράδειγμα, υπάρχει η ροη σύνδεσης του χρήστη με διεύθυνση ηλεκτρονικού ταχυδρομείου και κωδικό και η εφαρμογή στέλνει τα διαπιστευτήρια στην Firebase προκειμένου να επαληθευτεί.

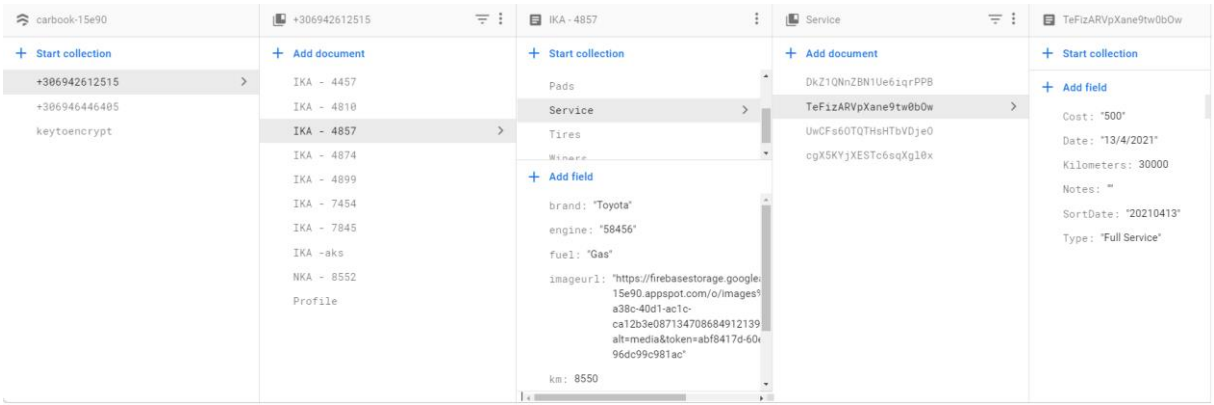
2.4.2 Cloud Firestore

Η Cloud Firestore είναι μια ευέλικτη, επεκτάσιμη βάση δεδομένων για ανάπτυξη κινητών, ιστού και διακομιστών η οποία διατίθεται και υποστηρίζεται απο την Firebase και το Google Cloud. Πρόκειται για μια βάση δεδομένων η οποία είναι cloud-hosted. Επιπλέον, η Firebase παρέχει πρόσβαση στη βάση δεδομένων Cloud Firestore σε πραγματικό χρόνο σε iOS, Android, JavaScript SDK, REST API, Admin SDK [16]. Το iOS και το Android SDK προορίζονται για ανάπτυξη εφαρμογών για κινητά με το αντίστοιχο λειτουργικό σύστημα. Το JavaScript SDK προορίζεται για εφαρμογές ιστού (web εφαρμογές). Το REST API μπορεί να χρησιμοποιηθεί από οποιαδήποτε γλώσσα προγραμματισμού. Τέλος, το Admin SDK χρησιμοποιείται για την ανάπτυξη εφαρμογών με JavaScript. Όλες οι συνδέσεις μεταξύ controller και SDK είναι κρυπτογραφημένες. Οι βασικοί λόγοι που επιλέχθηκε η Firestore στη συγκεκριμένη εφαρμογή είναι ότι πρόκειται για NoSQL, cloud-hosted βάση δεδομένων και ότι η πολιτική πληρωμών της βασίζεται στο μοντέλο pay as you grow.

2.4.2.1 NoSQL Database

Πρόκειται για μια NoSQL βάση δεδομένων το οποίο σημαίνει “Non-SQL” αλλά αρκετές φορές αναφέρεται ως “Not Only SQL” καθώς παρέχει μια πιο ευέλικτη προσέγγιση στην αποθήκευση δεδομένων. Ο όρος “NoSQL” χρησιμοποιήθηκε πρώτη φορά το 1998 απο τον Carlo Strozzi για την ανοιχτού κώδικα σχεσιακή βάση δεδομένων η οποία όμως δε χρησιμοποιούσε SQL [17]. Απο το 2009 και μετέπειτα ο όρος “NoSQL” χρησιμοποιείται για τις μη σχεσιακές βάσεις δεδομένων (non-relational). Παρόλου που οι NoSQL βάσεις δεδομένων υπάρχουν πολλά χρόνια, η χρήση τους αυξήθηκε δραματικά από τη στιγμή που ξεκίνησαν τα δεδομένα να αποθηκεύονται στο cloud και να χρησιμοποιούνται από διαφορετικές κινητές συσκευές. Επιλέγονται κυρίως γιατί είναι εύκολες στη χρήση και με υψηλή απόδοση.

Σε αντίθεση με τις παραδοσιακές βάσεις δεδομένων RDBMS, η Cloud Firestore είναι μια ευέλικτη και επεκτάσιμη βάση δεδομένων η οποία αποθηκεύει τις πληροφορίες σε μορφή JSON. Συγκεκριμένα αποτελείται από documents (έγγραφα) και σε collections (συλλογές). Κάθε collection είναι ένα οργανωμένο σύνολο από documents. Ένα document υποστηρίζει πολλούς διαφορετικούς τύπους δεδομένων, όπως για παράδειγμα κείμενο, αριθμούς, αντικείμενα κ.α.. Επίσης, υπάρχει η δυνατότητα να τοποθετηθούν collections εντός του document τα οποία αναφέρονται ως subcollections και δημιουργούνται ιεραρχικές δομές δεδομένων. Στη συγκεκριμένη εφαρμογή, κάθε χρήστης αποτελεί μια συλλογή δεδομένων ή αλλιώς ενα collection. Έπειτα, κάθε αυτοκίνητό του είναι ένα έγγραφο ή αλλιώς document. Το αυτοκίνητο με τη σειρά του αποτελείται από πολλά διαφορετικά subcollection αλλά και δεδομένα που περιέχουν τις βασικές του πληροφορίες. Κάθε subcollection είναι μια κατηγορία στην οποία μπορεί να πραγματοποιήσει αναβαθμίσεις ο χρήστης για το όχημά του. Πρόκειται για ένα collection που είναι εμφωλευμένο σε ένα αρχείο. Αυτό το collection περιλαμβάνει όλο το ιστορικό των αναβαθμίσεων που έχουν πραγματοποιηθεί στη συγκεκριμένη κατηγορία. Κάθε αναβάθμιση που έχει πραγματοποιηθεί είναι ένα έγγραφο το οποίο περιλαμβάνει τα στοιχεία της συγκεκριμένης αναβάθμισης, όπως για παράδειγμα την ημερομηνία, το κόστος κ.α.. Στο παρακάτω σχήμα (Εικόνα 2.6) παρατηρούμε αναλυτικά την ιεραρχία των δεδομένων της εφαρμογής στην Firestore βάση δεδομένων.



Εικόνα 2.6: Δεδομένα εφαρμογής αποθηκευμένα στην Firestore

Με την χρήση NoSQL βάση δεδομένων πετυχαίνουμε υψηλότερες αποδόσεις και λιγότερη πολυπλοκότητα στην βάση. Ένα παράδειγμα της υψηλής απόδοσης είναι η δυνατότητα της Google να επεξεργάζεται 20 petabyte την ημέρα που αποθηκεύονται στο Bigtable μέσω της προσέγγισης MapReduce. Στον παρακάτω πίνακα (Πίνακα 2.7) παρατηρούμε τις διαφορές μεταξύ της βάσης δεδομένων Firebase και μιας παραδοσιακής βάσης δεδομένων RDBMS.

Basis of comparison	Firebase	SQL(RDBMS)
Data Storage	Stored as JSON Tree	Stored in a Relational Model as Rows and Columns (Tables)
Schema flexibility	Dynamic Schema, data can be added, updated or deleted anytime	Fixed schema. Altering will result in going offline temporarily
Specialty	Data which has no definite type or Structure	Data whose type is known in advance
Technique	Synchronize data	Fire Query

Πίνακας 2.7: Σύγκριση μεταξύ Firebase και RDBMS [18]

2.4.2.2 Firestore ως Cloud-Hosted Database

Η Firestore είναι cloud-hosted βάση δεδομένων δηλαδή τα δεδομένα είναι αποθηκευμένα στο cloud της Google, για αυτό αρκετές φορές αναφέρεται ως Cloud Firestore. Αυτό έχει ως αποτέλεσμα να μην απειλείται η διαθεσιμότητα ή η ασφάλεια των δεδομένων. Στην προκειμένη περίπτωση, τα δεδομένα που αποθηκεύονται είναι οι αλλαγές που έχουν πραγματοποιηθεί σε κάθε αυτοκίνητο. Η απώλεια αυτών των δεδομένων θα είχε καταστροφικές συνέπειες τόσο για τον χρήστη καθώς θα έχανε όλο το ιστορικό των αλλαγών, όσο και για την εφαρμογή αφού δεν θα ήταν αξιόπιστη. Με την χρήση cloud-hosted βάση δεδομένων δεν υπάρχει κίνδυνος απώλειας των δεδομένων ακόμα και σε περίπτωση που φθαρεί ή χαθεί η συσκευή στην οποία είναι εγκατεστημένη η εφαρμογή.

Επιπλέον, είναι real-time βάση δεδομένων που σημαίνει ότι λειτουργεί σε πραγματικό χρόνο και ότι οι χρήστες βλέπουν άμεσα τις αλλαγές που πραγματοποιούνται.

2.4.2.3 Security of Firestore

Για να εξασφαλιστεί η ασφάλεια των δεδομένων η Firestore χρησιμοποιεί ισχυρή διαχείριση της πρόσβασης των χρηστών αλλά και έλεγχο δύο διαφορετικών μεθόδων, αναλόγως τις βιβλιοθήκες που χρησιμοποιούνται. Για παράδειγμα, για τις βιβλιοθήκες για εφαρμογές κινητών ή web εφαρμογές, χρησιμοποιείται το Firebase Authentication για την επικαιρότητα του χρήστη και το Cloud Firestore Security Rules για τον έλεγχο πρόσβασης στα δεδομένα. Αντιθέτως, για τις βιβλιοθήκες που χρησιμοποιούνται σε server-client χρησιμοποιείται το Identity and Access Management (IAM) για τον έλεγχο της πρόσβασης στα δεδομένα της βάσης.

2.4.3 Cloud Storage

2.4.3.1 Εισαγωγή

Στις περισσότερες εφαρμογές ο χρήστης επιθυμεί να αποθηκεύει και να μπορεί να κοινοποιεί τα αρχεία του. Η αποθήκευση των αρχείων μπορεί να επιτευχθεί εύκολα και γρήγορα χρησιμοποιώντας τον αποθηκευτικό χώρο της συσκευής στην οποία είναι εγκατεστημένη η εφαρμογή. Παρόλο αυτά, υπάρχουν αρκετά μειονεκτήματα όπως για παράδειγμα η διαθεσιμότητα των αρχείων, η ασφάλεια τους κ.α.. Τέλος, η τοπική αποθήκευση των αρχείων δεν εξυπηρετεί τους χρήστες στο να κοινοποιήσουν και να τα μοιραστούν τα αρχεία που επιθυμούν με άλλους χρήστες.

Η Firebase παρέχει μια ολοκληρωμένη λύση, το Cloud Storage. Το Cloud Storage είναι μια απλή, οικονομική αλλά ταυτόχρονα και ισχυρή υπηρεσία αποθήκευσης αντικειμένων. Έχει σχεδιαστεί και δημιουργηθεί για προγραμματιστές εφαρμογών και παρέχει αποθηκευτικό χώρο προκειμένου να αποθηκεύουν και να εξυπηρετούν περιεχόμενο που δημιουργείται από τους χρήστες των εφαρμογών. Τέτοιο περιεχόμενο μπορεί να είναι αρχεία εικόνων, ήχου, βίντεο ή άλλα αρχεία του χρήστη που χρησιμοποιεί μέσω της εφαρμογής.

Στη συγκεκριμένη εφαρμογή, το Cloud Storage χρησιμοποιήθηκε για την αποθήκευση εικόνων. Ο κάθε χρήστης μπορεί να επεξεργάζεται τα στοιχεία των οχημάτων που διαθέτει. Ένα από τα βασικά στοιχεία του οχήματος είναι και η φωτογραφία του. Η φωτογραφία αυτή μπορεί είτε να ληφθεί επιτόπου με την χρήση της φωτογραφικής κάμερας της συσκευής ή να επιλεχθεί από τις φωτογραφίες που ήδη υπάρχουν αποθηκευμένες στη συσκευή. Και στις δύο περιπτώσεις, η εικόνα του οχήματος αποθηκεύεται στο Cloud Storage. Τα πλεονεκτήματα και τους λόγους που επιλέχθηκε το Cloud Storage θα τα αναλύσουμε στην επόμενη ενότητα.

2.4.3.2 Πλεονεκτήματα χρήσης Cloud Storage

Κύριος λόγος επιλογής του Cloud Storage είναι το γεγονός ότι τα αντικείμενα είναι αποθηκευμένα στο cloud ώστε να είναι διαθέσιμα στον χρήστη ανεξαρτήτως τη συσκευή που χρησιμοποιεί, εξού και η ονομασία Cloud Storage. Με αυτόν τον τρόπο δεν απειλείται η διαθεσιμότητα των δεδομένων. Για την ακρίβεια, τα αρχεία αποθηκεύονται στο Google Cloud Storage bucket καθιστώντας τα έτσι προσβάσιμα μέσω του Firebase αλλά και του Google Cloud. Χρησιμοποιώντας το Firebase SDK για Cloud Storage μπορεί να επιτευχθεί upload και download των αρχείων. Επιπλέον, με την χρήση των Cloud Storage APIs ο διαχειριστής μπορεί να κάνει server-side επεξεργασία των αρχείων, όπως για παράδειγμα φιλτράρισμα των εικόνων.

Το Firebase SDK για το Cloud Storage εξασφαλίζει την ασφάλεια των αρχείων κατα τη διάρκεια του upload και του download από το cloud ανεξάρτητα από την ποιότητα του δικτύου. Επίσης, σε περίπτωση που η ποιότητα του διαθέσιμου δικτύου δεν επαρκεί για να ολοκληρωθεί η μεταφορά του αρχείου, τότε θα σταματήσει η διαδικασία αλλά με το που θα γίνει επανασύνδεση στο δίκτυο η μεταφορά θα συνεχιστεί απο το σημείο που σταμάτησε. Αυτό είναι πολύ χρήσιμο χαρακτηριστικό κυρίως όταν πρόκειται για μεταφορές μεγάλων αρχείων καθώς πάντα υπάρχει ο κίνδυνος διακοπής της σύνδεσης ή αργής ταχύτητας.

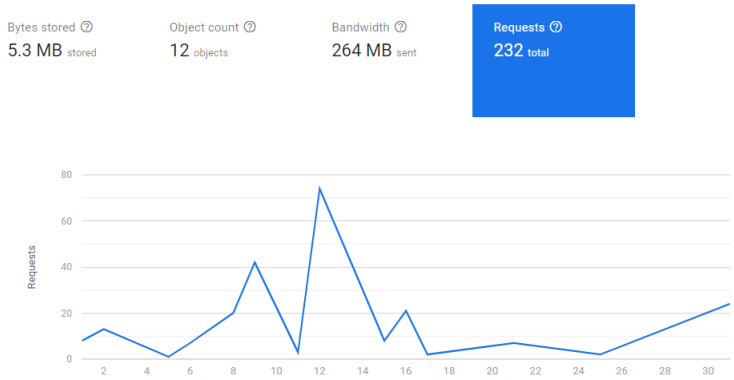
Ένα ακόμα πλεονέκτημα που έχει είναι η ασφάλεια των αρχείων όσων αφορά το διαμοιρασμό τους. Το Firebase SDK επιτρέπει το συνδυασμό του Cloud Storage με το Firebase Authentication έτσι ώστε να έχουν πρόσβαση στα αρχεία μονάχα οι επικυρωμένοι χρήστες που επιθυμούμε. Επιπλέον, δίνεται η δυνατότητα να δηλωθούν κανόνες στο μοντέλο ασφαλείας του Cloud Firestore προκειμένου

να επιτρέπεται η πρόσβαση στα αρχεία με βάση τα metadata τους, όπως για παράδειγμα το όνομα του αρχείου, το μέγεθος του κ.α..

Τελευταίο πλεονέκτημα αλλά εξίσου σημαντικό με τα προηγούμενα είναι η υψηλή δυνατότητα της επεκτασιμότητας του αποθηκευτικού χώρου. Το Cloud Storage σχεδιάστηκε έτσι ώστε να μπορεί να υποστηρίξει αποθηκευτικό χώρο μεγέθους έως και exabyte (1,0006 bytes), που σημαίνει ότι μπορεί να φιλοξενήσει εως και δισεκατομμύρια φωτογραφίες. Ο αποθηκευτικός χώρος κλιμακώνεται αυτόματα με αποτέλεσμα να μην υπαρχει η ανάγκη μετεγκατάστασης σε άλλο πάροχο ή αλλαγή πακέτου υπηρεσιών. Χρησιμοποιείται η ίδια υποδομή με αυτή του Google Photos και του Spotify.

2.4.3.3 Monitor Cloud Storage Activity

Η Firebase Console επιτρέπει στο διαχειριστή να ελέγξει τη χρήση του Cloud Storage, όπως για παράδειγμα τον αριθμό των Bytes που έχουν αποθηκευτεί, τα αιτήματα που πραγματοποιούνται από εφαρμογές, το μέσο Bandwidth κ.α.. Οι μετρήσεις αποθήκευσης, όπως για παράδειγμα τα αποθηκευμένα bytes και ο αριθμός των αντικειμένων, ενημερώνονται εντός 24 ωρών. Οι μετρήσεις σχετικά με την χρήση, όπως για παράδειγμα το brandwidth και ο αριθμός των αιτημάτων, ενημερώνονται κάθε λίγες ώρες. Στην παρακάτω εικόνα (Εικόνα 2.8) παρατηρούμε τις μετρήσεις του Cloud Storage που είναι διαθέσιμες στη συγκεκριμένη εφαρμογή και πιο συγκεκριμένα παρατηρούμε τα αιτήματα που έχουν πραγματοποιηθεί σε διάστημα ενός μήνα.



Εικόνα 2.8: Μετρήσεις αιτημάτων στο Cloud Storage

Επίσης, μέσω του Usage and Billing dashboard μπορεί να ελέγξει τις χρεώσεις με βάση τον αποθηκευτικό χώρο που χρησιμοποιείται. Το μοντέλο χρέωσης που εφαρμόζεται είναι το pay as you go που σημαίνει ότι αρχικά δίνεται η δυνατότητα δωρεάν χώρου αποθήκευσης και στο μετεπειτα ο χρήστης πληρώνει με βάση τις απαιτήσεις του και τον χώρο που χρειάζεται.

2.4.4 Firebase Cloud Messaging - Push Notifications

2.4.4.1 Εισαγωγή στα Push-Notifications

Οι αναδυόμενες ειδοποιήσεις είναι μηνύματα τα οποία εμφανίζονται στη συσκευή του χρήστη. Τέτοιου είδους ειδοποιήσεις μπορούν να εμφανιστούν εφόσον εκτελείται η εφαρμογή και δημιουργήθηκαν εξαιτίας μιας ενέργειας που έγινε στην εφαρμογή από τον χρήστη. Επίσης, μπορούν να δημιουργηθούν και σε περιπτώσεις που δεν εκτελείται η εφαρμογή και αυτό πετυχαίνεται με τη χρήση διακομιστή ο οποίος προωθεί το μήνυμα στη συσκευή του χρήστη. Με τη χρήση των αναδυόμενων ειδοποιήσεων η εφαρμογή μπορεί να αλληλεπιδρά με τον χρήστη ακόμα και όταν ο χρήστης δε βρίσκεται στη διεπαφή της εφαρμογής. Μέσω αυτών των ειδοποιήσεων μπορούν να γίνουν απλές ενέργειες, όπως για παράδειγμα η ενημέρωση του χρήστη για ένα σημαντικό συμβάν αλλά και πιο σύνθετες, όπως για παράδειγμα να αλληλεπιδράσει ο χρήστης με την εφαρμογή χωρίς όμως να μεταφερθεί στη διεπαφή της. Επιπλέον, με το πάτημα της αναδυόμενης ειδοποίησης ο χρήστης μπορεί να μεταφερθεί σε παράθυρο της επιλογής μας εντός της εφαρμογής.

Οι αναδυόμενες ειδοποιήσεις περιλαμβάνουν όλες τις απαραίτητες πληροφορίες για τη μετάδοση του μηνύματος. Μέσα σε αυτές, υπάρχουν και πληροφορίες σχετικά με τον service worker δηλαδή το χειριστή της ειδοποίησης. Ο service worker είναι αδρανής καθόλη τη διάρκεια ετσι ώστε να μην καταναλώνει πόρους. Ενεργοποιείται με το που ενημερωθεί ότι προωθήθηκε μια ειδοποίηση προκειμένου να την εμφανίσει και έπειτα επιστρέφει στην αρχική του κατάσταση. Επίσης, ενημερώνετε για τυχόν αλληλεπιδράσεις που γίνονται με την ειδοποίηση χωρίς όμως να χρησιμοποιεί πόρους. Για παράδειγμα, ο service worker εμφανίζει μια ειδοποίηση και έπειτα μεταβαίνει σε αδρανή κατάσταση. Στη συνέχεια με το που αλληλεπιδράσει ο χρηστης, π.χ. κάνοντας κλικ στην ειδοποίηση, ο service worker ενεργοποιείται χειρίζεται την αλληλεπίδραση και επιστρέφει σε αδρανή κατάσταση.

Στην παρούσα εφαρμογή η μεταφορά μηνυμάτων προς προβολή ως αναδυόμενες ειδοποιήσεις επιτεύχθηκε με την χρήση του Firebase Cloud Messaging το οποίο παρέχεται απο την Firebase και θα αναλύσουμε στην επόμενη ενότητα.

2.4.4.2 Firebase Cloud Messaging (FCM)

Το Firebase Cloud Messaging (FCM) είναι μια λύση πολλαπλών πλατφόρμων για μηνύματα και ειδοποιήσεις για εφαρμογές σε φορητές συσκευές, web και desktop εφαρμογες. Η υπηρεσία FCM παρέχεται επίσης απο την Firebase και πλέον υποστηρίζεται από την Google μιας και η Firebase πλέον ανήκει στην Google. Το Firebase Cloud Messaging (FCM) αντικατέστησε το Google Cloud Messaging (GCM) το 2016. Έχει κληρονομήσει την βασική υποδομή του αλλά εξελίχθηκε καθώς απλοποίησε την ανάπτυξη από την πλευρά του πελάτη. Αυτό έχει ως αποτέλεσμα ο τρόπος με τον οποίο χειρίζεται τα μηνύματα να είναι όμοιος με αυτόν την GCM. Το FCM έχει σχεδιαστεί να λειτουργεί σε αξιόπιστο περιβάλλον ώστε η υπηρεσία αποστολής ειδοποιήσεων να είναι αξιόπιστη και γρήγορη. Συγκεκριμένα το 98% των μηνυμάτων παραδίδονται και μάλιστα με καθυστέρηση ίση ή μικρότερη από 500 μιλιδευτερόλεπτα. Με την χρήση του Firebase Cloud Messaging (FCM) απλοποιείται και επιτυγχάνεται γρηγορότερα η διαδικασία σχεδιασμού και υλοποίησης εφαρμογών για κινητά καθώς και διευκολύνεται η διαδικασία αποστολής δοκιμαστικών μηνυμάτων με την χρήση του Notifications Composer στην κονσόλα Firebase. Μια ειδοποίηση μπορεί να μεταφέρει στην εφαρμογή μήνυμα χωρητικότητας έως 4KB.

Προκειμένου να γίνει χρήση της υπηρεσίας FCM χρειάζονται δύο βασικά στοιχεία ώστε να γίνει αποστολή και λήψη μηνυμάτων. Τα στοιχεία αυτά είναι τα εξής:

- Ένας διακομιστής ή ένα αξιόπιστο περιβάλλον μέσω του οποίου δημιουργούνται και αποστέλλονται τα μηνύματα. Ένα τέτοιο περιβάλλον θα μπορούσε να είναι το Cloud Functions που προσφέρεται από την Firebase.
- Η εφαρμογή του χρήστη που λαμβάνει και εμφανίζει τα μηνύματα μέσω της αντίστοιχης υπηρεσίας μεταφοράς συγκεκριμένης της πλατφόρμας. Η εφαρμογή αυτή μπορεί να είναι είτε Android είτε iOS είτε web εφαρμογή.

Όσον αφορά την εφαρμογή του χρήστη, απαιτείται ένα μοναδικό αναγνωριστικό (token) το οποίο παράγεται απο το FCM SDK και αντιπροσωπεύει τη συγκεκριμένη εφαρμογή που εκτελείται στη συσκευή απο τον παρόν χρήστη. Για να αποσταλεί μια ειδοποίηση χρειάζεται να περιέχει τις βασικές πληροφορίες του μηνύματος που επιθυμούμε να σταλεί αλλά και το μοναδικό αναγνωριστικό της ίδιας της εφαρμογής (token) ώστε να επιτευχθεί η αυθεντικοποίηση. Επιπλέον, απαιτείται να υπάρχει μια κεφαλίδα που υποδεικνύει το θέμα της ειδοποίησης. Η ειδοποίηση μπορεί να αποσταλεί είτε χρησιμοποιώντας το Admin SDK είτε μέσω των HTTP και XMPP APIs. Μια εφαρμογή μπορεί να είναι εγγεγραμμένη σε πολλαπλές κεφαλίδες θεμάτων, ή ακόμα και να δημιουργήσει νέο θέμα. Με αυτόν τον τρόπο, θα λαμβάνει τις ειδοποιήσεις που αποστέλλονται και προορίζονται για τα θέματα στα οποία είναι ήδη εγγεγραμμένη. Υπάρχουν τρεις κατηγορίες για αποστολή μηνυμάτων στην εφαρμογή. Ο πρώτος τύπος είναι η κατηγορία που είδαμε προηγουμένως, δηλαδή η αποστολή μηνυμάτων στις συσκευές που είναι εγγεγραμμένες σε ένα θέμα. Έπειτα, υπάρχει η δυνατότητα να σταλεί μήνυμα σε μια συσκευή μεμονωμένα. Τέλος, μπορούν να δημιουργηθούν ομάδες συσκευών έτσι ώστε να λάβουν το μήνυμα όλες οι συσκευές που ανήκουν στην ομάδα που θα επιλεχθεί ως παραλήπτης του μηνύματος. Ένα ακόμα βασικό πλεονέκτημα της υπηρεσίας FCM είναι ότι η εφαρμογή μπορεί να στείλει επιβεβαιώσεις, συνομιλίες και άλλου τύπου μηνύματα πίσω στο διακομιστή. Αυτό πετυχαίνεται με την χρήση του αξιόπιστου καναλιού σύνδεσης FCM, το οποίο είναι επίσης και αποδοτικό ως προς την χρήση της μπαταρίας της συσκευής.

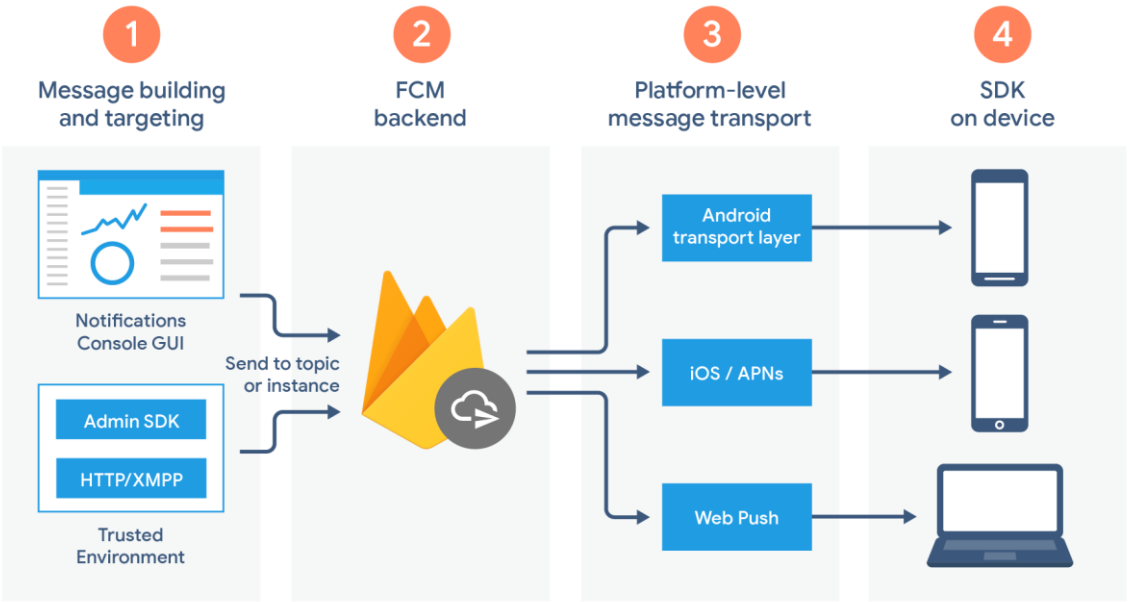
2.4.4.3 Αρχιτεκτονική Firebase Cloud Messaging

Η υπηρεσία Firebase Cloud Messaging (FCM) βασίζεται σε τέσσερα στοιχεία τα οποία δημιουργούν, μεταφέρουν και λαμβάνουν τα μηνύματα. Στην παρακάτω εικόνα (Εικόνα 2.9) παρατηρούμε τη σχηματική αναπαράσταση της αρχιτεκτονικής της υπηρεσίας Firebase Cloud Messaging. Αυτά τα στοιχεία στα οποία βασίζεται είναι τα εξής:

- Εργαλεία τα οποία είναι υπεύθυνα για τη σύνταξη ή τη δημιουργία αιτημάτων μηνυμάτων. Για πλήρη αυτοματοποίηση και υποστήριξη για όλους τους τύπους μηνυμάτων, πρέπει τα αιτήματα μηνυμάτων να δημιουργούνται σε ένα αξιόπιστο διακομιστή που υποστηρίζει το Firebase Admin SDK ή τα πρωτόκολλα διακομιστή FCM. Αυτό το περιβάλλον θα μπορούσε να είναι Cloud Functions για Firebase, App Engine ή το δικό σας διακομιστή εφαρμογών.
- Το FCM backend, το οποίο είναι υπεύθυνο για τον χειρισμό των μηνυμάτων ειδοποίησης. Επιτρέπει την παράδοση πτυσσόμενων αλλά και μη πτυσσόμενων μηνυμάτων.
- Το επίπεδο μεταφοράς στο επιπέδου πλατφόρμας, το οποίο δρομολογεί το μήνυμα προς τη στοχευμένη συσκευή, χειρίζεται την παράδοση μηνυμάτων και εφαρμόζει συγκεκριμένη διαμόρφωση πλατφόρμας όπου χρειάζεται. Στο επίπεδο μεταφοράς περιλαμβάνονται:
    - ➢ Το Android transport layer (ATL) για συσκευές με λειτουργικό σύστημα Android.

- ➢ Το Apple Push Notification service (APNs) για συσκευές με λειτουργικό σύστημα iOS.
- ➢ Πρωτόκολλο προώθησης μηνυμάτων για web εφαρμογές.

- Το FCM SDK στη συσκευή του χρήστη, όπου εμφανίζεται η ειδοποίηση ή γίνεται ο χειρισμός του μηνύματος σύμφωνα με την κατάσταση της εφαρμογής αλλά και οποιαδήποτε σχετική λογική ροή εργασιών της εφαρμογής.



Εικόνα 2.9: Αρχιτεκτονική Firebase Cloud Messaging

2.5 QR code

Ο κωδικός ταχείας απόκρισης (Quick Response code, στην συνέχεια θα αναφέρεται σαν QR code) είναι ένας δισδιάστατος γραμμικός κώδικας, ο οποίος μπορεί να αποθηκεύσει πληροφορίες δεδομένων. Εφευρέθηκε από την Ιαπωνική εταιρεια Denso Wave το 1994, η οποία είναι μια από τις μεγαλύτερες εταιρείες του ομίλου Toyota. Στόχος ήταν η γρήγορη αποκωδικοποίηση της πληροφορίας, γι' αυτο το λόγο ονομάστηκε κωδικός ταχείας απόκρισης, και η αποθήκευση πληροφορίας μεγαλύτερου όγκου.

Το QR code εγκρίθηκε ως διεθνές πρότυπο ISO το 2000 και η εταιρεία Denso Wave έκδωσε δίπλωμα ευρεσιτεχνίας στο δημόσιο τομέα προκειμένου να χρησιμοποιείται από οποιονδήποτε δωρεάν.

Το πλεονέκτημα του QR code, σε σχέση με τον παραδοσιακό γραμμικό κώδικα, είναι ότι μπορεί να περιέχει πολύ περισσότερα δεδομένα καθώς οι πληροφορίες κωδικοποιούνται τόσο την κατακόρυφη όσο και στην οριζόντια κατεύθυνση. Επιπλέον, υποστηρίζει διαφορετικούς χαρακτήρες οι οποίοι επιτρέπουν την υψηλότερη πυκνότητα δεδομένων. Ο παραδοσιακός γραμμικός κώδικας μπορεί να υποστηρίξει έως και 20 ψηφία ενώ το QR code υποστηρίζει έως 7089 χαρακτήρες. Στις παρακάτω εικόνες παρατηρούμε τα δεδομένα που είναι αποθηκευμένα σε έναν παραδοσιακό γραμμικό κώδικα (Εικόνα 2.10) και σε ένα QR code (Εικόνα 2.11).

Εικόνα 2.10: Παραδοσιακός γραμμικός κώδικας

Εικόνα 2.11: Κώδικας ταχείας απόκρισης

Δεδομένου ότι αναπτύχθηκε στην Ιαπωνία και προοριζόταν για χρήση από αυτοκινητοβιομηχανίες της Ιαπωνίας, το QR code υποστηρίζει επιπλέον και τους χαρακτήρες Kanji/Kana. Πρόκειται για ειδικούς χαρακτήρες της Ιαπωνικής γλώσσας και γίνεται πιο αποτελεσματική κωδικοποίηση κατα 20%. Η ίδια κωδικοποίηση μπορεί να εφαρμοστεί για οποιοδήποτε ειδικό χαρακτήρα σε κάθε γλώσσα, όπως για παράδειγμα εφαρμόζεται και στους κινέζικους ειδικούς χαρακτήρες.

Η πρόσβαση στα δεδομένα του QR code επιτυγχάνεται μέσω της λήψης φωτογραφίας του κώδικα από κάμερα, όπως για παράδειγμα απο την καμερα ενος smartphone. Σε αντίθεση με τον παραδοσιακό γραμμικό κώδικα, η φωτογραφία του QR code μπορεί να ληφθεί από οποιαδήποτε γωνία. Η συσκευή αναγνωρίζει την σωστή σειρά της πληροφορίας με την χρήση των τριών τετραγώνων τα οποία βρίσκονται στις τρεις γωνίες.

Η χωρητικότητα των πληροφοριών ενος QR code επηρεάζεται από την έκδοση του αλλά και από το επίπεδο διόρθωσης που θα εφαρμοστεί και τον τύπο της κωδικοποίησης. Παρέχονται 40 εκδόσεις/διαφορετικές προδιαγραφές QR code και 4 επίπεδα διόρθωσης (Πίνακας 2.1). Το επίπεδο διόρθωσης ορίζει τη διόρθωση σφαλμάτων που μπορεί να επιτευχθεί και τα επίπεδα είναι τα εξής:

- Επίπεδο L: σφάλματα τάξης μεγέθους 7% ή και λιγότερα μπορούν να διορθωθούν.
- Επίπεδο M: σφάλματα τάξης μεγέθους 15% ή και λιγότερα μπορούν να διορθωθούν.
- Επίπεδο Q: σφάλματα τάξης μεγέθους 25% ή και λιγότερα μπορούν να διορθωθούν.
- Επίπεδο H: σφάλματα τάξης μεγέθους 30% ή και λιγότερα μπορούν να διορθωθούν.

L	<=7%
M	<=15%
Q	<=25%
H	<=30%

Πίνακας 2.1:
Επίπεδα διόρθωσης

Το QR code αρχικά χρησιμοποιήθηκε από αυτοκινητοβιομηχανίες για την καταγραφή των ανταλλακτικών των οχημάτων. Έπειτα χρησιμοποιήθηκε όλο και σε περισσότερους τομείς με αποτέλεσμα στην σημερινή εποχή να χρησιμοποιείται καθημερινά σε πολλαπλές περιπτώσεις και σε διαφορετικές επιχειρήσεις. Ορισμένες από τους πιο δημοφιλής χρήσεις είναι οι εξής:

- Αποθήκευση ηλεκτρονικών διευθύνσεων/URL
- Πληροφορίες και στοιχεία όταν βρίσκεται σε αφίσες/πινακίδες
- Πληρωμές λογαριασμών/αμοιβών και αγορά αγαθών
- Σύνδεση και εγκατάσταση συσκευών στο δίκτυο

2.6 Adobe

Το Adobe Photoshop είναι ένα λογισμικό για υπολογιστές (macOS και Windows) και χρησιμοποιείται για την επεξεργασία και τον χειρισμό ψηφιακών εικόνων και γραφικών. Αποτελεί ένα πολύ χρήσιμο εργαλείο καθώς παρέχει πολλά διαφορετικά εργαλεία για την επεξεργασία εικόνων, όπως για παράδειγμα το εργαλείο περικοπής φωτογραφιών. Το Photoshop αναπτύχθηκε το από τους Thomas και John Knoll το 1987 και έπειτα το 1988 πούλησαν την άδεια στην Adobe Systems Incorporated και έκτοτε είναι διαθέσιμο από την Adobe Inc.. Πλέον είναι το πιο διαδεδομένο και δημοφιλές λογισμικό επεξεργασίας φωτογραφιών στον κόσμο. Επίσης, πρόκειται για ένα ισχυρό λογισμικό καθώς ο χρήστης έχει πληθώρα δυνατοτήτων επεξεργασίας. Χρησιμοποιείται από πληθώρα διαφορετικές βιομηχανίες, επαγγελματίες αλλά και από ερασιτέχνες για διάφορες χρήσεις, όπως για παράδειγμα για απλή επεξεργασία εικόνων έως και δημιουργία εταιρικής εικαστικής ταυτότητας.

Κατα τη διάρκεια των ετών έχει αναπτυχθεί, έχει αλλάξει ριζικά και έχουν προστεθεί νέες δυνατότητες. Η εκάστοτε ονομασία του Photoshop σχηματιζόταν με βάση τον αριθμό έκδοσης του. Η πρώτη έκδοση ήταν το Photoshop 0.63 η οποία όμως δεν ήταν διαθέσιμη στο κοινό. Τον Οκτώβριο του 2002 η ονομασία των προγραμμάτων άλλαξε καθώς εισήχθη το branding Creative Suite με αποτέλεσμα κάθε νέα έκδοση του Photoshop χαρακτηρίστηκε με "CS" συν τον αριθμό της έκδοσης, όπως για παράδειγμα το Adobe Photoshop CS6 η οποία ήταν και η τελευταία έκδοση της σειράς Creative Suite. Τον Ιούνιο του 2013 έγινε διαθέσιμη η έκδοση Adobe Photoshop CC, η οποία επιτρέπει στον χρήστη να συγχρονίζει τις προτιμήσεις του στο cloud αλλά και να ανεβάζει το έργο του. Επίσης άλλαξε η πολιτική τιμολόγησης και πλέον δεν απαιτείται σταθερή χρέωση για το λογισμικό αλλά χρέωση με βάση συνδρομής. Η τελευταία σταθερή έκδοση, που έγινε διαθέσιμη το 2021, περιλαμβάνει νέα εργαλεία για σύνθετη επεξεργασία, ανανεωμένες διεπαφές εργαλείων, καλύτερες αλλαγές στη ροή εργασίας και προστέθηκε η υποστήριξη για πρόσβαση και επεξεργασία κοινών εγγραφών στο cloud.

Στη συγκεκριμένη εφαρμογή το Photoshop χρησιμοποιήθηκε για το σχεδιασμό του εικονιδίου της εφαρμογής (Εικόνα 2.12) αλλά και των υπόλοιπων γραφικών στοιχείων που χρησιμοποιούνται εντός της εφαρμογής. Στις παρακάτω εικόνες παρατηρούμε τις εικόνες που σχεδιάστηκαν με την χρήση του Adobe Photoshop CC 2019 και χρησιμοποιούνται ως εικαστική ταυτότητα της εφαρμογής.

Εικόνα 2.12: Εικονίδιο εφαρμογής

Εικόνα 2.13: Γραφικό στοιχείο εντός της εφαρμογής

Ως κύριο χρώμα επιλέχθηκε το σκούρο μπλε και πιο συγκεκριμένα το ARGB(255,25,51,100). Με βάση την ψυχολογία των χρωμάτων, το μπλε χρώμα είναι φιλικό προς τον χρήστη και προκαλεί ξεκούραση. Επιπλέον, δημιουργεί το συναίσθημα της εμπιστοσύνης και της σταθερότητας. Αυτός ήταν και ο κύριος λόγος επιλογής του καθώς η εφαρμογή είναι υπεύθυνη για το ιστορικό αλλαγών των οχημάτων του χρήστη που σημαίνει ότι διαχειρίζεται ευαίσθητα και σημαντικά στοιχεία. Τέλος, τόσο στο εικονίδιο που δημιουργείται στο μενού εφαρμογών του κινητού όσο και στο logo το οποίο χρησιμοποιείται εντός της εφαρμογής, δόθηκε τρισδιάστατη μορφή.

2.7 Επίλογος

Για την ανάπτυξη της συγκεκριμένης εφαρμογής χρησιμοποιήθηκε το ανοιχτού κώδικα UI framework της Google, η Flutter η οποία χρησιμοποιεί την Dart ως γλώσσα προγραμματισμού. Προκειμένου η εφαρμογή να είναι όσο το δυνατόν πιο επίκαιρη στη σύγχρονη εποχή, ενσωματώθηκαν επιπλέον εργαλεία και υπηρεσίες. Επιπλέον έγινε χρήση των υπηρεσιών της Firebase, η οποία είναι μια πλατφόρμα ανάπτυξης εφαρμογών και παρέχεται επίσης από την Google. Πιο συγκεκριμένα, χρησιμοποιήθηκε οι εξής υπηρεσίες της Firebase. Το Firebase Auth προκειμένου να επιτευχθεί η αυθεντικοποίηση του χρήστη με την χρήση μονάχα του κινητού του τηλεφώνου. Έγινε Cloud Firestore ως κύρια βάση δεδομένων έτσι ώστε τα δεδομένα να είναι διαθέσιμα στον χρήστη ανεξαρτήτως της συσκευής που χρησιμοποιεί. Επιπλέον, έγινε χρήση του Cloud Storage για την αποθήκευση αρχείων φωτογραφίας στο Google Cloud. Τέλος, χρησιμοποιήθηκε και το Firebase Cloud Messaging για την προώθηση αναδυόμενων και μη ειδοποιήσεων στον χρήστη. Για τα γραφιστικά της εφαρμογής έγινε χρήση του λογισμικού Adobe Photoshop CC 2019.

Κεφάλαιο 3ο: Service Book App

3.1 Εισαγωγή

Στο παρόν κεφάλαιο θα γίνει αναλυτική περιγραφή της εφαρμογής Service Book, στον τρόπο λειτουργίας της αλλά και στη διεπαφή του χρήστη. Η εφαρμογή είναι γραμμένη στη γλώσσα προγραμματισμού Dart, την οποία μελετήσαμε στην ενότητα 2.3. Η εφαρμογή αποτελείται κυρίως από δύο βασικά κομμάτια. Το πρώτο είναι το κύριο μενού με τα οχήματα του χρήστη και τις λειτουργίες διαχείρισης τους, ενώ το δεύτερο είναι το παράθυρο διαχείρισης ενός συγκεκριμένου οχήματος, των στοιχείων του και των ενεργειών που μπορούν να πραγματοποιηθούν σε αυτό.

Στην πρώτη ενότητα περιγράφεται η λειτουργία σύνδεσης και αυθεντικοποίησης του χρήστη στην εφαρμογή. Αρχικά, πραγματοποιείται έλεγχος για το αν ο χρήστης είναι ήδη συνδεδεμένος και αν όχι τότε καλείται να συνδεθεί με την χρήση του τηλεφωνικού του αριθμού. Επίσης, γίνεται αναφορά στην υλοποίηση της αυθεντικοποίησης με το σύστημα της Firebase, το Firebase Auth.

Στην επόμενη ενότητα, στην ενότητα 3.3, γίνεται περιγραφή της κύριας οθόνης διαχείρισης οχημάτων. Γίνεται αναφορά τόσο στον τρόπο με τον οποίο σχεδιάζεται η διεπαφή του χρήστη όσο και στην υλοποίηση των ενεργειών που μπορεί να πραγματοποιήσει. Επιπλέον, στις υποενότητες περιγράφονται οι λειτουργίες που είναι διαθέσιμες στο χρήστη, όπως για παράδειγμα η αναζήτηση, η διαγραφή, η εισαγωγή και η επεξεργασία οχήματος.

Έπειτα, στην ενότητα 3.4 γίνεται αναφορά στο μενού εργασιών του χρήστη. Πρόκειται για αναδυόμενο μενου στην αριστερή πλευρά της οθόνης, το οποίο προσφέρει στον χρήστη τις βασικές λειτουργίες. Οι λειτουργίες οι οποίες προσφέρονται και θα αναλυθούν παρακάτω είναι η ενημέρωση σχετικά με τις βασικές πληροφορίες της εφαρμογής, η αναβάθμιση του προφίλ του χρήστη, οι οδηγίες χρήσης της εφαρμογής, ο έλεγχος για αναβαθμίσεις της εφαρμογής και η αποσύνδεση.

Τέλος, στην ενότητα 3.5, περιγράφεται το παράθυρο διαχείρισης συγκεκριμένου οχήματος. Μέσω αυτού του παραθύρου, δίνεται η δυνατότητα στο χρήστη να προσπελάσει τις αναβαθμίσεις που έχουν πραγματοποιηθεί είτε ανα κατηγορία είτε μαζικά με τη μορφή χρονοδιαγράμματος. Επιπλέον, αναλύεται ο τρόπος με τον οποίο γίνεται προσθήκη και διαγραφή μιας αναβάθμισης αλλά και η αλληλεπίδραση που γίνεται με την online βάση δεδομένων, την Firestore. Τέλος, αναλύεται η μέθοδος με την οποία δημιουργείται το QR code, το οποίο περιλαμβάνει κωδικοποιημένες τις απαραίτητες πληροφορίες για την κοινοποίηση του οχήματος.

3.2 Είσοδος χρήστη

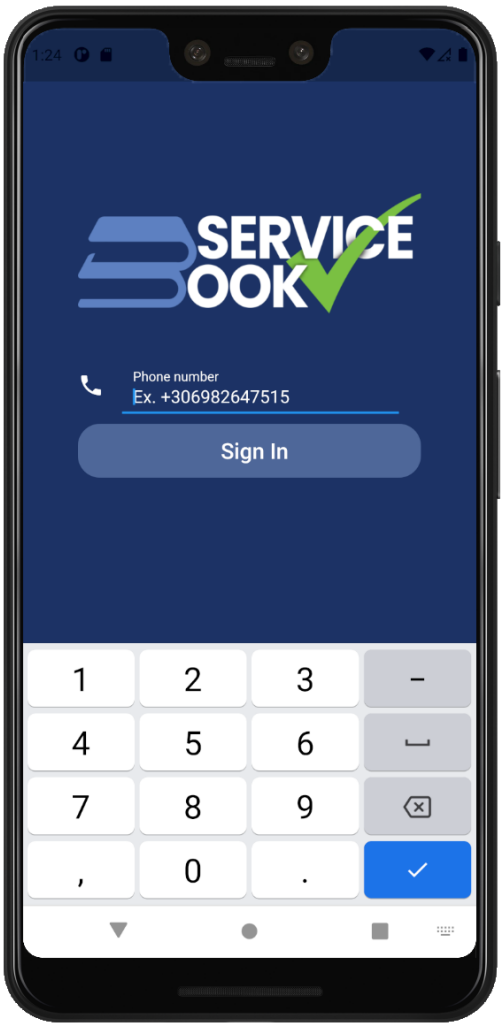
Η πρώτη οθόνη της εφαρμογής κατευθύνει τον χρήστη να συνδεθεί. Αρχικά, πραγματοποιείται έλεγχος αν ο χρήστης είχε συνδεθεί προηγουμένως και δεν αποσυνδέθηκε. Σε τέτοια περίπτωση, ο χρήστης μεταφέρεται κατευθείαν στο κυρίως μενού χωρίς να χρειαστεί να ξανασυνδεθεί. Αν ο χρήστης εισέρχεται πρώτη φορά στην εφαρμογή και επιθυμεί να εγγραφεί και όχι να συνδεθεί, τότε οι ενέργειες που χρειαζεται να κάνει είναι ακριβώς ίδιες με αυτές της σύνδεσης. Με αυτόν τον τρόπο, η εφαρμογή είναι πιο εύκολη στην χρήση καθώς δεν απαιτείται εγγραφή του χρήστη με πολλαπλά στάδια.

Η ταυτοποίηση και σύνδεση του χρήστη επιτυγχάνεται με την χρήση του κινητού τηλεφώνου του. Έγινε χρήση του συστήματος ελέγχου ταυτότητας της Firebase, το Firebase Auth, στο οποίο έγινε αναφορά προηγουμένως, στην ενότητα 2.4.1. Το κινητό τηλέφωνο προσφέρει πολλά πλεονεκτήματα που

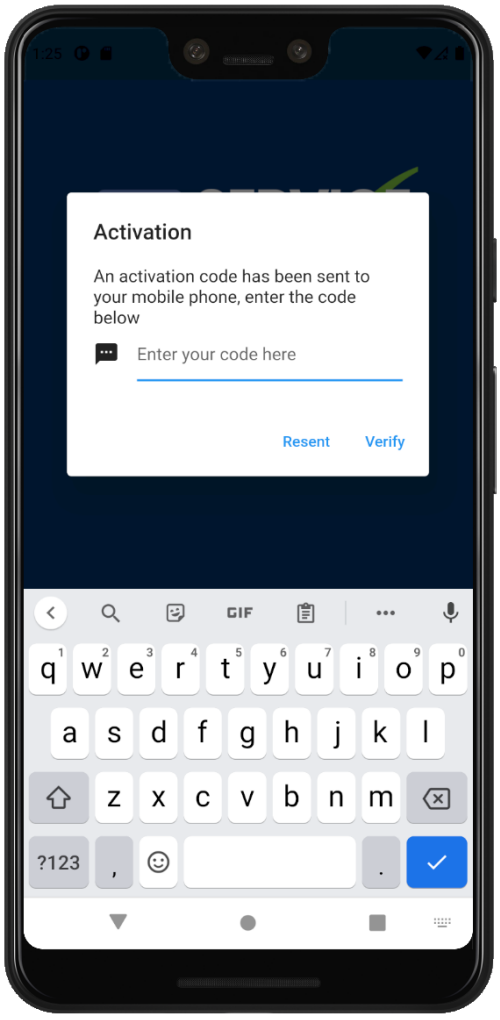
μπορούν να αξιοποιηθούν στον έλεγχο της ταυτότητας του χρήστη. Τα πιο σημαντικά εξ' αυτών είναι τα εξής:

- Τα περισσότερα άτομα διαθέτουν τουλάχιστον έναν αριθμό κινητού τηλεφώνου. Άρα κάθε πιθανός χρήστης μπορεί να πετύχει την αυθεντικοποίηση. Αντίθετα, σε περίπτωση 3rd party authentication, ενδεχομένως οι χρήστες να μη διαθέτουν λογαριασμούς στα μέσα που μπορούν να πιστοποιήσουν την ταυτότητά τους (όπως για παράδειγμα 3rd party authentication μέσω του λογαριασμού facebook).
- Το κινητό τηλέφωνο διαθέτει υπολογιστικές και επικοινωνιακές δυνατότητες οι οποίες κάνουν τη διαδικασία της αυθεντικοποίησης πιο αυτοματοποιημένη με αποτέλεσμα να είναι και πολύ πιο εύκολη για τον χρήστη. Αντιθέτως, η χρήση συνθηματικού προϋποθέτει ότι ο χρήστης θα κάνει εγγραφή, θα επιβεβαιώσει την ηλεκτρονική του διεύθυνση και έπειτα θα συνδεθεί με τα στοιχεία που έχει θέσει κατα την εγγραφή.
- Υπάρχουν ήδη αρκετοί και αξιόπιστοι μηχανισμοί ασφαλείας ενσωματωμένοι στο σύστημα GSM, οι οποίοι μπορούν να αξιοποιηθούν.

Στην εφαρμογή αρχικά παρουσιάζεται η διεπαφή σύνδεσης του χρήστη, η οποία αποτελείται απο το logo της εφαρμογής και το πλαίσιο σύνδεσης του χρήστη. Το logo είναι η γραφική αναπαράσταση του ονόματος της εφαρμογής, δηλαδή του "Service Book". Κατα την σύνδεση, ο χρήστης καλείται να συμπληρώσει τον τηλεφωνικό του αριθμό συμπεριλαμβανομένου το πρόθεμα που υποδεικνύει τη χώρα στην οποία είναι εγγεγραμμένος ο τηλεφωνικός αριθμός (Εικόνα 3.1). Για παράδειγμα, αν ο χρήστης διαθέτει ελληνική κάρτα τηλεφώνου (SIM card), τότε χρειάζεται να εισάγει το σύμβολο συν ('+') και στην συνέχεια το διεθνές πρόθεμα της ελλάδας, το οποίο είναι 030, και τέλος τον αριθμό του τηλεφώνου του. Στη συνέχεια, εμφανίζεται ένα αναδυόμενο παράθυρο, στο οποίο ο χρήστης χρειάζεται να εισάγει το μοναδικό του κωδικό (Εικόνα 3.2). Ο κωδικός αυτός, αποστέλλεται εντός ολίγων δευτερολέπτων στο κινητό τηλέφωνο του με την μορφή μηνύματος (SMS) και μπορεί να χρησιμοποιηθεί εντός ενός λεπτού καθώς έπειτα παύει να είναι ενεργός.



Εικόνα 3.1: Εισαγωγή αριθμού



Εικόνα 3.2: Εισαγωγή μοναδικού κωδικού

Η πιστοποίηση του χρήστη πετυχαίνεται με τη χρήση του Firebase Authentication API. Πιο συγκεκριμένα, χρησιμοποιείται η μέθοδος verifyPhoneNumber προκειμένου να επαληθευτεί ότι ο αριθμός τηλεφώνου, που εισήχθει απο το χρήστη, ανήκει πράγματι σε αυτό το χρήστη. Η Firebase στέλνει έναν κωδικό με τη μορφή μηνύματος και έπειτα το συγκρίνει με τον κωδικό που εισήγαγε ο χρήστης στην εφαρμογή. Κανένας κωδικός, ο οποίος στέλνεται με τη μορφή μηνύματος, δε μπορεί να αποτελεί διπλότυπο. Παρόλο αυτά, μπορεί να δοθεί η εντολή forceResendingToken, με την οποία αποστέλλεται ο ίδιος κωδικός. Η συγκεκριμένη εντολή δε χρησιμοποιείται στην εφαρμογή Service Book και όλοι οι κωδικοί είναι μοναδικοί. Ορισμένες συσκευές Android, παρέχουν αυτόματη επαλήθευση μέσω κωδικού με τη μορφή μηνύματος, επομένως ο χρήστης δε χρειάζεται να εισάγει τον κωδικό που έλαβε. Οι παράμετροι και οι ενέργειες που χρειάστηκαν να οριστούν στη μέθοδο verifyPhoneNumber είναι οι εξής:

- **phoneNumber**: Ορίζεται ο αριθμός τηλεφώνου με τον οποίο εγγράφεται ή συνδέεται ο χρήστης. Ο αριθμός αυτός αποτελείται από το σύμβολο συν (‘+’), το διεθνές κωδικό της χώρας και το τηλεφωνικό νούμερο του χρήστη.

```
phoneNumber: mobile
```

- **timeout**: Προσδιορίζει τον μέγιστο χρόνο που είναι ενεργός ο κωδικός επαλήθευσης, ο οποίος στέλνεται με την μορφή μηνύματος. Η μέγιστη επιτρεπτεί τιμή είναι τα δυο λεπτά αλλά στη συγκεκριμένη εφαρμογή το χρονικό διάστημα είναι ένα λεπτό. Έπειτα από το πέρας του χρονικού αυτού διαστήματος, ο κωδικός παύει να ισχύει και ο χρήστης χρειάζεται να αιτηθεί για καινούργιο κωδικό προκειμένου να εγγραφεί ή να συνδεθεί επιτυχώς.

```
timeout: Duration(seconds: 60)
```

- **verificationCompleted**: Σε αυτό το τμήμα, ορίζονται οι ενέργειες που θα γίνουν όταν ανακτηθεί αυτόματα το μήνυμα με τον κωδικό ή όταν επαληθευθεί ο τηλεφωνικός αριθμός.

```
verificationCompleted: (AuthCredential credential) async {
        UserCredential result =
            await
_auth.signInWithCredential(credential).catchError((e) {
          print(e);
        });
        User user = result.user;
        if (user != null) {
          Navigator.pushNamed(context, "/mainscreen", arguments:
user);
        } else {
          print("Error: null user");
        }
      }
```

- **verificationFailed**: Ενεργοποιήθηκε όταν προέκυψε σφάλμα κατά την επαλήθευση αριθμού τηλεφώνου. Ένα FirebaseAuthException παρέχεται όταν ενεργοποιείται. Στη συγκεκριμένη εφαρμογή, εμφανίζεται ένα αναδυόμενο παράθυρο το οποίο ενημερώνει το χρήστη οτι η αυθεντικοποίηση δεν ολοκληρώθηκε επιτυχώς.

```
verificationFailed: (FirebaseAuthException e) {
        if (e.code == 'invalid-phone-number') {
          showSingleDialogButton(context, "Error",
              Text('The provided phone number is not valid.'),
"OK", false);
          print('The provided phone number is not valid.');
        }
        //other errors
        else {
          showSingleDialogButton(context, "Error",
              Text(e.message), "OK", false);
          print(e.message);
        }
```

```
      }
```

- **codeSent**: Το τμήμα αυτό εκτελείται όταν το μήνυμα με τον κωδικό αποσταλεί στον χρήστη. Στην παρούσα εφαρμογή, εμφανίζεται ένα αναδυόμενο παράθυρο, το οποίο προτρέπει το χρήστη να εισάγει τον κωδικό που έλαβε. Το verifyCodePrompt είναι ένα custom widget που δημιουργήθηκε για την εφαρμογή και είναι υπεύθυνο για το αναδυόμενο παράθυρο εισαγωγής κωδικού. Την εμφάνιση του widget verifyCodePrompt την παρατηρήσαμε προηγουμένως στη εικόνα 3.2.

```
codeSent: (String verificationId, int resendToken) async {
        return verifyCodePrompt(context, mobile, verificationId,
_auth);
      }
```

- **codeAutoRetrievalTimeout**: Εκτελείται όταν λήξει ο κωδικός που έχει αποσταλεί με την μορφή μηνύματος και παρέχει ένα verificationID. Επομένως, εφόσον ορίστηκε ένα λεπτό ως το χρονικό διάστημα, κατα το οποίο είναι ενεργός ο κωδικός, η μέθοδος codeAutoRetrievalTimeout θα κληθεί μετά το πέρας ενός λεπτού.

```
codeAutoRetrievalTimeout: (String verificationId) {
        print(verificationId);
        print("Timeout of SMS code");
      }
```

3.2.1 SharedPreferences & stayLoggedIn

Κατα την εκκίνηση της εφαρμογής, πραγματοποιείται έλεγχος για το αν ο χρήστης παρέμεινε συνδεδεμένος, δηλαδή για το αν δεν πραγματοποίησε αποσύνδεση την τελευταία φορα που εκτέλεσε την εφαρμογή ή οχι. Ο έλεγχος επιτυγχάνεται με τα SharedPreferences και την μέθοδο stayLoggedIn. Στην περίπτωση σύνδεσης, με το που πραγματοποιηθεί μετάβαση στο κύριο παράθυρο της εφαρμογής, καλείται η μέθοδος stayLoggedIn, η οποία είναι υπεύθυνη να ενημερώσει την εφαρμογή οτι ο χρήστης συνδέθηκε επιτυχώς και μετέβη στην αρχική σελίδα.

Τα SharedPreferences χρησιμοποιούνται για την αποθήκευση δεδομένων με την μορφή τιμής-κλειδιού (key-values) σε Android και iOS. Στην Flutter, τα SharedPreferences χρησιμοποιούνται για Android και το NSUserDefaultson για iOS. Με αυτόν τον τρόπο, πετυχαίνεται μόνιμη αποθήκευση δεδομένων στην συσκευή. Τα SharedPreferences χρησιμοποιούνται κυρίως για αποθήκευση μικρών τιμών (πιθανών flags), οι οποίες χρησιμοποιούνται κατα την εκκίνηση της εφαρμογής. Με αυτόν τον τρόπο, μας δίνεται η δυνατότητα να γράφουμε και να διαβάζουμε εύκολα δεδομένα, με την μορφή τιμής-κλειδιού, σε μερικές μονάχα γραμμές κώδικα. Αντιθέτως, τα SharedPreferences δεν συνίσταται για την αποθήκευση σύνθετων σχεσιακών δεδομένων. Δεδομένου ότι Flutter SDK δε διαθέτει υποστήριξη των SharedPreferences, η χρήση τους επιτεύχθηκε με το πρόσθετο shared_preferences.

Αρχικά χρειάστηκε να γίνει αναφορά του πρόσθετου shared_preferences στο pubspec.yaml αρχείο, στο οποίο αναφέρονται όλα τα πρόσθετα που χρησιμοποιούνται στην εφαρμογή.

```
shared_preferences: ^2.0.5
```

Έπειτα έγινε εισαγωγή του πρόσθετου σε όλα τα αρχεία από τα οποία χρησιμοποιείται.

```
import 'package:shared_preferences/shared_preferences.dart';
```

Κατα την εκκίνηση της εφαρμογής, ελέγχεται η τιμή που έχει η μεταβλητή "islogged", η οποία είναι τύπου bool και ανήκει στα shared_preferences. Σε περίπτωση που η μεταβλητή δεν έχει τιμή τότε για τιμή της εκχωρείται το false. Ταυτόχρονα, καλείται και το preference με όνομα “phone”, στο οποίο αποθηκεύεται ο τηλεφωνικός αριθμός του χρήστη.

```
bool isloggedin = preferences.getBool("islogged") ?? false;
String phone = preferences.getString("phone");
```

Στην περίπτωση που ο χρήστης ήταν συνδεδεμένος, δηλαδή το preference “islogged” ‘ηταν true, τότε θα μεταβεί στο κυρίως μενού ειδάλλως στο παράθυρο σύνδεσης.

```
runApp(new MyApp(home: (isloggedin) ? MainScreen() : LoginPhoneWidget()))
```

Αντιθέτως, αν ο χρήστης δεν ήταν συνδεδεμένος και πραγματοποιήσει επιτυχή σύνδεση τότε αποθηκεύεται ο τηλεφωνικός του αριθμός στο preference “phone”.

```
preferences.setString("phone", phoneController.text.replaceAll(" ", ""));
```

Με την μετάβαση του χρήστη στην αρχική σελίδα της εφαρμογής, καλείται η μέθοδος stayLoggedIn προκειμένου να αποθηκευτεί το γεγονός ότι ο χρήστης συνδεθηκε επιτυχώς και μετέβει στο κύριο μενού.

```
void stayLoggedIn() async {
    SharedPreferences preferences = await
SharedPreferences.getInstance();
    preferences.setBool("islogged", true);
  }
```

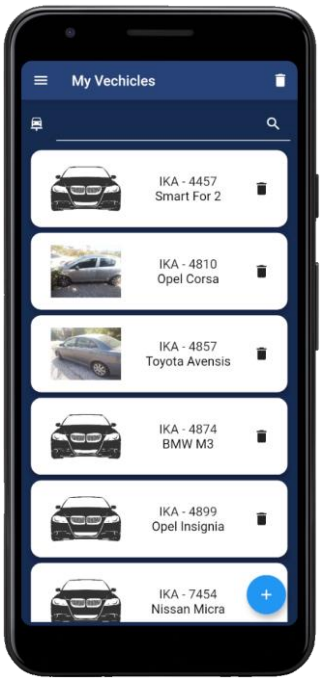
Τέλος, με την επιλογή του κουμπιού εξόδου από την εφαρμογή επιστρέφουν τα preferences στην αρχική τους κατάσταση, αποσυνδέεται ο χρήστης απο το FirebaseAuth και επιστρέφει στο παράθυρο σύνδεσης.

```
ListTile(
leading: Icon(Icons.logout,color: Colors.white),
              title: Text("Log out", style: TextStyle(color:
```

```
Colors.white)),
              onTap: () async {
                globals.gsettings.phone = "";
                globals.gsettings.savesettings();
                SharedPreferences preferences = await
SharedPreferences.getInstance();
                preferences.setBool("islogged", false);
                preferences.setString("phone", "");
                await FirebaseAuth.instance.signOut();
                Navigator.push( context,MaterialPageRoute(builder:
(context) => LoginPhoneWidget(),),);
              },
)
```

3.3 Κυρίως Μενού / Αρχική Σελίδα

Εφόσον ολοκληρωθεί επιτυχώς η σύνδεση ή η εγγραφή του χρήστη, τότε ο χρήστης μεταφέρεται στην αρχική σελίδα της εφαρμογής, η οποία αποτελεί και το κύριο μενού διαχείρισης των οχημάτων. Μέσω αυτού του παραθύρου, ο χρήστης έχει τη δυνατότητα διαχείρισης των οχημάτων τα οποία κατέχει. Πιο συγκεκριμένα, μπορει να πραγματοποιήσει αναζήτηση οχήματος, είτε με βάση τον αριθμό κυκλοφορίας του είτε με την μάρκα και το μοντέλο, να προσθέσει καινούργια οχήματα, να διαγράψει και να επεξεργαστεί τα ήδη υπάρχον οχήματα. Στην παρακάτω εικόνα (Εικόνα 3.3) παρατηρούμε τη διεπαφή της αρχικής σελίδας ενός λογαριασμού που κατέχει εννέα αυτοκίνητα (προβάλλονται τα έξι και τα υπόλοιπα είναι πιο κάτω στην συνέχεια).

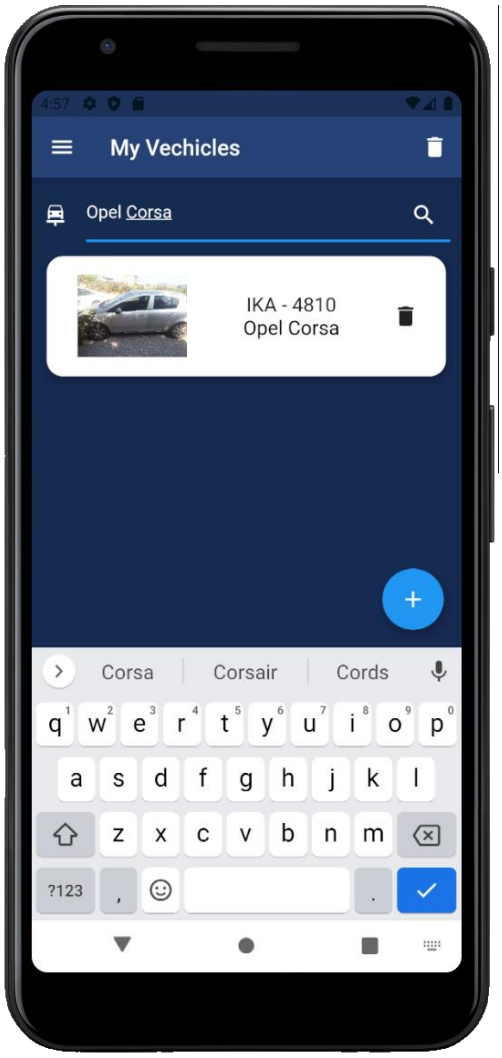


Εικόνα 3.3: Κυρίως μενού της εφαρμογής

3.3.1 Αναζήτηση

Ο χρήστης έχει τη δυνατότητα να πραγματοποιήσει αναζήτηση οχήματος. Το εργαλείο αυτό απευθύνεται σε χρήστες με πολλαπλά οχήματα, όπως για παράδειγμα εταιρείες. Η αναζήτηση οχήματος μπορεί να επιτευχθεί είτε με βάση τον αριθμό κυκλοφορίας του (Εικόνα 3.4) είτε με βάση την μάρκα ή το μοντέλο ή τον συνδυασμό τους (Εικόνα 3.5).

Εικόνα 3.4: Αναζήτηση βάση αρ. κυκλοφορίας.



Εικόνα 3.5: Αναζήτηση βάση μάρκας μοντέλου

Η μεταβλητή searchString αντιπροσωπεύει την εισαγωγή που έχει κάνει ο χρήστης προκειμένου να πραγματοποιήσει αναζήτηση οχήματος. Εφόσον το searchString αντιστοιχιστεί με οποιοδήποτε όχημα, που πληροί έναν από τους ελέγχους που προαναφέρθηκαν, τότε θα εμφανιστεί η καρτέλα του οχήματος αυτού. Αρχικά το searchString αρχικοποιείται ως “” επομένως αντιστοιχίζεται με όλα τα οχήματα και για αυτόν τον λόγο εμφανίζονται όλα.

```
String searchString = "";
```

Έπειτα για κάθε όχημα υπάρχει ο έλεγχος αντιστοιχίας.

```
(snapshot.data.docs[index].id.contains(searchString) ||
(snapshot.data.docs[index].get("brand") + " " +
snapshot.data.docs[index].get("model"))contains(searchString))
                                ? makeDismissibleCard(snapshot, index)
                                : Container()
```

Τέλος, χρειάζεται το πεδίο στο οποίο θα μπορεί ο χρήστης να εισάγει είτε τον αριθμό κυκλοφορίας είτε την μάρκα ή το μοντέλο του οχήματος ώστε να πραγματοποιήσει αναζήτηση. Η αναζήτηση πραγματοποιείται αυτόματα με το που εισαχθεί ή διαγραφεί ένας χαρακτήρας. Αυτό πετυχαίνεται με την χρήση του setState εντός του onChange του πεδίου.

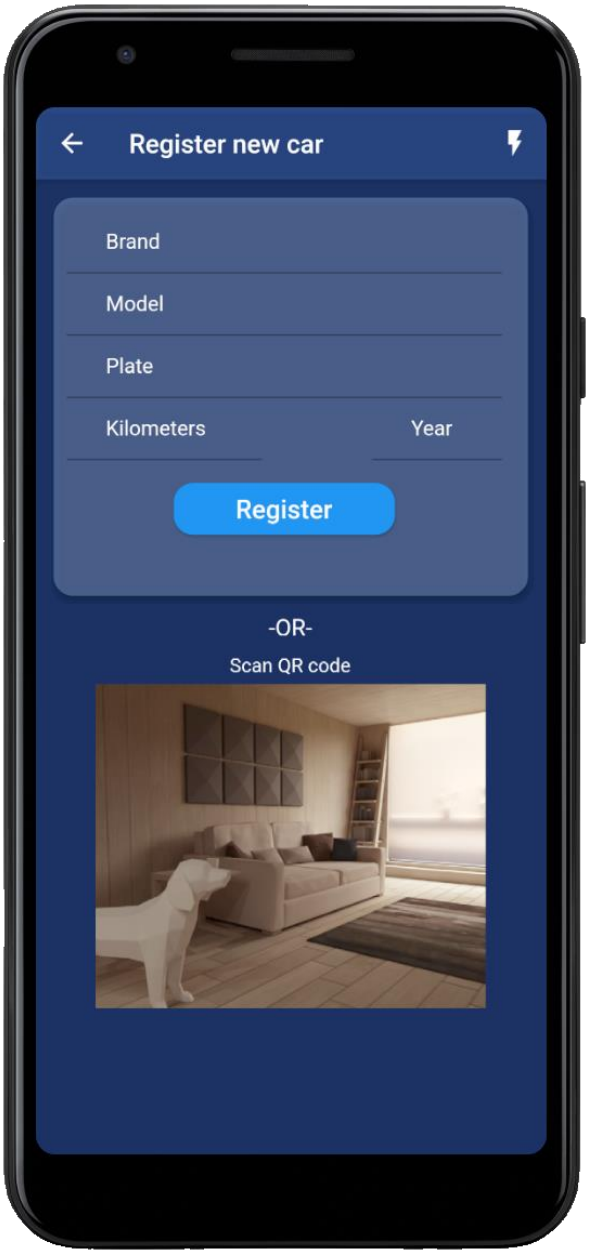
```
TextFormField(
              style: TextStyle(color: Colors.white),
              textCapitalization: TextCapitalization.words,
              focusNode: focusCars,
              controller: _carSearch,
              decoration: InputDecoration(
                enabledBorder: UnderlineInputBorder(
                  borderSide: BorderSide(color: Colors.white),
                ),
                suffixIcon: Icon(Icons.search, color: Colors.white),
                icon: Icon( Icons.car_repair, color: Colors.white),
              ),
              onChanged: (value) {
                setState(() {
                  searchString = value;
                });
              },
            )
```

3.3.2 Εισαγωγή οχήματος

Ο χρήστης έχει τη δυνατότητα να προσθέσει νέα οχήματα στην εφαρμογή ή και να προσπελάσει οχήματα που έχουν ήδη δημιουργηθεί από άλλους χρήστες. Η εισαγωγή ενός οχήματος επιτυγχάνεται με την επιλογή του κουμπιού συν (“+”), το οποίο βρίσκεται στην αρχική σελίδα της εφαρμογής. Με την επιλογή αυτού του κουμπιού, ο χρήστης μεταφέρεται σε ένα διαφορετικό παράθυρο το οποίο είναι υπεύθυνο για την εισαγωγή οχημάτων (Εικόνα 3.6). Παρακάτω παρατηρούμε την λειτουργία του κουμπιού συν (“+”), το οποίο είναι υπεύθυνο για την εναλλαγή παραθύρων αλλά και για τον χειρισμό του οχήματος το οποίο προστέθηκε.

```
floatingActionButton: FloatingActionButton(
```

```
        onPressed: () async {
          String sharedcar = await Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => AddCar()),
          );
          if (sharedcar != null) {
            String phone = sharedcar.split("|")[0];
            String plates = sharedcar.split("|")[1];
            Navigator.push(
              context,
              MaterialPageRoute(
                  builder: (context) => Options(
                        plt: plates,
                        collection: phone,
                        isowner: false,
                      )),
            );
          }
        },
        tooltip: 'Add new car',
        child: Icon(Icons.add),
      )
```



Εικόνα 3.6: Διεπαφή προσθήκης οχήματος

3.3.2.1 Εισαγωγή οχήματος χειροκίνητα

Όπως προαναφέρθηκε, για την εισαγωγή οχήματος ο χρήστης έχει δύο επιλογές. Στο επάνω μέρος της οθόνης μπορεί να εισάγει όχημα χειροκίνητα εισάγοντας τα βασικά του χαρακτηριστικά, όπως τον αριθμό κυκλοφορίας του, την μάρκα, το μοντέλο, τα χιλιόμετρα αλλά και το έτος κυκλοφορίας. Δεδομένου ότι ο αριθμός κυκλοφορίας ενός οχήματος είναι μοναδικός, ο χρήστης δεν επιτρέπεται να εισάγει αριθμό κυκλοφορίας, τον οποίο ήδη κατέχει, ή να αφήσει κενό το πεδίο του. Αρχικά, με την μέθοδο getCarsList(), δημιουργείται μια λίστα με τα ήδη υπάρχων οχήματα και τα στοιχεία τους.

```
void getCarList() async {
    QuerySnapshot querySnapshot = await
FirebaseFirestore.instance.collection(gsettings.phone).get();
    cars = querySnapshot.docs;
  }
```

Έπειτα, πραγματοποιείται έλεγχος οτι εισήχθει αριθμός κυκλοφορίας αλλά και έλεγχος του αριθμού κυκλοφορίας που εισήχθει σε σχέση με τους αριθμούς των κατοχυρωμένων οχημάτων, ωστέ να μην υπάρχει ήδη. Οι έλεγχοι αυτοί γίνονται απο το Widget το οποίο είναι υπεύθυνο για την αναπαράσταση του πεδίου εγγραφής του αριθμού κυκλοφορίας.

```
TextFormField(
textCapitalization: TextCapitalization.characters,
            controller: plateController,
            style: TextStyle(color: Colors.white),
            decoration: InputDecoration( contentPadding:
                              EdgeInsets.fromLTRB(30, 5, 30, 5),
                              labelText: "Plate",
                              labelStyle: TextStyle(color:
Colors.white),
                            ),
                            validator: (val) {
                              if (val.trim().isEmpty)
                                return "Enter the plates of the car";
                              for (var caro in cars) {
                                if (caro.id == val) {
                                  return "Car already exists";
                                }
                              }
                              return null;
                            },
                          ),
```

Τέλος, με την επιλογή του κουμπιού “Register”, δημιουργείται το όχημα και εκχωρείται στο συγκεκριμένο χρήστη. Στην βάση δεδομένων Firestore, δημιουργείται ένα καινούργιο document στο collection του χρήστη. Το document αυτό, έχει ως id τον αριθμό κυκλοφορίας του οχήματος, αποθηκευμένες τις βασικές πληροφορίες που εκχωρήθηκαν νωρίτερα και στην συνέχεια θα περιέχει subcollection με τις αλλαγές που θα πραγματοποιηθούν. Η μέθοδος addnewCar() είναι υπεύθυνη για τη δημιουργία του document του αυτοκινήτου.

```
void addnewCar() async {
```

```
    await databaseReference
        .collection(gsettings.phone)
        .doc(plateController.text.trim())
        .set({
      'brand': brandController.text.trim(),
      'model': modelController.text.trim(),
      'km': int.parse(kilometresController.text.trim()),
      'year': int.parse(yearConttroller.text.trim())
    }).then((value) => Navigator.pop(context));
  }
```

Εφόσον το document δημιουργηθεί επιτυχώς, ο χρήστης μεταφέρεται στην αρχική οθόνη, στην οποία εμφανίζεται πλέον και το καινούργιο του όχημα.

3.3.2.2 Προσπέλαση οχήματος με QR code

Εκτός από την χειροκίνητη εισαγωγή οχήματος, ο χρήστης έχει τη δυνατότητα να προσπελάσει ένα όχημα σκανάροντας το QR code του, ο οποίος δημιουργήθηκε από τον ιδιοκτήτη του οχήματος με την χρήση της εφαρμογής. Ο κωδικός QR περιλαμβάνει πληροφορίες σχετικά με το όχημα, οι οποίες είναι απαραίτητες προκειμένου ο χρήστης που θα το αποκωδικοποιήσει να μπορεί να αποκτήσει πλήρη ιστορικό των αλλαγών που έχουν πραγματοποιηθεί στο αυτοκίνητο αλλά και δικαιώματα να προσθέσει αλλαγές.

Το QR code μπορεί να δημιουργηθεί μονάχα από τον κύριο χρήστη, δηλαδή τον ιδιοκτήτη του οχήματος. Ο χρήστης που θα αποκωδικοποιήσει το QR code δεν έχει τα ίδια δικαιώματα για λόγους ασφαλείας. Δεν έχει δικαιώματα επεξεργασίας των στοιχείων του οχήματος αλλά ούτε και διαγραφής αλλαγών που έχουν πραγματοποιηθεί, έτσι ώστε να μην μπορεί να αλλοιωθεί το ιστορικό από χρήστη πέρα του ιδιοκτήτη. Έχει όμως δικαίωμα πρόσβασης στο ιστορικό αλλαγών και δικαίωμα να προσθέσει νέες αλλαγές καθώς ο χρήστης στον οποίο απευθύνεται το QR code είναι τα συνεργεία αυτοκινήτων. Επιπλέον, η πληροφορία του QR code είναι κωδικοποιημένη προκειμένου ο χρήστης που αποκωδικοποιεί το QR code να μη γνωρίζει πως δομείται η πληροφορία του, ώστε να μην μπορεί να δημιουργήσει όμοια QR code. Η κωδικοποίηση που χρησιμοποιείται είναι η AES και το κλειδί είναι 256 bit και βρίσκεται στην online βάση δεδομένων Cloud Firestore. Με το που πραγματοποιηθεί σκανάρισμα QR code γίνεται και αποκωδικοποίηση χρησιμοποιώντας το ίδιο κλειδί.

Για τη δημιουργία και την κοινοποίηση του QR code θα γίνει αναφορά σε επόμενη ενότητα. Στη συγκεκριμένη ενότητα, θα γίνει ανάπτυξη του τρόπου σκαναρίσματος QR code και αποκωδικοποίησης των πληροφοριών.

Αρχικά, με την επιτυχή σύνδεση του χρήστη η εφαρμογή λαμβάνει το κλειδί κωδικοποίησης μεγέθους 256 bit απο την Firestore. Το κλειδί είναι αποθηκευμένο στο collection “keytoencrypt”.

```
void getkey() async {
    DocumentSnapshot variable = await FirebaseFirestore.instance
        .collection("keytoencrypt")
```

```
        .doc("keyof32bytes")
        .get();
    setState(() {
      password = variable.data()["password"];
    });
  }
```

Έπειτα η μέθοδος _onQRViewCreated είναι υπεύθυνη για το σκανάρισμα του QR code. Η μέθοδος περιλαμβάνει έναν listener για την ανάγνωση πιθανόν QR code με την χρήση της κάμερας της συσκευής. Με το που ανιχνεύσει QR code η κάμερα σταματάει προσωρινά να λειτουργεί καθώς βρίσκεται σε κατάσταση παύση μέχρι την αποκωδικοποίηση του. Στην περίπτωση που η αποκωδικοποίηση του QR code δεν ολοκληρωθεί επιτυχώς, εμφανίζεται ανάλογο μήνυμα στον χρήστη και επαναφέρεται η κατάσταση της κάμερας προκειμένου να επιτρέπεται το σκανάρισμα ξανά. Περίπτωση λανθασμένου σκαναρίσματος QR code, μπορεί να είναι το σκανάρισμα QR code το οποίο δεν έχει δημιουργηθεί από την συγκεκριμένη εφαρμογή επομένως δεν έχει τη δομή που ακολουθείται.

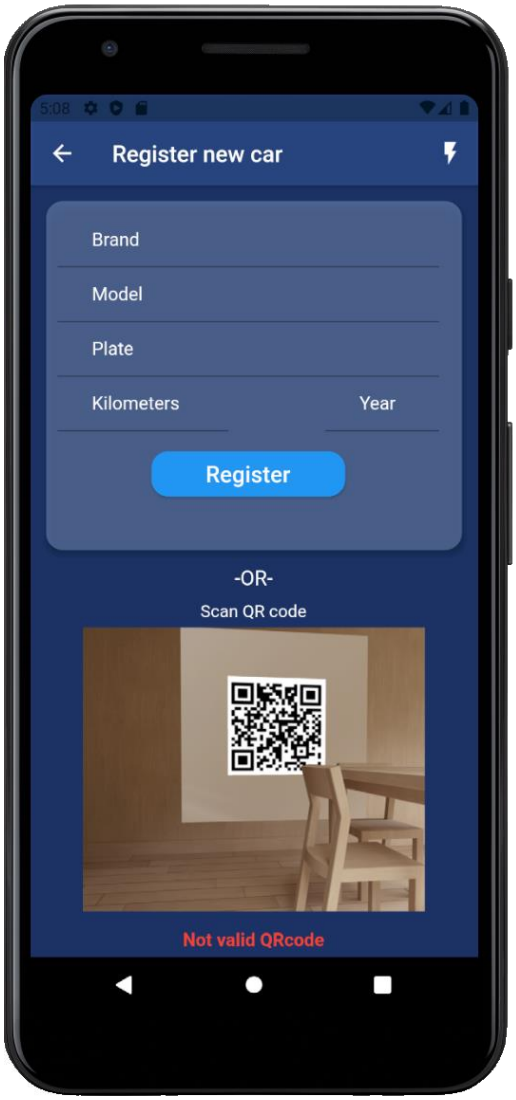
```
void _onQRViewCreated(QRViewController controller) {
    this.controller = controller;
    controller.scannedDataStream.listen((scanData) {
      String plainText = scanData.code;
      controller.pauseCamera();
      try {
        decrypted = encrypter.decrypt64(plainText, iv: iv);
        print(decrypted);
        Navigator.of(context).pop(decrypted);
      } catch (e) {
        print("Wrong QRcode");
        setState(() {
          error = true;
        });
        controller.resumeCamera();
      }
    });
  }
```

Επίσης, δίνεται η δυνατότητα ενεργοποίησης του φλας της συσκευής. Αρχικά το φλας είναι απενεργοποιημένο και με την επιλογή του κουμπιού με το ανάλογο εικονίδιο, το φλας είτε ενεργοποιείται είτε απενεργοποιείται.

Η εμφάνιση του μηνύματος λάθος επιτυγχάνεται με την χρήση Visibility Widget το οποίο θα εμφανιστεί στη διεπαφή μονάχα όταν η συνθήκη του είναι true. Στη συγκεκριμένη περίπτωση, ως συνθήκη χρησιμοποιείται το flag “error’’, το οποίο αρχικοποιείται ως false και γίνεται true μονάχα όταν δεν ολοκληρωθεί επιτυχώς η αποκωδικοποίηση του QR code.

```
Expanded(
                    flex: 1,
                    child: Visibility(
                      visible: error,
                      child: Center(
                        child: Text('Not valid QRcode',
                            style: TextStyle(color: Colors.white)),
                      ),
                    ),
                  )
```

Στην παρακάτω εικόνα (Εικόνα 3.7) παρατηρούμε τη διεπαφή όταν δοθεί λανθασμένο QR code.

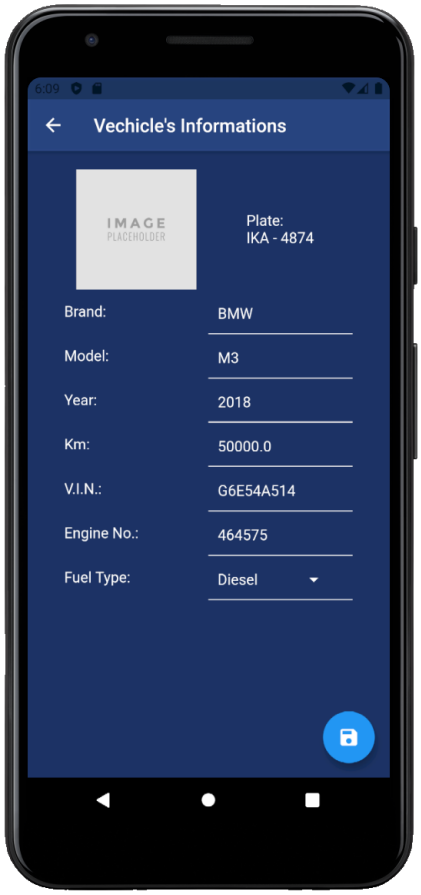


Εικόνα 3.7: Σκανάρισμα λανθασμένου QR code

Εφόσον η αποκωδικοποίηση του QR code ολοκληρωθεί επιτυχώς, τερματίζεται η μέθοδος και επιστρέφονται οι απαραίτητες πληροφορίες για την προσπέλαση του ιστορικού του οχήματος. Τέλος, ο χρήστης μεταβαίνει στο παράθυρο του οχήματος αλλά με λιγότερα δικαιώματα σε σχέση με τον ιδιοκτήτη του οχήματος. Επιτρέπεται στον χρήστη να προσπελάσει το ιστορικό των αλλαγών που πραγματοποιήθηκαν, να πάρει τα βασικά στοιχεία του οχήματος αλλά και να προσθέσει καινούργιες αλλαγές. Παρόλο αυτά, δεν έχει δικαίωμα να αφαιρέσει αλλαγές από το ιστορικό αλλά ούτε και να τροποποιήσει τα στοιχεία του οχήματος. Εφόσον ο χρήστης αποσυνδεθεί από το όχημα ή κλείσει την εφαρμογή, τότε χάνει την πρόσβαση στο όχημα, το όχημα δεν εμφανίζεται στα δικά του οχήματα και προκειμένου να επανακτήσει την πρόσβαση χρειάζεται να ξανα σκανάρει το QR code του οχήματος.

3.3.3 Επεξεργασία οχήματος

Ο χρήστης έχει τη δυνατότητα να επεξεργαστεί τα στοιχεία των οχημάτων που κατέχει και να προσθέσει τυχόν στοιχεία που δεν συμπληρώθηκαν κατα την εγγραφή του οχήματος. Αντιθέτως, ο χρήστης που σκανάρει QR code από άλλο όχημα δεν έχει το δικαίωμα να επεξεργαστεί τα στοιχεία του αλλά μονάχα να τα προσπελάσει. Με το παρατεταμένο πάτημα στην καρτέλα του οχήματος, ο χρήστης θα μεταφερθεί σε διαφορετικό παράθυρο (Εικόνα 3.8) το οποίο είναι υπεύθυνο για τα στοιχεία του οχήματος. Τα στοιχεία τα οποία αποθηκεύονται για κάθε όχημα είναι τα εξής: η μάρκα, το μοντέλο, το έτος κατασκευής, τα χιλιόμετρα, ο σειριακός του αριθμός, ο αριθμός της μηχανής, ο τύπος καυσίμου και η φωτογραφία του.

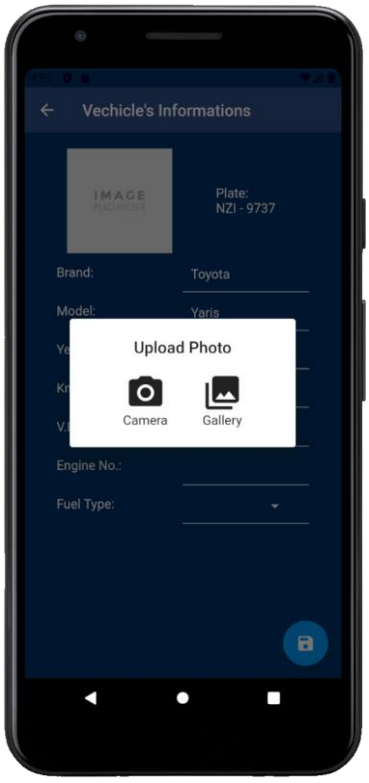


Εικόνα 3.8: Επεξεργασία στοιχείων οχήματος

Παρακάτω παρατηρούμε την μέθοδο που είναι υπεύθυνη για την μετάβαση στο παράθυρο των στοιχείων του οχήματος. Πρόκειται για μια απο τις μεθόδους που περιλαμβάνει το widget GestureDetector, το οποίο είναι επίσης υπεύθυνο για τη δημιουργία της κάρτας του οχήματος.

```
onLongPress: () {
          Navigator.push(
            context,
            MaterialPageRoute(
                builder: (context) => EditInfo(
                      plt: snapshot.data.docs[index].id,
                      collection: gsettings.phone,
                      isowner: true,
                    )),
          );
        }
```

Όσον αφορά την φωτογραφία του οχήματος, δίνεται η δυνατότητα αλλαγής της. Ο χρήστης μπορεί να επιλέξει να ανεβάσει συγκεκριμένη φωτογραφία από την συλλογή του ή να τραβήξει καινούργια φωτογραφία και να την αποθηκεύσει κατευθείαν ως φωτογραφία του οχήματος. Αρχικά, χρειάζεται να πατήσει εντός του πλαισίου της εικόνας. Έπειτα, θα εμφανιστεί ένα αναδυόμενο παράθυρο (Εικόνα 3.9) το οποίο θα τον προτρέπει να επιλέξει τρόπο εισαγωγής φωτογραφίας. Σε περίπτωση που πατήσει οπουδήποτε εκτός του παραθύρου, το παράθυρο θα κλείσει.



Εικόνα 3.9: Επιλογή εικόνας οχήματος

Εφόσον επιλέξει ο χρήστης τη μέθοδο με την οποία επιθυμεί να ανανεώσει την εικόνα του οχήματός του, καλείται η μέθοδος getImage με παράμετρο τη μέθοδο που επέλεξε ο χρήστης. Η μέθοδος getImage

είναι υπεύθυνη για την αλλαγή της εικόνας απο την καρτέλα του οχήματος, την συμπίεση της θέτοντας την ποιότητα της στο 60% αλλά και το κάλεσμα της μεθόδου uploadImageToFirebase, με την οποία επιτυγχάνεται η αποθήκευση της εικόνας στο Cloud Storage.

```
getImage(ImageSource source) async {
    final image = await picker.getImage(source: source, imageQuality:
60);
    if (image != null) {
      this.setState(() {
        _imageFile = File(image.path);
        uploadImageToFirebase();
      });
    } else {
      print("Uploading image failed");
    }
  }
```

Για την αποθήκευση των εικόνων χρησιμοποιείται ο αποθηκευτικός χώρος του Cloud Storage, όπως αναφέρθηκε προηγουμένως. Κάθε όχημα διαθέτει δικό του φάκελο στο Cloud Storage, στον οποίο αποθηκεύονται οι φωτογραφίες του. Η μέθοδος uploadImageToFirebase είναι τύπου Future και είναι υπεύθυνη για το ανέβασμα των εικόνων στο Cloud Storage και την αποθήκευση του url τους, το οποίο δημιουργείται εφόσον ανεβούν επιτυχώς και χρησιμοποιείται ως αναφορά απο το Firestore.

```
Future uploadImageToFirebase() async {
    String fileName = basename(_imageFile.path);
    String plate = widget.plt;
    FirebaseStorage storage = FirebaseStorage.instance;
    Reference reference = storage.ref().child('images/$plate/$fileName');
    UploadTask uploadTask = reference.putFile(_imageFile);
    TaskSnapshot taskSnapshot = await uploadTask;
    imageurl = await taskSnapshot.ref.getDownloadURL();
    this.setState(() {});
  }
```

Με το που ολοκληρωθεί επιτυχώς το ανέβασμα μιας φωτογραφίας γίνεται ανανέωση του παραθύρου επεξεργασίας στοιχείων το αυτοκινήτου και εμφανίζεται η καινούργια φωτογραφία. Σε περίπτωση που δεν έχει επιλεχθεί φωτογραφία, χρησιμοποιείται ως προεπιλεγμένη εικόνα οχήματος μια εικόνα που αναπαριστά ένα όχημα σε ασπρόμαυρο σχέδιο.

Τέλος όλες οι αλλαγές αποθηκεύονται όταν ο χρήστης επιλέξει το κουμπί αποθήκευσης, το οποίο έχει ως εικονίδιο τη δισκέτα αποθήκευσης. Εφόσον η συσκευή είναι συνδεδεμένη στο διαδίκτυο και όλα τα πεδία έχουν συμπληρωθεί, καλείται η μέθοδος updateVechicleInformation προκειμένου να ενημερώσει τα στοιχεία του οχήματος στην Firestore. Με την επιτυχή ενημέρωση των στοιχείων, ο χρήστης μεταφέρεται αυτόματα εκτός του παραθύρου επεξεργασίας των στοιχείων του οχήματος.

```
void updateVechicleInformation() {
```

```
    try {
databaseReference.collection(widget.collection).doc(widget.plt).update({
        'brand': brand.text.trim(),
        'model': model.text.trim(),
        'year': int.parse(year.text.trim()),
        'km': double.parse(km.text.trim()),
        'vin': vin.text.trim().toUpperCase(),
        'engine': engine.text.trim().toUpperCase(),
        'fuel': currentType,
        'imageurl': imageurl
      });
    } catch (e) {
      print(e.toString());
    }
    Navigator.pop(this.context);
  }
```

3.3.4 Διαγραφή οχήματος

Ο χρήστης έχει τη δυνατότητα να διαγράψει ένα όχημα από αυτά που κατέχει. Με τη διαγραφή του οχήματος, το όχημα αλλά και οι αλλαγές, που έχουν πραγματοποιηθεί σε αυτό, διαγράφονται από την βάση δεδομένων Firestore. Η διαγραφή μπορεί να επιτευχθεί είτε σέρνοντας την καρτέλα του οχήματος προς τα αριστερά είτε επιλέγοντας το κουμπί που έχει για εικονίδιο έναν κάδο. Και στις δύο περιπτώσεις, θα εμφανιστεί μήνυμα επιβεβαίωσης της διαγραφής του οχήματος μαζί με τα βασικά στοιχεία του. Το Widget Dismissible επιτρέπει το χρήστη να μπορεί να σύρει την κάρτα του οχήματος, χειρίζεται τα γραφικά και είναι υπεύθυνο για την ενέργεια που θα εκτελεστεί, δηλαδή για την εμφάνιση μηνύματος επιβεβαίωσης και τη διαγραφή του οχήματος. Επίσης, περιλαμβάνει την καρτέλα του οχήματος ως εμφωλευμένο widget.

```
Dismissible(
      key: UniqueKey(),
      direction: DismissDirection.endToStart,
      confirmDismiss: (direction) async {
        ConfirmPromt action = await confirm(
            context,
            Icon(Icons.delete),
            "Are you sure you want to delete " +
                snapshot.data.docs[index].id +
                "?",
            "NO",
            "YES");
        if (action == ConfirmPromt.YES) {
          deleteCar(snapshot.data.docs[index].id);
          return true;
```

```
        } else
          return false;
      }//then there is the card of the car
```

Ενώ, για τη διαγραφή του οχήματος με την επιλογή κουμπιού είναι υπεύθυνο το IconButton Widget στο οποίο ορίζεται μονάχα το εικονίδιο που θα έχει το κουμπί και οι ενέργειες που θα πραγματοποιηθούν εφόσον επιλεχθεί από τον χρήστη.

```
IconButton(
                    icon: Icon(Icons.delete),
                    onPressed: () async {
                      ConfirmPromt action = await confirm(
                          context,
                          Icon(Icons.delete),
                          "Are you sure you want to delete " +
                              snapshot.data.docs[index].id +
                              "?",
                          "NO",
                          "YES");
                      if (action == ConfirmPromt.YES) {
                        deleteCar(snapshot.data.docs[index].id);
                      }
                    },
                  )
```

Η εμφάνιση του μηνύματος επιβεβαίωσης επιτυγχάνεται με την μέθοδο confirm, η οποία είναι τύπου Future<ConfirmPromt>. Η μέθοδος αυτή είναι γενική και χρησιμοποιείται αρκετές φορές κατα τη διάρκεια της εφαρμογής και όχι μονάχα για διαγραφή οχήματος. Σαν παραμέτρους δέχεται

- το title, τύπου String και εμφανίζεται ως τίτλος του αναδυόμενου παραθύρου,
- το message, τύπου String και εμφανίζεται ως μήνυμα του αναδυόμενου παραθύρου. Για παράδειγμα στην συγκεκριμένη περίπτωση μπορεί να είναι “Are you sure you want to delete NKA -5332?”.
- δύο επιλογές, το option1 και option2, οι οποίες είναι τύπου String και είναι το κείμενο που εμφανίζεται στα κουμπιά του αναδυόμενου παραθύρου. Για παράδειγμα, “Yes” και “No”.

Η μέθοδος είναι τύπου Future<ConfirmPromt> καθώς επιστρέφει ασύγχρονα μια τιμή. Η τιμή αυτή είναι το ποιά από τις δύο διαθέσιμες επιλογές επιλέχθηκε από το χρήστη. Εφόσον επιβεβαιώσει τη διαγραφή του οχήματος, καλείται η μέθοδος deleteCar, η οποία είναι υπεύθυνη για τη διαγραφή των δεδομένων του οχήματος από τη βάση δεδομένων Firestore. Η μέθοδος deleteCar είναι τύπου void και δέχεται ως παράμετρο τον αριθμό κυκλοφορίας του οχήματος, ο οποίος αποτελεί και το μοναδικό αναγνωριστικό του.

```
void deleteCar(String plate) {
    try {
      databaseReference.collection(gsettings.phone).doc(plate).delete();
    } catch (e) {
```

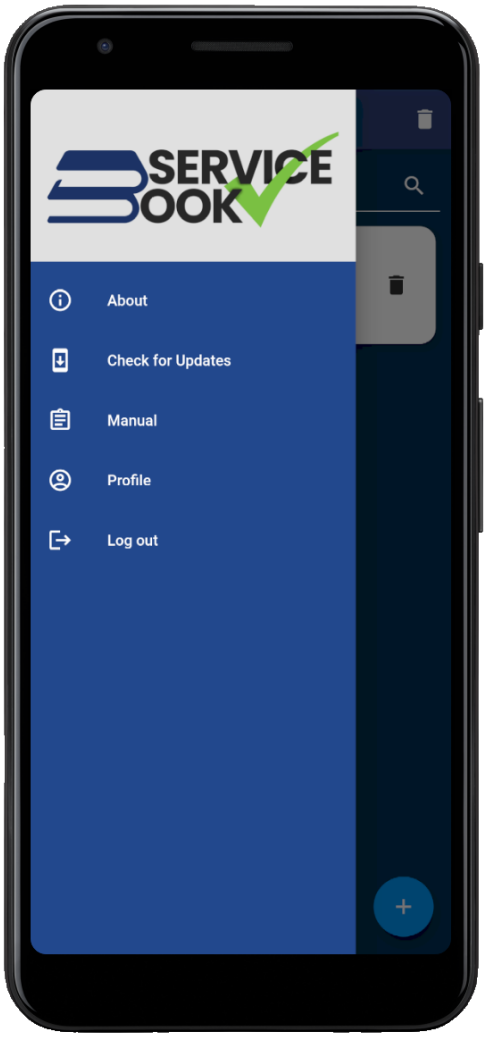
```
      print(e.toString());
    }
  }
```

Τέλος, δίνεται η δυνατότητα στον χρήστη να διαγράψει μαζικά όλα τα οχήματα του με την επιλογή του αντίστοιχου εικονιδίου που βρίσκεται στην κεντρική γραμμή εντολών. Η υλοποίηση είναι όμοια με τη διαγραφή ενός αυτοκινήτου, καθώς υπάρχει IconButton widget το οποίο εμφανίζει μήνυμα επιβεβαίωσης στον χρήστη. Παρόλο αυτά, αλλάζει το μήνυμα του αναδυόμενου παραθύρου και η μέθοδος που καλείται εφόσον επιβεβαιώσει ο χρήστης τη διαγραφή των οχημάτων. Στη συγκεκριμένη λειτουργία, καλείται η μέθοδος deleteAllCars η οποία είναι τύπου void, δε δέχεται παραμέτρους και διαγράφει όλα τα οχήματα που ανήκουν στο collection του χρήστη.

```
void deleteAllCars() {
    try {
      databaseReference.collection(gsettings.phone).get().then((snapshot)
{
        for (DocumentSnapshot car in snapshot.docs) {
          if (car.id != "Profile") car.reference.delete();
        }
      });
    } catch (e) {
      print(e.toString());
    }
  }
```

3.4 Μενού χρήστη

Το μενου του χρήστη περιλαμβάνει βασικές λειτουργίες σχετικά με την εφαρμογή. Προκειμένου να εμφανιστεί το μενού, χρειάζεται ο χρήστης να επιλέξει το αντίστοιχο εικονίδιο που βρίσκεται στην αρχική σελίδα, το οποίο συχνά αναφέρεται ως “hamburger menu icon”. Υπεύθυνη για το σχεδιασμό και τη λειτουργικότητα του μενου είναι η κλάση GetDrawer. Το μενου αποτελείται από την εικόνα-logo της εφαρμογής και πέντε λειτουργίες διαθέσιμες για τον χρήστη (Εικόνα 3.10).

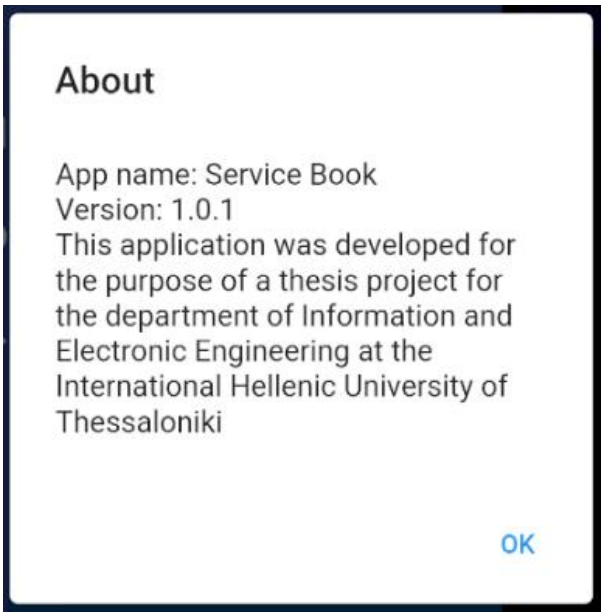


Εικόνα 3.10: Μενου χρήστη

Οι λειτουργίες, οι οποίες είναι διαθέσιμες στον χρήστη, είναι οι εξής.

- **About**

  Εμφανίζει αναδυόμενο παράθυρο με τα βασικά χαρακτηριστικά της εφαρμογής. Η εμφάνιση του παραθύρου επιτυγχάνεται με την χρήση της μεθόδου showSingleDialogButton, η οποία είναι τύπου Future και δέχεται ως παραμέτρους τον τίτλο, το μήνυμα, την επιλογή που θα είναι διαθέσιμη στον χρήστη με τη μορφή κουμπιού και το flag το οποίο προσδιορίζει αν το παράθυρο θα κλείνει με το πάτημα του χρήστη εκτός κουμπίου. Στην προκειμένη περίπτωση, το μήνυμα έχει την παρακάτω μορφή (Εικόνα 3.11) και μπορεί να κλείσει επιλέγοντας μονάχα το κουμπί-επιλογή που δίνεται.



Εικόνα 3.11: Αναδυόμενο παράθυρο About

- **Check for Updates**

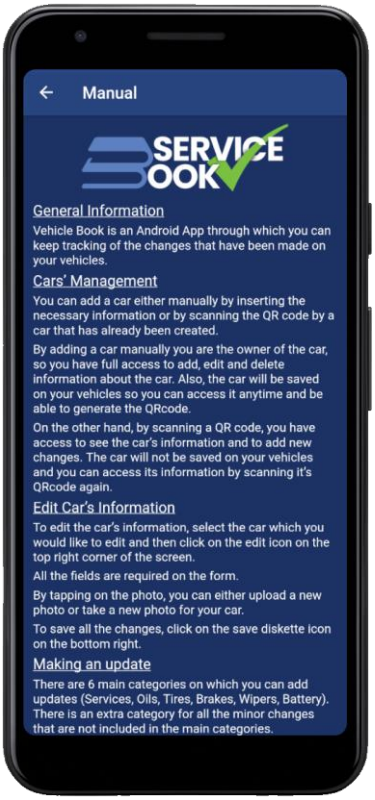
  Η επιλογή “Check for Updates” ελέγχει αν υπάρχουν διαθέσιμες ενημερώσεις της εφαρμογής στο Play Store. Δεδομένου ότι απευθύνεται μονάχα σε συσκευές με λειτουργικό σύστημα Android, η επιλογή αυτή είναι διαθέσιμη μονάχα σε αυτές. Ο έλεγχος για τυχόν ενημερώσεις επιτυγχάνεται με την μέθοδο τύπου Future<void> checkForUpdate. Αν υπάρχει διαθέσιμη ενημέρωση, πραγματοποιείται αυτόματη αναβάθμιση της εφαρμογής. Ειδάλλως, εμφανίζεται αντίστοιχο μήνυμα με την μορφή μπάρας στην κάτω πλευρά της διεπαφής. Σε περίπτωση που ο έλεγχος δεν ολοκληρωθεί λόγω σφάλματος, εμφανίζεται μήνυμα λάθους, το οποίο έχει επίσης μορφή μπάρας και περιλαμβάνει το σφάλμα που διέκοψε τον έλεγχο. Τέτοια σφάλματα μπορεί να είναι η μη εύρεσης της εφαρμογής στο Play Store, η χαμηλή μπαταρία της συσκευής κ.α..

```
AppUpdateInfo _updateInfo;
  // Platform messages are asynchronous, so we initialize in an
async method.
  Future<void> checkForUpdate() async {
    InAppUpdate.checkForUpdate().then((info) {
      setState(() {
        _updateInfo = info;
        if (_updateInfo?.updateAvailability ==
            UpdateAvailability.updateAvailable) {
          InAppUpdate.performImmediateUpdate().catchError((e) => {
                ScaffoldMessenger.of(context)
                    .showSnackBar(SnackBar(content:
Text(e.toString())))
              });
        } else {
          ScaffoldMessenger.of(context).showSnackBar(SnackBar(
```

```
              content: Text("There are no currently updates
aveible")));
        }
      });
    }).catchError((e) => {
          if (e.toString().contains("Install Error(-6)"))
            {
              ScaffoldMessenger.of(context).showSnackBar(SnackBar(
                  content: Text(
                      "Check for updates is not avaiable due to the
device")))
            },
          ScaffoldMessenger.of(context)
              .showSnackBar(SnackBar(content: Text(e.toString())))
        });
  }
```

- **Manual**

  Με την επιλογή του “Manual”, ο χρήστης ανακατευθύνεται σε νέο παράθυρο (Εικόνα 3.12), το οποίο περιλαμβάνει οδηγίες χρήσης της εφαρμογής σχετικά με όλες τις λειτουργίες που υποστηρίζει.



Εικόνα 3.12: Manual εφαρμογής

- **Profile**

Με την επιλογή “Profile”, ο χρήστης μπορεί να επεξεργαστεί τα προσωπικά του στοιχεία, όπως για παράδειγμα το ονοματεπώνυμο του ή το όνομα της επιχείρησης του. τον φορολογικό αριθμό της επιχείρησης κ.α.. Το παράθυρο σχετικά με τα στοιχεία του χρήστη πετυχαίνεται απο την κλάση UserProfile. Η λειτουργικότητα είναι όμοια με αυτήν του παραθύρου επεξεργασίας στοιχείων του οχήματος που αναφέρθηκε προηγουμένως στην υποενότητα 3.3.3. Υπάρχουν τα πεδία που απαιτούνται να συμπληρωθούν από το χρήστη και εφόσον υπάρχει σύνδεση στο διαδίκτυο τότε ανανεώνονται τα στοιχεία του στην Firestore και αποχωρεί αυτόματα από το παράθυρο.

- **Log out**

  Με την επιλογή “Log out”, ο χρήστης αποσυνδέεται από την εφαρμογή και την επόμενη φορά θα χρειαστεί να επανασυνδεθεί με τον αριθμό του κινητού του τηλεφώνου. Το preference “islooged” παίρνει τιμή false και το preference “phone” παίρνει κενή τιμή. Έπειτα ο χρήστης μεταφέρεται στο παράθυρο σύνδεσης.

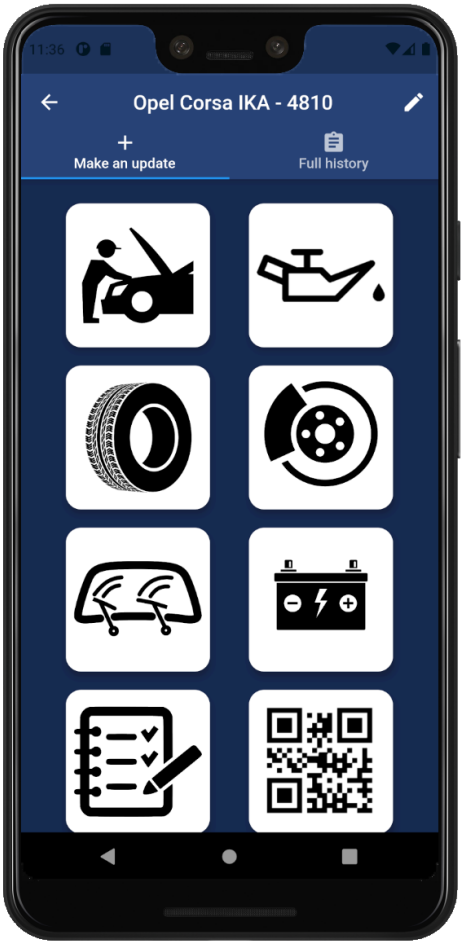
```
onTap: () async {
                globals.gsettings.phone = "";
                globals.gsettings.savesettings();
                SharedPreferences preferences =
                    await SharedPreferences.getInstance();
                preferences.setBool("islogged", false);
                preferences.setString("phone", "");
                await FirebaseAuth.instance.signOut();
                Navigator.pushReplacementNamed(context, "/login");
              },
```

3.5 Διαχείριση οχήματος

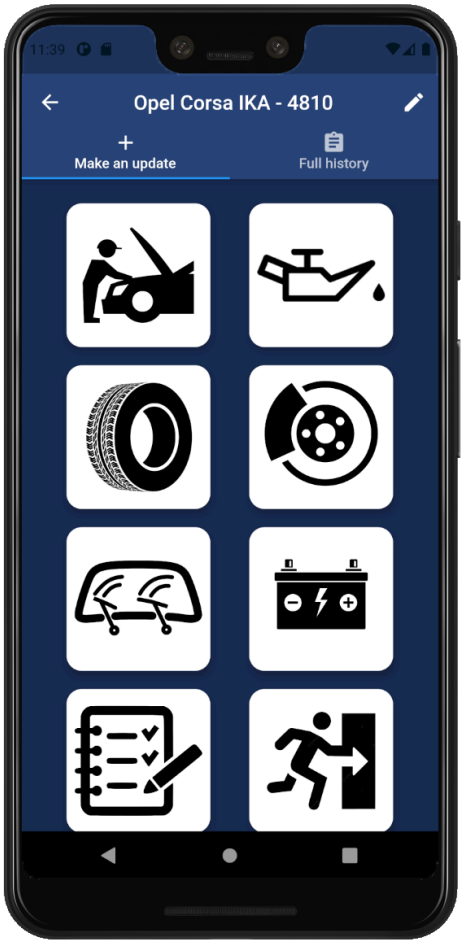
Πατώντας την καρτέλα του οχήματος, ο χρήστης μεταφέρεται στο παράθυρο διαχείρισης του. Εφόσον ο χρήστης είναι ο ιδιοκτήτης, μπορεί να προσθέσει αναβαθμίσεις που πραγματοποιούνται στο όχημα, να διαγράψει αναβαθμίσεις, να προσπελάσει το ιστορικό του οχήματος από την ημέρα εγγραφής του αλλά και να δημιουργήσει QR code για την κοινοποίηση του οχήματος (Εικόνα 3.13). Σε περίπτωση που ο χρήστης δεν είναι ο ιδιοκτήτης αλλά τρίτο πρόσωπο, του επιτρέπεται μονάχα πρόσβαση στο ιστορικό και προσθήκη νέων αλλαγών (Εικόνα 3.14). Το παράθυρο διαχείρισης οχήματος αποτελείται από δύο tabs, το tab στο οποίο οι αλλαγές είναι κατηγοριοποιημένες και το tab απο το οποίο ο χρήστης μπορεί να προσπελάσει όλο το ιστορικό του οχήματος. Οι αλλαγές/αναβαθμίσεις χωρίζονται σε επτά κατηγορίες, οι οποίες είναι οι εξής:

- Προσθήκη Service
- Αλλαγή λαδιών
- Αλλαγή ελαστικών
- Αλλαγή φρένων
- Αλλαγή μπαταρίας
- Αλλαγή υαλοκαθαριστήρων

- Προσθήκη σημείωσης για αλλαγές που τυχόν δεν περιλαμβάνονται στις προαναφερθέντες κατηγορίες, όπως για παράδειγμα αλλαγή αισθητήρα λ.

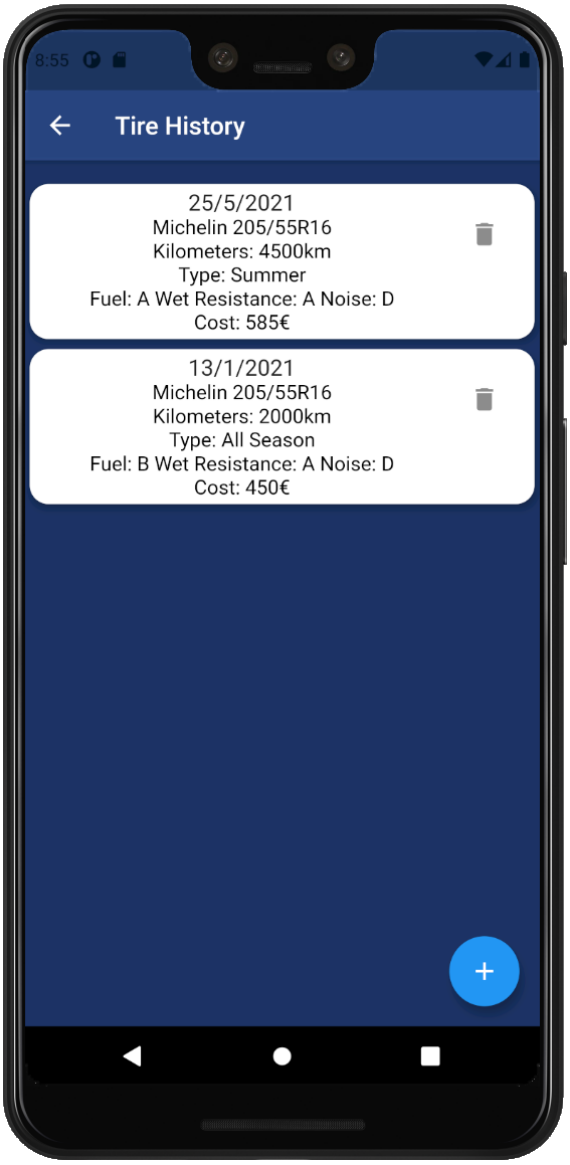


Εικόνα 3.13: Διαχείριση οχήματος από τον ιδιοκτήτη



Εικόνα 3.14: Διαχείριση οχήματος απο τρίτο χρήστη

Επιλέγοντας την κατηγορία που επιθυμεί ο χρήστης, μεταφέρεται σε ένα καινούργιο παράθυρο (Εικόνα 3.15), στο οποίο εμφανίζονται οι αναβαθμίσεις που έχουν πραγματοποιηθεί στη συγκεκριμένη κατηγορία. Για κάθε κατηγορία υπάρχει και αντίστοιχη κλάση που είναι υπεύθυνη για το σχεδιασμό και τη λειτουργικότητα του παραθύρου. Τόσο ο σχεδιασμός όσο και η λειτουργικότητα, είναι όμοια μεταξύ των κατηγοριών. Σε περίπτωση που δεν έχει πραγματοποιηθεί καμία αναβάθμιση, εμφανίζεται αντίστοιχο μήνυμα. Οι αναβαθμίσεις που έχουν πραγματοποιηθεί εμφανίζονται με τη μορφή καρτέλας. Ως τίτλος της καρτέλας είναι η ημερομηνία κατα την οποία πραγματοποιήθηκε η αναβάθμιση, και χρησιμοποιείται επίσης ως κριτήριο ταξινόμησης των καρτελών προκειμένου να εμφανίζονται στο επάνω μέρος του παραθύρου οι πιο πρόσφατες αναβαθμίσεις. Επιπλέον, στο υπόλοιπο τμήμα της καρτέλας εμφανίζονται πληροφορίες σχετικά με την αναβάθμιση, όπως για παράδειγμα το κόστος της, η μάρκα που χρησιμοποιήθηκε κ.α..



Εικόνα 3.15: Ιστορικό αλλαγών κατηγορίας ελαστικών

3.5.1 Εμφάνιση αναβαθμίσεων

Για τη λήψη των δεδομένων απο την Firestore χρησιμοποιήθηκε το widget StreamBuilder, το οποίο έχει τη δυνατότητα να δέχεται ροές δεδομένων και να επιστρέφει widgets αλλά και το στιγμιότυπο των ληφθέντων δεδομένων ροής.

```
StreamBuilder(
                  stream: FirebaseFirestore.instance
                      .collection(widget.collection)
                      .doc(widget.plt)
                      .collection("Tires")
                      .orderBy('Kilometers', descending: true)
                      .snapshots(),
```

```
                    builder: (BuildContext context,
                        AsyncSnapshot<QuerySnapshot> snapshot) {
```

Έπειτα, ελέγχεται η τρέχουσα κατάσταση σύνδεσης του στιγμιότυπου, έτσι ώστε καθόλη τη διάρκεια της μεταφοράς δεδομένων να εμφανίζεται αντίστοιχο γραφιστικό στο χρήστη, δηλαδή ο κύκλος φόρτισης.

```
if (snapshot.connectionState == ConnectionState.waiting) {
                        return (Center(child:CircularProgressIndicator()));
}
```

Τέλος ελέγχεται ο αριθμός των δεδομένων που λήφθηκαν. Αν ο αριθμός των δεδομένων είναι μηδενικός, δηλαδή δεν υπάρχουν αναβάθμισης στην συγκεκριμένη κατηγορία, τότε εμφανίζεται αντίστοιχο μήνυμα εντος widget στο χρήστη.

```
if (snapshot.data.docs.length == 0) {
                        id = null;
                        return Center(
                            child: Column(children: [
                          Text(
                            "It seems that there are no changes on tires",
                            style: TextStyle(color: Colors.white),
                          ),
                          Text(
                            "Press the + button to perform a tires'
change",
                            style: TextStyle(color: Colors.white),
                          )
                        ]));
                      }
```

Ενώ, αν υπάρχουν αναβαθμίσεις, επιστρέφεται το widget ListView.builder, το οποίο είναι υπεύθυνο για τη δημιουργία μιας scrollable σειράς γραφιστικών στοιχείων, τα οποία είναι οι κάρτες.

```
return ListView.builder(
                          shrinkWrap: true,
                          itemCount: snapshot.data.docs.length,
                          itemBuilder: (context, index) {
                            id = snapshot.data.docs.first.id;
```

Εφόσον ο χρήστης είναι ο ιδιοκτήτης του οχήματος, η κάρτα της αναβάθμισης είναι εμφωλευμένο widget εντος του widget Dismissable, το οποίο δίνει τη δυνατότητα στο εσωτερικό του widget να μπορεί να γίνει dismiss προς την ορισμένη κατεύθυνση. Εφόσον ο χρήστης δεν είναι ιδιοκτήτης, δεν έχει δικαίωμα διαγραφής της αναβαθμισης επομένως δε γίνεται χρήση του widget Dismissable αλλά κατευθείαν της κάρτας. Για την δημιουργία της κάρτας δημιουργήθηκε το custom widget getCard. Στο

περίγραμμα των καρτών έχει δοθεί η ιδιότητα BorderRadius.circular(15.0), με την οποία γίνονται πιο κυκλικές οι τέσσερις γωνίες της κάρτας. Επιπλέον, με την παράμετρο elevation: 5, η κάρτα αποκτά σκιά προκειμένου ο σχεδιασμός να είναι πιο φιλικός προς τον χρήστη.

```
Widget getCard(QueryDocumentSnapshot snapshot) {
    return Card(
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(15.0),
      ),
      elevation: 5,
      child: ListTile(
        title: Text((snapshot.get("Date")),
            textAlign: TextAlign.center, style: TextStyle(fontSize: 18)),
        trailing: (widget.isowner)
            ? IconButton(
                icon: Icon(Icons.delete),
                onPressed: () async {
                  bool isconnected = await
CheckInternet().check(context);
                  if (isconnected) {
                    ConfirmPromt action = await confirm(
                        context,
                        Icon(Icons.delete),
                        "Are you sure you want to delete the tire of " +
                            snapshot.get("Date") +
                            "?",
                        "NO",
                        "YES");
                    if (action == ConfirmPromt.YES) {
                      deleteTire(snapshot.id);
                    }
                  }
                },
              )
            : Text(""),
        subtitle: Column(
          children: [
            Text((snapshot.get('Brand') + " " + snapshot.get('Code')),
                textAlign: TextAlign.center,
                style: TextStyle(color: Colors.black, fontSize: 16)),
            Text("Kilometers: " + snapshot.get('Kilometers') + "km",
                textAlign: TextAlign.center,
                style: TextStyle(color: Colors.black, fontSize: 16)),
            Text("Type: " + snapshot.get('Type'),
                textAlign: TextAlign.center,
                style: TextStyle(color: Colors.black, fontSize: 16)),
            Text(
```

```
                ("Fuel: " +
                    snapshot.get('Fuel') +
                    " Wet Resistance: " +
                    snapshot.get('Wet') +
                    " Noise: " +
                    snapshot.get('Noise')),
                textAlign: TextAlign.center,
                style: TextStyle(color: Colors.black, fontSize: 16)),
            Text("Cost: " + snapshot.get('Cost') + "€",
                textAlign: TextAlign.center,
                style: TextStyle(color: Colors.black, fontSize: 16)),
            (snapshot.get('Notes') != "")
                ? Text(snapshot.get("Notes"))
                : Container()
          ],
        ),
      ),
    );
```

3.5.2 Διαγραφή αναβάθμισης

Ο ιδιοκτήτης του οχήματος μπορεί να διαγράψει μια αναβάθμιση. Για την διαγραφή μιας αναβάθμισής χρησιμοποιείται η ίδια λογική που υλοποιήθηκε και στη λίστα των οχημάτων, η οποία αναφέρθηκε στην υποενότητα 3.3.4. Ο χρήστης δηλαδή, μπορεί να διαγράψει την καρτέλα της αναβάθμισης είτε επιλέγοντας το σχετικό κουμπί με εικονίδιο τον κάδο είτε σέρνοντας την καρτέλα προς τα αριστερα. Έπειτα εμφανίζεται αναδυόμενο παράθυρο επιβεβαίωσης της διαγραφής, το οποίο επιτυγχάνεται επίσης με την μέθοδο confirm. Εφόσον ο χρήστης επιβεβαιώσει την διαγραφή της αναβάθμισης, καλείται η αντίστοιχη μέθοδος διαγραφής του document με παράμετρο τα στοιχεία που προσδιορίζουν το document, όπως για παράδειγμα το τηλέφωνο του χρήστη στον οποίο ανήκει το όχημα, το όχημα, το είδος της κατηγορίας και το id της αναβάθμισης προς διαγραφή.

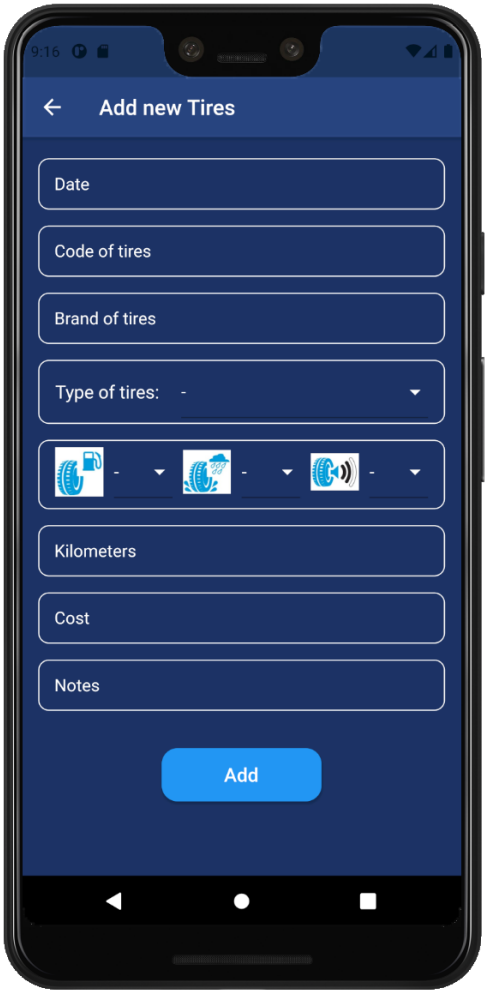
```
void deleteUpdate(String phone, String plate, String category, String id)
{
  try {
    databaseReference
        .collection(phone)
        .doc(plate)
        .collection(category)
        .doc(id)
        .delete();
  } catch (e) {
    print(e.toString());
  }
}
```

Για παράδειγμα, παρακάτω παρατηρούμε το κάλεσμα της μεθόδου διαγραφής αναβάθμισης που έχει πραγματοποιηθεί στην κατηγορία των φρένων.

```
if (action == ConfirmPromt.YES) {
      deleteUpdate(widget.collection, widget.plt, "Pads", snapshot.id);
}
```

3.5.3 Προσθήκη αναβάθμισης

Τέλος, ο χρήστης μπορεί να προσθέσει καινούργια αναβάθμιση επιλέγοντας το κουμπί με το εικονίδιο συν (“+”). Επιλέγοντας το κουμπί προσθήκης αναβάθμισης, ο χρήστης μεταφέρεται σε νέο παράθυρο (Εικόνα 3.16), το οποίο τον προτρέπει να εισάγει τα στοιχεία της αναβάθμισής. Τα στοιχεία της αναβάθμισης περιλαμβάνουν τις βασικές πληροφορίες, όπως για παράδειγμα μάρκα, τιμή κ.α., αλλά έχουν και ένα επιπλέον πεδίο για τυχόν σχόλια/σημειώσεις που επιθυμεί να προσθέσει ο χρήστης.



Εικόνα 3.16: Προσθήκη αναβάθμισης

Είναι υποχρεωτικό ο χρήστης να εισάγει όλες τις πληροφορίες και αν επιθυμεί μπορεί να συμπληρώσει και το πεδίο των σχολίων. Επίσης τα χιλιόμετρα που θα εισάγει πρέπει να είναι μεγαλύτερα από τα

χιλιόμετρα της τελευταίας αναβάθμισης που πραγματοποιήθηκε στην συγκεκριμένη κατηγορία. Με την μέθοδο void getpreviouskm(), η εφαρμογή παίρνει τα προηγούμενα χιλιόμετρα έτσι ώστε να πραγματοποιήσει έλεγχο κατα την εισαγωγή της νέας αναβάθμισης. Ο έλεγχος οτι συμπληρώθηκαν τα υποχρεωτικά πεδία πραγματοποιείται σε κάθε πεδίο ξεχωριστά με τη χρήση του validator. Στο validator ορίζονται όλες οι σχέσεις/περιορισμοί και το μήνυμα λάθους που θα εμφανιστεί στον χρήστη σε κάθε περίπτωση. Αν ο χρηστής συμπληρώσει ορθά το πεδίο, τότε ο validator επιστρέφει null, δηλαδή οτι δεν προέκυψε πρόβλημα κατα την εισαγωγή. Παρακάτω παρατηρούμε τους δυο περιορισμούς που υπάρχουν στο πεδίο των χιλιομέτρων.

```
validator: (val) {
                        var value = val.replaceAll(" ", "");
                        if (value.trim().isEmpty) {
                          return "Enter the km of the car";
                        }
                        if (previousKilometers != null &&
                            double.parse(value.trim()) <=
previousKilometers) {
                          return "Kilometers must be more than " +
                              previousKilometers.toString() +
                              "km";
                        }
                        return null;
                      }),
```

3.5.4 Δημιουργία QR code

Μονάχα ο ιδιοκτήτης του οχήματος μπορεί να δημιουργήσει QR code για το όχημα του. Με την χρήση του QR code επιτρέπεται σε τρίτους να προσπελάσουν το ιστορικό αλλαγών και να προσθέσουν νέες αλλαγές. Με την επιλογή του εικονιδίου QR code στη σελίδα διαχείρισης του οχήματος, δημιουργείται το QR code και εμφανίζεται σε αρχείο pdf. Πιο συγκεκριμένα, ο χρήστης μεταφέρεται σε καινούργιο παράθυρο (Εικόνα 3.17) για το οποίο είναι υπεύθυνη η κλάση QR code. Στο καινούργιο αυτό παράθυρο, εμφανίζεται προεπισκόπηση του αρχείου pdf, το οποίο περιλαμβάνει τον αριθμό κυκλοφορίας του οχήματος, το QR code του αλλά και βασικές πληροφορίες. Ως τίτλος του αρχείου pdf είναι ο αριθμός κυκλοφορίας του οχήματος.

Εικόνα 3.17: Προεπισκόπηση PDF με το QR code

Η πληροφορία που περιλαμβάνει το QR code είναι κωδικοποιημένη με την μέθοδο AES προκειμένου να μην είναι εμφανής η δομή της και να μην είναι δυνατόν να δημιουργηθεί QR code από τρίτους. Το κλειδί κωδικοποίησης των δεδομένων είναι αποθηκευμένο επίσης στην Firestore και είναι μεγέθους 256 bit. Τα δεδομένα που χρειάζονται για την κοινοποίησης του οχήματος είναι ο τηλεφωνικός αριθμός του ιδιοκτήτη και ο αριθμός κυκλοφορίας του οχήματος.

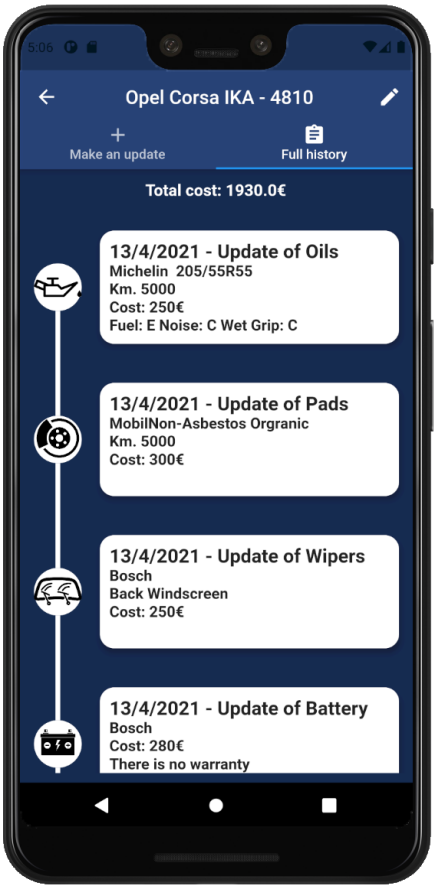
```
final qrdata = gsettings.phone + "|" + plate;
final encrypted = encrypter.encrypt(qrdata, iv: iv);
```

Το επίπεδο διόρθωσης που εφαρμόστηκε στο QR code είναι το H, δηλαδή σφάλματα μικρότερα ή ίσα με το 30% και το μέγεθος του είναι 150 x 150 pixel.

```
pw.BarcodeWidget(
     barcode: pw.Barcode.qrCode(
             errorCorrectLevel:pw.BarcodeQRCorrectionLevel.high),
             width: 150,
             height: 150,
             data: encrypted.base64
    )
)
```

3.5.5 Πλήρες ιστορικό αλλαγών

Ο χρήστης έχει τη δυνατότητα να προσπελάσει το ιστορικό των αλλαγών ανεξαρτήτως κατηγορίας (Εικόνα 3.18). Με την εναλλαγή στο δεύτερο tab “Full History”, εμφανίζεται το συνολικό κόστος των αναβαθμίσεων που έχουν πραγματοποιηθεί στο όχημα από τη στιγμή εγγραφής του αλλά και όλες οι αναβαθμίσεις ταξινομημένες με χρονολογική σειρά. Το παράθυρο είναι scrollable, επομένως όσο ο χρήστης κάνει scroll προς τα κάτω μπορεί και βλέπει τις πιο παλιές αναβαθμίσεις. Κάθε αναβάθμιση έχει την μορφή κάρτας, η οποία έχει ως τίτλο την ημερομηνία που πραγματοποιήθηκε η αναβάθμιση και την κατηγορία στην οποία ανήκει. Επίσης περιλαμβάνει ως κείμενο τις βασικές πληροφορίες της εκάστοτε αναβάθμισης. Τέλος, στα αριστερά από την καρτέλα υπάρχει το εικονίδιο της κατηγορίας προκειμένου να είναι πιο ευανάγνωστο το ιστορικό προς τον χρήστη. Το παράθυρο έχει σχεδιαστεί έτσι ώστε να έχει την μορφή χρονοδιαγράμματος.



Εικόνα 3.18: Χρονοδιάγραμμα αλλαγών

Η μέθοδος getData() μηδενίζει τη μεταβλητή του συνολικού κόστους και τη λίστα των αλλαγών. Έπειτα για κάθε κατηγορία καλεί μια Future<void> μέθοδος, η οποία είναι υπεύθυνη για την προσθήκη των αναβαθμίσεων της κατηγορίας της στην λίστα των συνολικών αναβαθμίσεων αλλά και για την αύξηση του συνολικού κόστους. Έπειτα, η λίστα changes θα περιλαμβάνει όλες τις αναβαθμίσεις που έχουν πραγματοποιηθεί αλλά και ορισμένες βασικές πληροφορίες για αυτές. Προκειμένου να γίνει προβολή των αλλαγών, χρειάστηκε να γίνει χρήση του widget FutureBuilder καθώς η λίστα γεμίζει καλώντας ασύγχρονες μεθόδους. Εως ότου να ολοκληρωθούν οι μέθοδοι και να ληφθούν όλα τα δεδομένα, στο παράθυρο εμφανίζεται ο κυκλικός δείκτης προόδου, ο οποίος αναπαριστά το φόρτωμα/λήψη των δεδομένων.

```
FutureBuilder<List<TimelineModel>>(
      future: getData(),
      builder: (context, snapshot) {
      if (snapshot.hasData) {.....}
      else if (snapshot.hasError) {....}
      return Center(child: CircularProgressIndicator());
}
```

Έπειτα, αν ο αριθμός των δεδομένων είναι μηδενικός, δηλαδή δεν έχουν πραγματοποιηθεί αναβαθμίσεις, εμφανίζεται ανάλογο μήνυμα στο χρήστη. Ενώ αν υπάρχουν αναβαθμίσεις, η προβολή τους πετυχαίνεται με την χρήση του ListView.builder, το οποίο επιτρέπει τον σχεδιασμό λίστα οπου τα widget της έχουν δυναμικό περιεχόμενο. Το πλήθος των widget που θα περιέχει η λίστα ισούται με το πλήθος των δεδομένων της λίστας αναβαθμίσεων που γέμισε νωρίτερα με το FutureBuilder.

```
ListView.builder(
    itemCount: snapshot.data.length,
    itemBuilder: (context, index) {
        return Container(
            child: Padding(
            padding: const EdgeInsets.only(right: 10),
            child: //sxediasmos cartas
))})
```

Τέλος, για το σχεδιασμό της εκάστοτε κάρτας αναβάθμισης, χρησιμοποιήθηκε το widget Row το οποίο στο αριστερό μέλος περιλαμβάνει το αντίστοιχο εικονίδιο ενώ στο δεξί την κάρτα αναβάθμισης.

```
Row(
    children: [
        //ekkinisi sxediasmoy eikonidiou
          Column(
            children: [
                 Container(
                      width: 5,height: 55,
                      color: (index == 0) ? null : Colors.white,
```

```
                                    ),
                Container(
                     width: 50,height: 50,
                     margin:EdgeInsets.only(left: 15, right: 15),
                     decoration: BoxDecoration(
                           color: Colors.white,
                           borderRadius:BorderRadius.circular(50)),
                           child: selectImage(
snapshot.data[index].typeofchange),
                           ),
                Container(
                     width: 5,height: 55,
                     color: (index == snapshot.data.length - 1)
                             ? null: Colors.white),
              ],),
              //ekkinisi sxediasmou kartas
            Expanded(
                child: Card(elevation: 4,
                       shape: RoundedRectangleBorder(
                       borderRadius: BorderRadius.circular(15.0),),
                       child: Padding(
                           padding: EdgeInsets.all(10),
                           child: Column(
                           CrossAxisAlignment:CrossAxisAlignment.start,
                            children: <Widget>[
                            //plirofories sxetika me tin anabathmisi
                            //se widget Text()
                        ],
                    ),
                ),
            ),
        ),
    ],
),
```

3.6 Επίλογος

Η εφαρμογή Service Book έχει ως κύριο στόχο την ψηφιοποίηση του βιβλιαρίου αναβαθμίσεων του αυτοκινήτου προκειμένου να είναι προσβασιμότητα από τον χρήστη οποιαδήποτε στιγμή και να εξασφαλίζεται η διαθεσιμότητα των δεδομένων. Με αυτόν τον τρόπο, επιτυγχάνεται η προσαρμογή στη καθημερινότητα και ο εκσυγχρονισμός της καταγραφής των δεδομένων σχετικά με τη διαδικασία συντήρησης του οχήματος. Το μεγάλο πλεονέκτημα της εν λόγω εφαρμογής, είναι το γεγονός ότι είναι φιλική προς τον χρήστη και ως εκ τούτου απευθύνεται και σε χρήστες που δεν είναι ιδιαίτερα

εξοικειωμένοι με τα μέσα τεχνολογίας. Χαρακτηριστικό παράδειγμα αποτελεί η λειτουργία εισόδου, η οποία πραγματοποιείται με τη χρήση μονάχα του τηλεφωνικού αριθμού του χρήστη, με αποτέλεσμα να απλοποιείται η διαδικασία και ο χρήστης να μην χρειάζεται να επαληθεύει τα στοιχεία του ή να θυμάται τα συνθηματικά του. Τέλος, η εφαρμογή δίνει την δυνατότητα κοινοποίησης των αναβαθμίσεων που έχουν πραγματοποιηθεί σε ένα όχημα, επιτρέποντας έτσι την προσπέλαση του πλήρες ιστορικού από τρίτα πρόσωπα, όπως για παράδειγμα τα συνεργεία αυτοκινήτων. Ο κώδικας της εφαρμογής, ο οποίος είναι γραμμένος σε Dart, είναι εύκολα επεκτάσιμος και διορθώσιμος δεδομένου ότι δεν είναι περίπλοκος και έγινε χρήση διαφορετικών μεθόδων για κάθε λειτουργία έτσι ώστε να αποτραπεί το μεγάλο μέγεθος κώδικα ανα κλάση.

Κεφάλαιο 4ο: Συμπεράσματα ή/και προτάσεις βελτίωσης

Η ανάπτυξη της συγκεκριμένης εφαρμογής είχε ως αποτέλεσμα να εξοικειωθώ και να αποκτήσω περισσότερη εμπειρία με τη Flutter, τη Dart αλλά και τις υπηρεσίες που προσφέρονται από τη Firebase. Όσον αφορά τον προγραμματισμό εφαρμογών για κινητές συσκευές με τη Flutter, είναι αρκετά εύκολος καθώς η Flutter προσφέρει στον προγραμματιστή πλήθος λειτουργιών οι οποίες στοχεύουν στη γρηγορότερη και ευκολότερη ανάπτυξη του κώδικα. Επίσης δίνεται αναλυτικό documentation απο την Google και παρόλο που πρόκειται για πρόσφατη τεχνολογία υπάρχει αρκετά μεγάλο community.

Για τον προγραμματισμό της εφαρμογής χρησιμοποιήθηκε η γλώσσα Dart και ο κώδικας είναι καθαρογραμμένος, χωρίς σφάλματα και έχει γίνε χρήση custom widget προκειμένου να μην είναι περίπλοκος και να είναι εύκολα αναγνώσιμος από τρίτους προγραμματιστές. Επιπλέον, ο κώδικας είναι εύκολα επεκτάσιμος.

Όσον αφορά την ασφάλεια των δεδομένων, τα δεδομένα είναι αποθηκευμένα σε online βάση δεδομένων προκειμένου να μην απειλείται η διαθεσιμότητα τους και ο χρήστης να μπορεί να τα προσπελάσει ασχέτως τη συσκευή που χρησιμοποιεί. Επίσης, παρέχεται αυθεντικοποίηση του χρήστη κατα την είσοδο του στην εφαρμογή έτσι ώστε να μπορούν να δοθούν δικαιώματα ανάλογα με τους κανόνες ασφαλείας που ακολουθούνται αλλά και να γίνεται καταγραφή του ατόμου που πραγματοποιεί αλλαγές. Σχετικά με την κοινοποίηση οχήματος, οι υπόλοιποι χρήστες δεν έχουν τα ίδια δικαιώματα με τον ιδιοκτήτη του οχήματος έτσι ώστε να μην υπάρχει κίνδυνος αλλοίωσης του ιστορικού των αλλαγών. Τέλος, εφαρμόστηκε κωδικοποίηση AES των δεδομένων που περιλαμβάνει το QR code με κλειδί μεγέθους 256bit προκειμένου να μην είναι ευδιάκριτη η δομή του και να μην μπορούν να δημιουργηθούν QR code για οχήματα εκτός της εφαρμογής.

Η εφαρμογή είναι συνδεδεμένη με την υπηρεσία προώθησης μηνυμάτων της Firebase και αποθηκεύει στην βάση δεδομένων όλες τις απαραίτητες πληροφορίες για την αποστολή και λήψη ειδοποιήσεων, όπως για παράδειγμα το αναγνωριστικό token της εκάστοτε συσκευής. Μια χρήσιμη αλλά ανεξάρτητη από την εφαρμογή βελτίωση, θα ήταν η δημιουργία προγράμματος το οποία θα εκτελείται μόνιμα και θα πραγματοποιεί περιοδικούς ελέγχους στις ημερομηνίες των αλλαγών ώστε να ενημερώνει το χρήστη για τυχόν αλλαγές που εκκρεμούν. Όλα τα απαραίτητα δεδομένα για την πραγματοποίηση ελέγχων βρίσκονται στη Firebase. Τέτοια δεδομένα είναι οι αλλαγές που πραγματοποιήθηκαν σε συνδυασμό με το timestamp τους και τα χιλιόμετρα κατα τα οποία πραγματοποιήθηκαν. Επίσης είναι αποθηκευμένα τα μοναδικά αναγνωριστικά κάθε συσκευής (token) έτσι ώστε να προωθηθεί ανάλογη ειδοποίηση προς τον αντίστοιχο ιδιοκτήτη οχήματος, ο οποίος χρειάζεται να ενημερωθεί για την αλλαγή που εκκρεμεί.

Η εφαρμογή μπορεί να εκτελεστεί σε συσκευές με λειτουργικό σύστημα Android 7 και πάνω και ο χρήστης θα μπορεί να την κατεβάσει απο το Play Store, προς το παρόν δεν είναι διαθέσιμη. Η εφαρμογή πέτυχε τον στόχο της καθώς αποτελεί ένα εύχρηστο εργαλείο, το οποίο βοηθάει στη ψηφιοποίηση του βιβλιαρίου αναβαθμίσεων του αυτοκινήτου και στον εκσυγχρονισμό των δεδομένων σχετικά με τη διαδικασία συντήρησης του οχήματος. Τέλος, είμαι ικανοποιημένος και ευχαριστημένος απο τις καινούργιες γνώσεις και εμπειρίες που απέκτησα μέσω της διπλωματικής εργασίας.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Internet Site

[2] Google Firebase, Available: https://firebase.google.com/

[3] Flutter Showcase, Available: https://flutter.dev/showcase

[4] Desktop vs Mobile vs Tablet vs Console Market Share Worldwide. Available: https://gs.statcounter.com/platform-market-share#monthly-202001-202101https://gs.statcounter.com/platform-market-share#monthly-202001-202101

[5] Android, Available: https://www.android.com/

[6] iOS, Available: https://www.apple.com/ios/ios-14/

[8] App Revenue Data of 2021. Available: https://www.businessofapps.com/data/app-revenues/

[9]Number of apps available in leading app stores as of 4th quarter 2020. Available: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[12] Flutter Documentation, Available: https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html

[13] Dart Overview - Documentation, Available: https://dart.dev/overview

[15] Sans 'Glossary of Security Terms', Available: https://www.sans.org/security-resources/glossary-of-terms

[19] Introduction to Push Notification - Documentation Available: https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications

Paper in Conference Proceedings

[1] Jakhongir Fayzullaev, "Native-like Cross-Platform Mobile Development Multi-OS Engine & Kotlin Native vs Flutter", South-Eastern Finland University of Applied Sciences 2018. Available: https://www.theseus.fi/bitstream/handle/10024/148975/thesis_Jakhongir_Fayzullaev.pdf

[7] Awel Eshetu Fentaw, "Cross platform mobile application development: a comparison study

of React Native Vs Flutter", University of Jyvaskyla Faculty of Information Technology 2020. Available:https://jyx.jyu.fi/bitstream/handle/123456789/70969/1/URN%3ANBN%3Afi%3Ajyu-202006295155.pdf

[10] Wenhao Wu, "React Native vs Flutter, cross-platform mobile application frameworks", Metropolia University of Applied Sciences 2018. Available: https://www.theseus.fi/bitstream/handle/10024/146232/thesis.pdf

[11] Jakhongir Fayzullaev, "Native-like Cross-Platform Mobile Development Multi-OS Engine & Kotlin Native vs Flutter", South-Eastern Finland University of Applied Sciences 2018. Available: https://www.theseus.fi/bitstream/handle/10024/148975/thesis_Jakhongir_Fayzullaev.pdf

[14] Nilanjan Chatterjee, Souvik Chakraborty, Aakash Decosta and Dr. Asoke Nath, "Real-time Communication Application Based on Android Using Google Firebase" 4 April 2018. Available:

https://www.researchgate.net/profile/Asoke-Nath-4/publication/324840628_Real-time_Communication_Application_Based_on_Android_Using_Google_Firebase/links/5ae721760f7e9b9793c82cbf/Real-time-Communication-Application-Based-on-Android-Using-Google-Firebase.pdf

[20] Miroslav Mikolaj, "Using Flutter framework in multi-platform application implementation", Masaryk University Faculty of Informatics 2020. Available: https://is.muni.cz/th/on45r/bachelors_thesis.pdf

Journal Articles

[16] Wu-Jeng Li, Chiaming Yen, You-Sheng Lin, Shu-Chu Tung and ShihMiao Huang, "JustIoT Internet of Things based on the Firebase Real-time Database" IProceedings of the 2018 International Symposium on Semiconductor Manufacturing Intelligence (ISMI2018) Available: https://www.researchgate.net/profile/Wu-Jeng-Li/publication/323342152_JustIoT_Internet_of_Things_based_on_the_Firebase_Real-time_Database/links/5bceb416a6fdcc204a013325/JustIoT-Internet-of-Things-based-on-the-Firebase-Real-time-Database.pdf

[17] Dr. S. George, "NOSQL - NOT ONLY SQL", nternational Journal of Enterprise Computing and Business Systems, International Manuscript ID : ISSN22308849-V2I2M3-072013 Available: http://www.ijecbs.com/July2013/3.pdf

[18] Chunnu Khawas and Pritam Shah, "Application of Firebase in Android App Development-A Study" International Journal of Computer Applications (0975 – 8887) Volume 179 – No.46, June 2018 Available: https://www.researchgate.net/profile/Chunnu-Khawas/publication/325791990_Application_of_Firebase_in_Android_App_Development-A_Study/links/5bab55ed45851574f7e6801e/Application-of-Firebase-in-Android-App-Development-A-Study.pdf

[21] Ghusoon Idan Arb and Kadhum Al-Majdi, "A Freights Status Management System Based on Dart and Flutter Programming Language", Journal of Physics: Conference Series 2020. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1530/1/012020/pdf

[22] Sai Spandhana Reddy Emmadi and Sirisha Potluri, "Android Based Instant Messaging Application Using Firebase" International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-7 Issue-5S2, January 2019 Available: https://www.researchgate.net/profile/Sirisha-Potluri/publication/332264701_Android_based_instant_messaging_application_using_firebase/links/5cf5ef204585153c3db19ff3/Android-based-instant-messaging-application-using-firebase.pdf

Βιβλία

[23] Laurence Moroney, *The Definitive Guide to Firebase*. New York: Springer Science & Business Media, 2017.