ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ MOBILE ΕΦΑΡΜΟΓΗΣ ΜΕ ΤΗΝ ΧΡΗΣΗ FLUTTER

Του φοιτητή

Πελτέκη Παναγιώτη

Αρ. Μητρώου 164730
Ημερομηνία 17-09-2020

Επιβλέπων

Ευκλείδης Κεραμόπουλος

Αναπληρωτής Καθηγητής

Τίτλος Δ.Ε: Ανάπτυξη Mobile Εφαρμογής με την χρήση Flutter

Κωδικός Δ.Ε.: 20155

Ονοματεπώνυμο φοιτητή/τών: Πελτέκης Παναγιώτης

Ονοματεπώνυμο εισηγητή: Ευκλείδης Κεραμόπουλος

Ημερομηνία ανάληψης Δ.Ε.: 06-04-2020

Ημερομηνία Περάτωσης Δ.Ε.: 06-07-2021

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 εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε

Πρόλογος

Το θέμα της διπλωματικής αυτής αποτελεί ένα νέο τομέα στο κομμάτι της ανάπτυξης εφαρμογών για Mobile συσκευές. Το Flutter αποτελεί μια τεχνολογία η οποία υπόσχεται την ανάπτυξη λογισμικού με μία κοινή βάση κώδικα τόσο στην πλατφόρμα του Android όσο και σε αυτήν του iOS. Επίσης μέσω του Flutter έγινε η απόπειρα να μειωθεί το κενό ανάμεσα στο σχεδιασμό της διεπαφής του χρήστη και στην υλοποίηση της καθώς από την αρχή της συνύπαρξης των δύο, οι απαιτήσεις των σχεδιαστών ποτέ δεν μπορούσαν να ικανοποιηθούν. Αυτή η εργασία λοιπόν θα κάνει μια προσομοίωση της υλοποίησης εφαρμογής με Flutter ώστε να βγούν τα κατάλληλα συμπεράσματα.

Περίληψη

Το θέμα της διπλωματικής αυτής αποτελεί ένα νέο τομέα στο κομμάτι της ανάπτυξης εφαρμογών για Mobile συσκευές. Το Flutter αποτελεί μια τεχνολογία της Google η οποία υπόσχεται την ανάπτυξη λογισμικού για κινητές συσκευές με μία κοινή βάση κώδικα τόσο στην πλατφόρμα του Android όσο και σε αυτήν του iOS. Χρησιμοποιώντας αυτό το συνόλο εργαλείων κάθε προγραμματιστής μπορεί να έχει την δυνατότητα να αποκτήσει τον πλήρη έλεγχο στην ανάπτυξη της εφαρμογής από όλες τις δυνατές πλευρές και να γράψει κώδικα ο οποίος έχει ως κέντρο έναν τομέα ο οποίος ίσως θα μπορούσε να θεωρηθεί και ένα νευραλγικό σημείο για την επιτυχίας μιας εφαρμογής. Την διεπαφή του χρήστη. Το Flutter αποτελεί την απόπειρα να μειωθεί το κενό ανάμεσα στο σχεδιασμό της διεπαφής του χρήστη και στην υλοποίηση του υπόλοιπου κομματιού της εφαρμογής, το οποίο περιλαμβάνει το σύστημα αποθήκευσης δεδομένων, την όλη υλοποίηση του πηγαίου κώδικα σχετικά με τα μοντέλα των δεδομένων της εφαρμογής, την αρχιτεκτονική της εφαρμογής και πολλά άλλα στα οποία θα αναφερθούμε και στην συνέχεια. Έχοντας λοιπόν ένα συγκεκριμένο σενάριο για την ανάπτυξης μιας εφαρμογής, η εργασία αυτή θα κάνει μια προσομοίωση της υλοποίησης με Flutter και καθ’όλη την πορεία θα βγούν τα κατάλληλα συμπεράσματα για το αποτέλεσμα, τι ωφέλησε και τι όχι και μια γενική ανάλυση της κάθε πτυχής του Project.

Development of a Mobile Sports App using the Flutter Framework

Panagiotis

Peltekis

Abstract

This thesis subject is about the development of a Mobile App using the Flutter Framework. The project will revolve around the initial concept of a new app mainly focusing on the design of the User Interface and the implementation of the app itself and then there will be discussion about the results of the Flutter Development.

Περιεχόμενα

Κατάλογος Σχημάτων

Συντομογραφίες

UI	User Interface

Κεφάλαιο 1ο: Εισαγωγή

Ένα βασικό θέμα που αποτελεί κομμάτι του κόσμου του προγραμματισμού είναι η ύπαρξη ενός μεγάλου αριθμού γλωσσών προγραμματισμού καθώς και συστημάτων τα οποία μπορούν είτε να εξυπηρετούν διαφορετικούς σκοπούς, είτε απλά να αποτελούν μια διαφορετική υλοποίηση για την λύση του ίδιου προβλήματος. Κάτι τέτοιο θα μπορούσε να θεωρηθεί ως μια υγιής ποικιλία εργαλείων τα οποία έχει ένας προγραμματιστής στην διάθεση του, ωστόσο με το πέρας του καιρού, κανείς μπορεί πολύ εύκολα να συμπεράνει ότι αυτό έχει φτάσει τον μέσο προγραμματιστή στο χείλος της υπερπληροφόρησης. Κάτι τέτοιο αποτελεί ένα τεράστιο εμπόδιο στην ανάπτυξη καθώς οι γνώσεις που χρειάζονται κάθε χρόνο αυξάνονται εκθετικά. Ωστόσο αυτήν την περίοδο αρχίζουν να αναδύονται λύσεις οι οποίες προσφέρουν μια ενοποιημένη διαχείριση της διαδικασίας ανάπτυξης λογισμικού. Η παρούσα πτυχιακή αποτελεί ένα παράδειγμα ανάπτυξης mobile εφαρμογής για τις δύο βασικές πλατφόρμες που αποτελούν κομμάτι της σημερινής αγοράς: Android και iOS. Μέσω αυτής της εργασίας θα γίνει μια επίδειξη της ανάπτυξης εφαρμογών για παραπάνω από μια πλατφόρμες με την χρήση του σετ των εργαλείων που προσφέρονται από την Google ονόματι Flutter.

Συγκεκριμένα, στο πρώτο κεφάλαιο θα γίνει μια γενική ανάλυση του τι ακριβώς είναι το Flutter, πως ακριβώς η υπάρξη του μπορεί να εξυπηρετήσει τον μέσο προγραμματιστή για την ανάπτυξη mobile εφαρμογών και ποια είναι τα βασικά χαρακτηριστικά της. Στο 2ο κεφάλαιο θα γίνει η περιγραφή του σεναρίου που επιλέχθηκε για να μπορέσει να γίνει η ανάπτυξη της εφαρμογής, με μια γενική αναφορά των απαιτήσεων του συγκεκριμένου σεναρίου. Στην συνέχεια, θα γίνει μια εκτεταμένη ανάλυση της έννοιας του User Interface Design ή αλλιώς Σχεδιασμός Διεπαφής Χρήστη. Σε αυτό το κεφάλαιο θα γίνει μια καταγραφή του τι ακριβώς ορίζεται ως User Interface Design καθώς και των βημάτων που λαμβάνουν χώρα. Έπειτα θα πάμε στο πιο σημαντικό κομμάτι της πτυχιακής που είναι η ανάπτυξη της εφαρμογής. Σε αυτό το κομμάτι θα μιλήσουμε για όλα τα κομμάτια που απαρτίζουν την υλοποίηση μιας εφαρμογής σε Flutter, ξεκινώντας από τα αρχεία του Project και καταλήγοντας στον πηγαίο κώδικα που θα χρησιμοποιηθεί στις εφαρμογές. Επίσης θα γίνει ανάλυση και του Firebase το οποίο ουσιαστικά είναι ένα σύστημα της Google για αποθήκευση δεδομένων που χρησιμοποιήθηκε ως κομμάτι της παρούσας εφαρμογής. Κλείνοντας στο τελευταίο κεφάλαιο, θα γίνει μια αναφορά των συμπερασμάτων που δημιουργήθηκαν από την διαδιακασία ανάπτυξης της Mobile εφαρμογής.

Οι Mobile εφαρμογές αποτελούν ένα αδιαμφισβήτητο κομμάτι της σημερινής αγοράς και το ποσοστό το οποίο καταλαμβάνουν συνεχώς αυξάνεται. Νέες τεχνολογίες αναδύονται καθημερινά οι οποίες έχουν ως στόχο την ανάπτυξη εφαρμογών με μεθοδολογίες οι οποίες επιτρέπουν στους υπεύθυνους της ανάπτυξης των εφαρμογών να επιτελούν αυτό το έργο μέσα από όσο το δυνατόν πιο εύκολες και εύχρηστες διαδικασίες χωρίς πολλά έξοδα και σπατάλη χρόνου. Μέσα σε αυτόν τον αγώνα ένας από τους συναγωνιστές σήμερα αποτελεί και το Flutter. Πολλές εταιρίες σήμερα χρησιμοποιούν το παρόν SDK για την ανάπτυξη των εφαρμογών τους και με τον καιρό το Flutter αρχίζει να εδραιώνεται στον χώρο. Το βασικό ερώτημα όμως είναι: Τι είναι το Flutter και πως ακριβώς λειτουργεί?

Το Flutter [1] είναι ένα mobile SDK, κατασκευασμένο από την Google [1] και είναι Open Source [10]. Το βασικό χαρακτηριστικό που κάνει το Flutter μοναδικό είναι ότι με μία βάση κώδικα γίνεται ανάπτυξη μιας εφαρμογής σε πολλαπλές πλατφόρμες. Εν συντομία το Flutter είναι ένα ολοκληρωμένο

SDK το οποίο παρέχει ότι χρειάζεται ένας σύγχρονος δημιουργός εφαρμογών, όπως Rendering Engine [11], Frameworks και εργαλεία για την ανάπτυξη και έλεγχο μιας εφαρμογής και πολλά άλλα.

Οι σύγχρονες εφαρμογές Flutter είναι γραμμένες στην γλώσσα προγραμματισμού Dart. Η Dart είναι ιδιοκτησία της Google και συντηρείτε επίσης από αυτήν. Αν και δεν αποτελεί μια από τις ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού σήμερα, οι λόγοι ύπαρξης και χρήσης της υπάρχουν και είναι οι εξής:

1) Η Dart υποστηρίζει just-in-time (JIT) compiling [1] και ahead-of-time (AOT) [1] compiling.
Αναλυτικότερα:

- Ο AOT μεταγλωττιστής αλλάζει την Dart σε native κώδικα. Αυτό σημαίνει ότι όλα γίνονται αυτόματα και γρήγορα οπότε ο Developer δεν χρειάζεται να ασχοληθεί με τις λεπτομέρειες και μπορεί να τροποποιήσει σχεδόν τα πάντα. [1]

- Η προαιρετική JIT μεταγλώττιση της Dart επιτρέπει στον χρήστη την χρήση της λειτουργίας Hot Reload.

- Η Dart είναι μια αντικειμενοστρεφής γλώσσα προγραμματισμού. Αυτό την κάνει εύκολη στην συγγραφή και ανάπτυξη οπτικών εμπειριών, χωρίς την ανάγκη χρήσης γλώσσας σήμανσης

Το Flutter μεταγλωτίζεται κατευθείαν σε ARM κώδικα [1]. Επίσης το Flutter περιλαμβάνει την δικιά του Rendering Engine. Η έννοια των Rendering Engines είναι εκτός του πεδίου αυτής της μελέτης παρ'όλα αυτά είναι ένα απαραίτητο σημείο το οποίο πρέπει να αναγνωρίσουμε πως αποτελεί κομμάτι της όλης νοοτροπίας και τρόπου λειτουργίας του Flutter καθώς τα δύο σημεία που αναφέρθηκαν προηγουμένως σε συνδυασμό μας οδηγούν στο συμπέρασμα ότι κάθε εφαρμογή λειτουργεί Natively, δηλαδή επικοινωνεί απευθείας με Native γεγονότα και ελέγχει κάθε Pixel της οθόνης ανεξάρτητα.

Σε ένα υψηλό επίπεδο, το Flutter είναι μια reactive (αντιδραστική), δηλωτική και συνθετική βιβλιοθήκη. Το Flutter έχει ως βάση τα Widgets, που αποτελούν και το βασικό χαρακτηριστικό αυτής της βιβλιοθήκης. Τα Widgets περιγράφουν το πως η όψη μιας εφαρμογής θα έπρεπε να μοιάζει για τις δεδομένες ρυθμίσεις και την δεδομένη κατάσταση ή αλλιώς State. Όταν το State ενός Widget αλλάξει, το Widget κάνει Rebuild του εαυτού του, δηλαδή ξαναχτίζεται βάσει της περιγραφής του.

Η εισαγωγική αυτή ανάλυση εν κατακλείδι μας δείχνει πως το Flutter προσφέρει μια ποικιλία από λειτουργίες οι οποίες συνεισφέρουν σε μεγάλο βαθμό στην ευκολία ανάπτυξης κώδικα για τις εφαρμογές και αποτελεί έναν νέο τρόπο ανάπτυξης αυτών των εφαρμογών με την χρήση νέων εργαλείων. Στην σημερινή αγορά όμως πέραν του προγραμματισμού κομμάτι ανάπτυξης εφαρμογών είναι και ο σχεδιασμός των διεπαφών χρήστη.

Κεφάλαιο 2ο: User Interface / User Experience Design

2.1 Εισαγωγή

Κομμάτι της υλοποίησης μιας εφαρμογής αποτελεί και ο κατάλληλος σχεδιασμός και υλοποίηση της διεπαφής του χρήστη. Στον κομμάτι του σχεδιασμού της διεπαφής δύο είναι τα βασικά σημεία που κάθε User Interface Designer θα πρέπει να λαμβάνει υπόψη του. Το πρώτο σημείο είναι η έρευνα. Το συγκεκριμένο σημείο απαρτίζεται από δύο υποκατηγορίες οι οποίες είναι η έρευνα στο κομμάτι των χρηστών (User Research) της οποίας ο στόχος είναι να ορίσει ποιος είναι ο χρήστης της εφαρμογής, ποια είναι τα χαρακτηριστικά που τον καθορίζουν και πως θα χρησιμοποιήσει την εφαρμογή και την έρευνα στο κομμάτι της αγοράς. Ένας τρόπος για να μπορέσει μια εφαρμογή να υλοποιηθεί με μεγαλύτερη ευκολία και ταχύτητα είναι η έρευνα και αξιολόγηση άλλων Project τα οποία έχουν δημοσιευτεί και αποτελούν κομμάτι της σημερινής αγοράς. Το δεύτερο σημείο είναι ο σχεδιασμός και το Prototyping τα οποία απαρτίζονται από τον σχεδιασμό βασικών σκίτσων του πως θα είναι η κάθε οθόνη της εφαρμογής ως προς την ολότητα της και στην συνέχεια την ολοκλήρωση τους σε κανονικά πρωτότυπα διεπαφών χρήστη. Έχοντας λοιπόν υπόψη τα παραπάνω σημεία η υλοποίηση του User Interface ξεκίνησε με την έρευνα σχετικά με το προφίλ του χρήστη. Με βάση το Concept το οποίο έχει οριστεί για την υλοποίηση της εφαρμογής, έγινε ο σχεδιασμός τόσο του User Interface όσο και του User Experience.

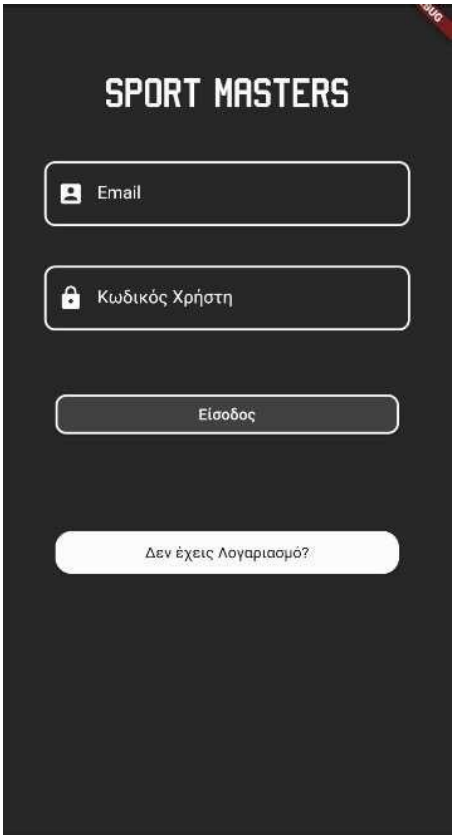
2.2 User Research

Το Project με βάση τα δεδομένα που έχουμε έχει ως μέσο χρήστη έναν άνθρωπο ο οποίος αποτελεί και τον μέσο οπαδό αθλητικών δρώμενων. Αυτό σημαίνει ότι παρακολουθεί σε καθημερινή βάση την αθλητική επικαιρότητα και το διαδικτυακό περιεχόμενο της ομάδας του Sport Masters. Αυτό συνεπάγεται στην περίπτωση μας με τα διαδικτυακά Podcasts τα οποία είναι ανεβασμένα στις πλατφόρμα κοινωνικής δικτύωσης Youtube. Έτσι λοιπόν θα μεταφερθούμε στον επόμενο βήμα το οποίο είναι ο σχεδιασμός των διεπαφών έχοντας ως βασικό άξονα τα δεδομένα ή αλλιώς προφίλ του μέσου χρήστη.

2.3 Wireframes

Έχοντας λοιπόν ορίσει ποιος θα είναι ο μέσος χρήστης και έχοντας κάποια δεδομένα συνεχίζουμε με την δημιουργία των Wireframes. Τα Wireframes ουσιαστικά αποτελούν τον τελικό σχεδιασμό της δομής μιας διεπαφής χρήστη. Στην συγκεκριμένη εφαρμογή οι οθόνες που θέλουμε είναι οι εξής:

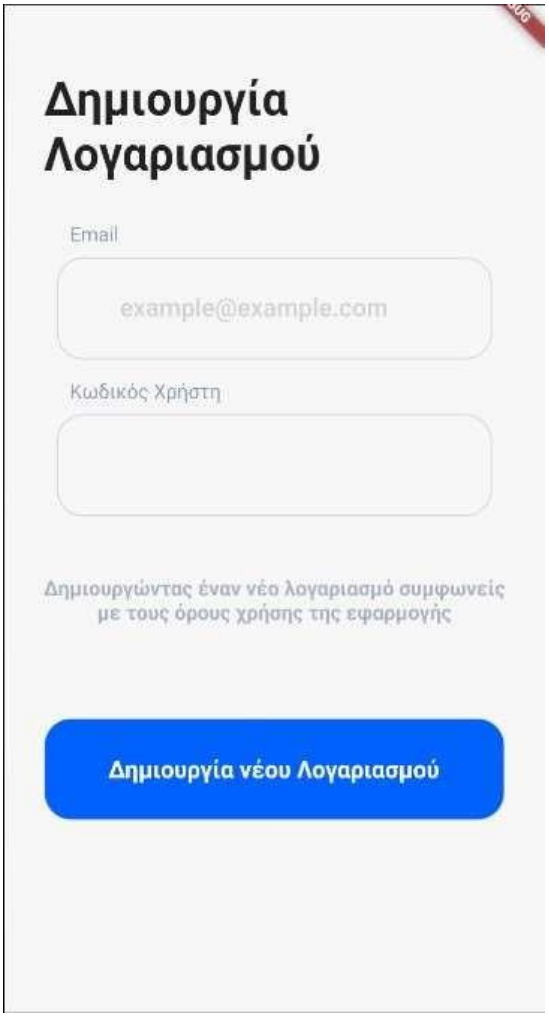
- Είσοδος:



Εικόνα 1: Mockup Οθόνης Login 1

Η Εικόνα 1 αποτελεί την οθόνη του Login. Η οθόνη αποτελείται από μια κλασσική διάταξη μιας οθόνης εφαρμογής για είσοδο στο κυρίως περιεχόμενο. Τα βασικά τμήματα αυτής της οθόνης είναι αρχικά τα πεδία email και κωδικός χρήστη. Τα δύο πεδία χρησιμοποιούνται από τον χρήστη ώστε να μπορέσει να γίνει η είσοδος των στοιχείων του. Το πεδίο του κωδικού χρήστη έχει το χαρακτηριστικό μετατροπής των χαρακτήρων που εισάγονται από τον χρήστη σε τελείες για την απόκρυψη της εισόδου με σκοπό την ασφάλεια των στοιχείων του χρήστη. Έπειτα υπάρχουν τα κουμπιά εισόδου και δημιουργίας λογαριασμού. Το κουμπί εισόδου είναι υπεύθυνο να συλλέξει τα στοιχεία του χρήστη και να επικοινωνήσει με το σύστημα αυθεντικοποίησης του Firebase ώστε να ταυτοποιηθούν τα στοιχεία και να γίνει είτε η είσοδος στην εφαρμογή είτε η εμφάνιση μηνύματος ότι τα στοιχεία που πληκτρολογήθηκαν είναι λάθος.

- Εγγραφή:



Εικόνα 2: Mockup Οθόνης Registration 1

Στην Εικόνα 2 φαίνεται το mockup της οθόνης εγγραφής. Η οθόνη της εγγραφής αποτελείται από δύο πεδία και ένα κουμπί. Στα δύο πεδία εισόδου, ο χρήστης θα πρέπει να πληκτρολογήσει τα στοιχεία του για να μπορέσει να γίνει η εγγραφή του. Το πρώτο πεδίο θα πρέπει να περιλαμβάνει το email του χρήστη, ενώ το δεύτερο έναν κωδικό ο οποίος θα χρησιμοποιείται από τον ίδιο για την ταυτοποίηση του. Το πεδίο του Email καθώς και του κωδικού πρόσβασης θα έχουν έναν έλεγχο εισόδου, όπου αυτό το οποίο θα πληκτρολογήσει ο χρήστης θα πρέπει να τηρεί κάποιους συγκεκριμένους κανόνες ώστε να μπορέσει να χρησιμοποιηθεί για την εγγραφή του χρήστη στο σύστημα. Για παράδειγμα, ο κωδικός θα πρέπει να περιέχει ένα κεφαλαίο γράμμα, έναν αριθμό και έναν ειδικό χαρακτήρα. Στην περίπτωση που αυτό που ο χρήστης πληκτρολογήσει δεν είναι σωστό τότε στο κάτω μέρος του πεδίου θα εμφανιστεί ένα μήνυμα όπου θα του δείχνει τα κριτήρια αυτά με την μορφή κειμένου. Το κουμπί αποτελεί το τελικό κομμάτι του flow της οθόνης όπου μέσω αυτού ο χρήστης θα ενεργοποιήσει την όλη διαδικασία ελέγχου και αν όντως όλα είναι σωστά γίνεται η επικοινωνία με το Firebase σύστημα έτσι ώστε να γίνει η δημιουργία του νέου χρήστη. Όταν ολοκληρωθεί, τότε η εφαρμογή μεταφέρεται στο κυρίως περιεχόμενο.

- Αρχική Οθόνη:

Εικόνα 3: Mockup οθόνης HomePage 1

Στην Εικόνα 3 βλέπουμε την αρχική σελίδα. Η αρχική σελίδα αποτελεί την πρώτη οθόνη μετά την είσοδο των στοιχείων του χρήστη, είτε αν ο χρήστης είναι ήδη εγγεγραμμένος είτε αν κάνει λογαριασμό για πρώτη φορά. Σε αυτήν την οθόνη ο χρήστης θα μπορεί να δεί τα πιο πρόσφατα άρθρα και να περιηγηθεί σε αυτά. Η διάταξη της οθόνης ξεκινάει αρχικά με την μπάρα που βρίσκεται στο πάνω μέρος και αποτελεί ένα γενικό κομμάτι το οποίο εφαρμόζεται σε παραπάνω από μια οθόνες. Στην μπάρα περιλαμβάνονται δύο στοιχεία. Το πρώτο είναι ο τίτλος ο οποίος βρίσκεται κεντραρισμένος και έπειτα ένα κουμπί στο αριστερό μέρος το οποίο όταν πατηθεί ανοίγει ένα επιπλέον μενού από τα αριστερά. Στο κάτω μέρος, υπάρχει μια ακόμα μπάρα η οποία επίσης αποτελεί ένα γενικό κομμάτι της εφαρμογής. Η μπάρα αυτή αποτελείται από τρία κουμπιά, κάθε ένα από τα οποία πατώντας τα, αλλάζει και το περιεχόμενο της οθόνης με το αντίστοιχο περιεχόμενο με το οποίο είναι συνδεδεμένο κάθε κουμπί. Στην περίπτωση μας, τα κουμπιά αντιστοιχούν στην αρχική οθόνη, την οθόνη των νέων και την οθόνη των podcasts αντίστοιχα. Για να μπορέσει να υπάρξει μια ανάδραση και ο χρήστης να γνωρίζει σε ποια οθόνη βρίσκεται τα κουμπιά της μπάρας έχουν την δυνατότητα αλλαγής χρώματος όταν ο χρήστης πάνω σε ένα από αυτά. Για παράδειγμα, αν ο χρήστης βρίσκεται στην αρχική σελίδα και θελήσει να μεταφερθεί στην οθόνη 'Νέα' θα πατήσει το μεσαίο κουμπί. Αυτό λοιπόν πέραν του ότι θα αλλάξει και όλο το περιεχόμενο της οθόνης θα αλλάξει τα χρώματα του κουμπιού της οθόνης που βρίσκεται ήδη ο χρήστης σε λευκό και το χρώμα του κουμπιού που πάτησε σε μπλε. Έτσι θα μπορεί να γνωρίζει ποια είναι η τρέχουσα οθόνη που καταλαμβάνει την οθόνη του. Στο κυρίως περιεχόμενο της αρχικής σελίδας περιλαμβάνονται δύο βασικά τμήματα, εκ των οποίων το ένα είναι ένα καρουσελ από άρθρα το οποίο ο χρήστης μπορεί να κυλήσει για να μπορέσει να δεί όλα τα άρθρα. Επίσης το ίδιο το καρουσελ γυρνάει αυτόματα ώστε να

δελεάσει τον χρήστη και να του δώσει μια αίσθηση του τρόπου με τον οποίο λειτουργεί. Κάθε στοιχείο μπορεί να πατηθεί και αν κάτι τέτοιο γίνει θα μεταφερθεί στην οθόνη με τις πληροφορίες του άρθρου. Ακριβώς από κάτω υπάρχει μια ακόμα λίστα με αντίκειμενα τα οποία είναι επιπρόσθετα άρθρα τα οποία ο χρήστης μπορεί να πατήσει και να μεταφερθεί και μέσω αυτών στην οθόνη με τις πληροφορίες του αντίστοιχου άρθρου στο οποίο έχει πατήσει.

- Νέα:

Εικόνα 4: Mockup οθόνης NewsPage 1

Στην Εικόνα 4 υπάρχει η οθόνη των Νέων. Εδώ ο χρήστης έχει την δυνατότητα να προβάλει τα νέα με βάση την κατηγοριοποίηση που έχει καθοριστεί. Συγκεκριμένα, τα άρθρα πέρα από τις πληροφορίες τους, όπως ο τίτλος και το κυρίως περιεχόμενο τους, χαρακτηρίζονται και από ένα πεδίο "Κατηγορίας" το οποίο αξιοποιείται στην οθόνη των νέων έτσι ώστε να μπορέσει ο χρήστης να έχει πρόσβαση στην κατηγορία της προτίμησης του και να μην χρειαστεί να χάνει τον χρόνο του ψάχνοντας. Έτσι λοιπόν η οθόνη ξεκινάει με μια λίστα επιλογών που αποτελεί τις διαθέσιμες κατηγορίες της εφαρμογής που είναι "Ποδόσφαιρο", "Basket", "Tennis", "Volley" και "Άλλα". Όταν ο χρήστης πατήσει πάνω σε μια από τις κατηγορίες αυτό θα ενεργοποίησει τον μηχανισμό κατηγοροποίησης των άρθρων και το υπόλοιπο μέρος της οθόνης, το οποίο αποτελείται από άρθρα της αντίστοιχης επιλεγμένης κατηγορίας θα αλλάξει

με βάση την κατηγορία την οποία ο χρήστης επιλέξει. Έτσι λοιπόν αν για παράδειγμα ο χρήστης πατήσει πάνω στο κουμπί του Basket, το υπόλοιπο τμήμα της οθόνης θα αλλάξει και έτσι πλέον η λίστα των άρθρων που θα απεικονίζονται θα περιλαμβάνει μόνο άρθρα τα οποία είναι σχετικά με την κατηγορία Basket.

- Πληροφορίες Άρθρου:

Εικόνα 5: Mockup οθόνης NewsDetails 1

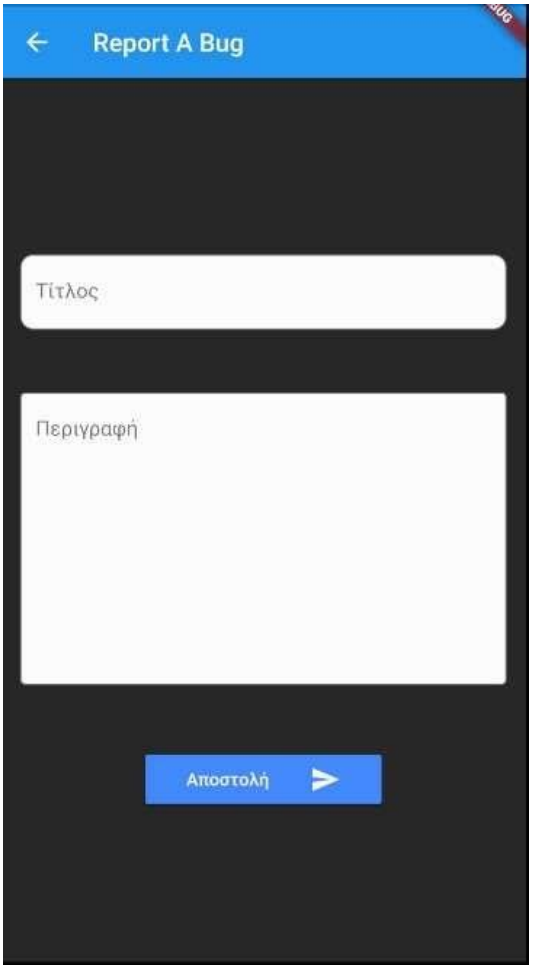
Στην Εικόνα 5 υπάρχει η απεικόνιση της οθόνης με τις πληροφορίες ενός άρθρου. Στην οθόνη αυτή η διάταξη έχει ως εξής: Ξεκινώντας από πάνω μπορούμε να δούμε ότι η μπάρα έχει αλλάξει και πλέον εμφανίζεται μόνο ένα βέλος στο αριστερό μέρος. Εκείνο το κομμάτι είναι ένα κουμπί το οποίο χρησιμοποιείται για την πλοήγηση μέσα στην εφαρμογή και ουσιαστικά μεταφέρει τον χρήστη στην προηγούμενη οθόνη απο εκεί όπου πάτησε πάνω στο αντίστοιχο άρθρο. Όσον αφορά το περιεχόμενο της οθόνης αρχικά ξεκινάει με την εικόνα του άρθρου στο πάνω μέρος και τον τίτλο του άρθρου. Στην συνέχεια ακολουθεί η σύντομη περιγραφή του άρθρου και έπειτα υπάρχει το κυρίως κείμενο. Βασικό χαρακτηριστικό αυτής της οθόνης είναι ότι το μέγεθος της και συγκεκριμένα το ύψος της είναι δυναμικό βάσει του περιεχομένου της. Αυτό σημαίνει ότι ανάλογα το μέγεθος του κειμένου και του πλήθους των εικόνων που ακολουθούν θα υπολογίζεται αντίστοιχα και το ύψος που θα χρειαστεί για να μπορέσει να χωρέσει όλο το περιεχόμενο. Τέλος στο κάτω μέρος της οθόνης υπάρχουν αντίστοιχες εικόνες του άρθρου οι οποίες έχουν προστεθεί.

- Podcasts:

Εικόνα 6: Mockup οθόνης Podcasts

Η Εικόνα 6 είναι η οθόνη των Podcasts που ουσιαστικά αποτελείται από δύο βασικά στοιχεία. Αρχικά υπάρχει ένα κάθετο καρουσελ με τα διαθέσιμα Podcasts. Ο χρήστης μπορεί να γυρίσει το carousel έτσι ώστε να πλοηγηθεί στα διάφορα διαθέσιμα podcasts. Κάθε φορά θα μεταφέρεται στο κέντρο του carousel ένα στοιχείο και αυτό θα θεωρείται ως η επιλογή του χρήστη. Με βάση λοιπόν αυτήν την επιλογή θα λειτουργεί αντίστοιχα και το δεύτερο κομμάτι της οθόνης. Κάτω από το καρουσελ λοιπόν βρίσκεται ένας αναπαραγωγέας ηχητικών αρχείων ο οποίος χρησιμοποιείται για να μπορέσει να γίνει η αναπαραγωγή του Podcast που ο χρήστης έχει επιλέξει. Ο αναπαραγωγέας έχει τρείς βασικές επιλογές: *Έναρξη*, *Παύση* και *Σταμάτημα*. Πατώντας το κουμπί της έναρξης ο χρήστης μπορεί να ξεκινήσει την αναπαραγωγή του ηχητικού αρχείου. Με το κουμπί *Παύση*, μπορεί να σταματήσει την αναπαραγωγή σε οποιαδήποτε χρονική στιγμή και να μπορέσει να συνεχίσει την αναπαραγωγή ξανά από το ίδιο σημείο. Τέλος το κουμπί *Σταμάτημα* σταματάει την αναπαραγωγή και μεταφέρει τον δείκτη στην αρχή.

- Report a Bug Φόρμα:



Εικόνα 7: Mockup οθόνης Report A Bug

Στην Εικόνα 7, η Report a Bug οθόνη αποτελεί μια απλή φόρμα αποστολής σχολίων. Αποτελείται από δύο πεδία εισαγωγής κειμένου όπου ο χρήστης, μπορεί να γράψει τα σχόλια του. Για να μπορέσει να γνωρίζει ο χρήστης σε τι ακριβώς απευθύνεται κάθε πεδίο το τι ακριβώς εξυπηρετεί το κάθε πεδίο αναγράφεται μέσα στα ίδια τα πεδία. Τέλος, στο κάτω μέρος, υπάρχει ένα κουμπί όπου, αφού ο χρήστης εισάγει την απαραίτητη πληροφορία ενεργοποιεί τον μηχανισμό αποστολής του μηνύματος.

Στο κεφάλαιο αυτό λοιπόν, έγινε εμφανής ότι τελικά ο σχεδιασμός των διεπαφών χρήστης αποτελούν ένα απαραίτητο κομμάτι της όλης διαδικασίας το οποίο δεν θα πρέπει να παραβλέπεται και να λαμβάνεται υπόψη εξίσου με όλα τα υπόλοιπα. Έχοντας λοιπόν ένα βασικό πλάνο περί του πως θα είναι η διάταξη της κάθε οθόνης και γενικότερα της εφαρμογής, συνεχίζουμε με την δημιουργία της ίδιας της εφαρμογής.

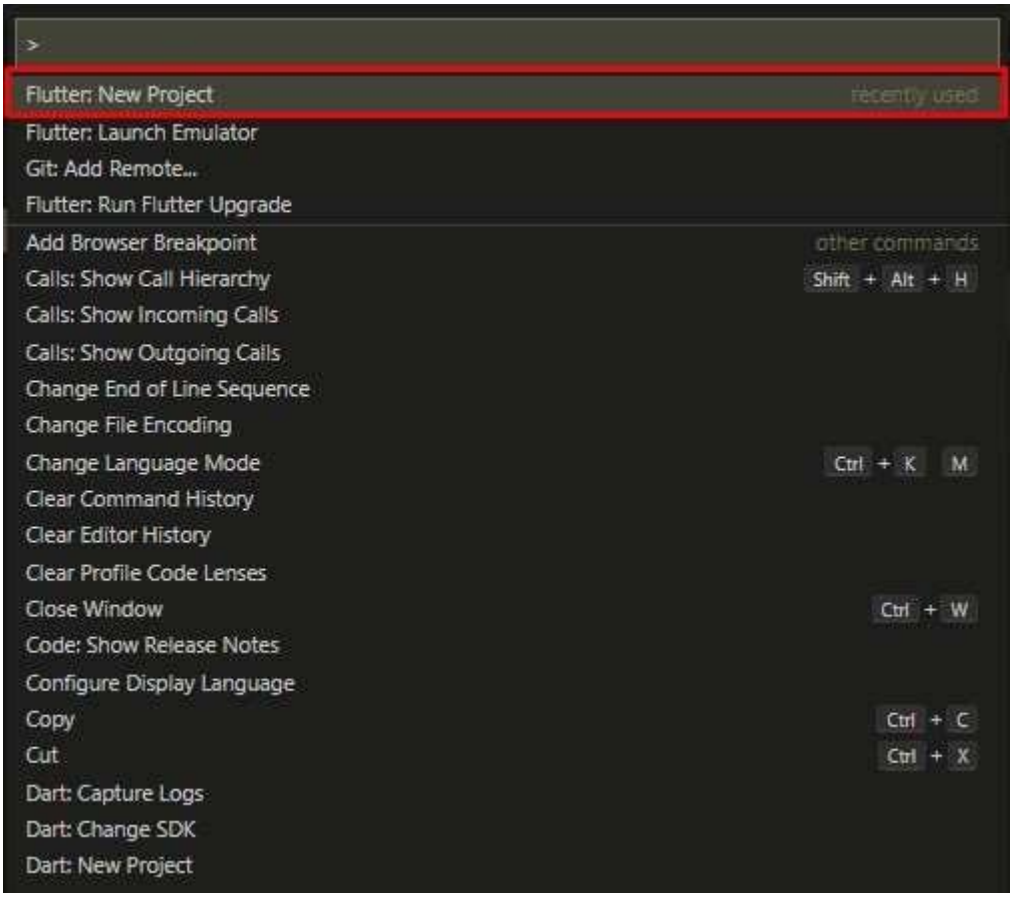
Κεφάλαιο 3ο: Ανάπτυξη Εφαρμογής

3.1 Εισαγωγή

Το βασικό περιβάλλον ανάπτυξης για την τρέχουσα εφαρμογή ήταν το Visual Studio Code. Ο λόγος χρήσης του στην συγκεκριμένη περίπτωση είναι ότι προσφέρει έναν Editor ο οποίος δίνει την δυνατότητα στον χρήστη να υλοποιήσει γρήγορους κύκλους διαδικασιών όπως Debugging, Task Running και Version Control κάτι το οποίο σε συνδυασμό με τα χαρακτηριστικά του Flutter είναι ένα ιδανικό σενάριο για κάθε Developer. Ένα από τα βασικά Features του Visual Studio Code είναι τα Extensions. Τα Extensions αποτελούν ένα τρόπο για την προσθήκη γλωσσών προγραμματισμού, εργαλείων Debugging και γενικών εργαλείων για την υποστήριξη της ροής του Development. Το τρέχων Project περιέλαβε την χρήση τριών βασικών Extensions, του Flutter, του Dart και του GitHub. Το Flutter Extension παρέχει υποστήριξη για το Flutter ως προς την ολότητα του και Debugger, κάτι το οποίο ισχύει αντίστοιχα και για το Dart Extension. Το GitHub Extension παρέχει ενσωμάτωση του GitHub μέσα στο Visual Studio Code κάτι το οποίο χρησιμοποιήθηκε συχνά καθώς το Project έχει το δικό του Private GitHub Repository.

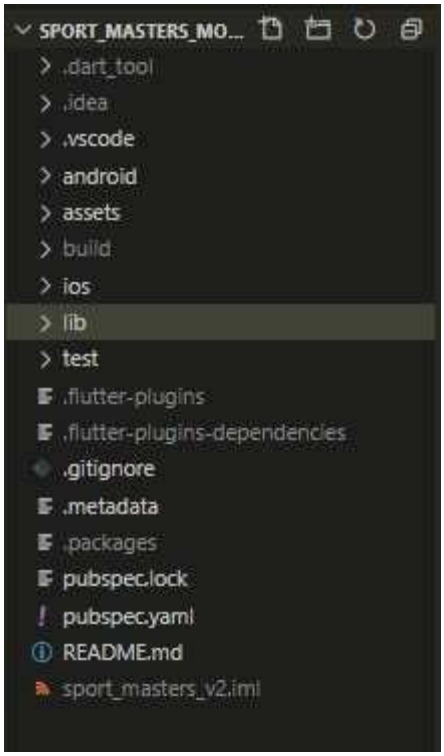
3.2 Εκκίνηση Ανάπτυξης

Για την εκκίνηση ενός Flutter Project υπάρχουν δύο βασικοί τρόποι. Ο ένας είναι με την χρήση της εντολής “flutter create” μέσω τερματικού συστήματος και ο άλλος είναι η χρήση της ίδιας εντολής αλλά με την χρήση του Command Palette του Visual Studio. Και οι δύο τρόποι έχουν το ίδιο αποτέλεσμα. Ο πρώτος τρόπος είναι ο πιο άμεσος και περιλαμβάνει την εκτέλεση της εντολής “flutter create my_project_name”, όπου my_project_name είναι το όνομα του φακέλου στον οποίο θα γίνει η αρχικοποίηση και εγκατάσταση των απαραίτητων αρχείων για την ύπαρξη ενός Flutter Project. Το ίδιο μπορεί να γίνει και με την χρήση του Visual Studio μέσω του Command Palette, η οποία εμφανίζεται στην Εικόνα 8, όπου γίνονται τα ίδια βήματα απλά με πιο φιλικό προς τον χρήστη τρόπο.



Εικόνα 8: Command Palette

Αφού γίνει η εκτέλεση της εντολής γίνονται η απαραίτητες διαδικασίες από πλευράς Flutter για να μπορέσει να γίνει το Setting του Project. Μόλις τελειώσει η διαδικασία στο Visual Studio εμφανίζεται ο φάκελος που ονομάσαμε και ορίσαμε ως βάση του Project και η αντίστοιχη δομή του. Η δομή του αποτελείται από συγκεκριμένους φακέλους οι οποίοι παράγονται δυναμικά με την κάθε δημιουργία ενός νέου Flutter Project. Με βάση την Εικόνα 9 ξεκινώντας από πάνω προς τα κάτω βλέπουμε τους εξής φακέλους:

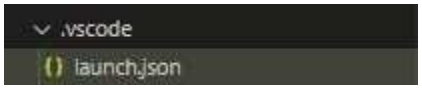


Εικόνα 9: Δομή Flutter Project

1) .dart_tool και .packages: Οι φάκελοι αυτοί αρχίζουν να υπάρχουν μόλις ο γίνει εκτέλεση της εντολής pub get. Θα γίνει εξήγηση του τι ακριβώς κάνει αυτή η εντολή παρακάτω όταν μιλήσουμε για το αρχείο pubspec.yaml

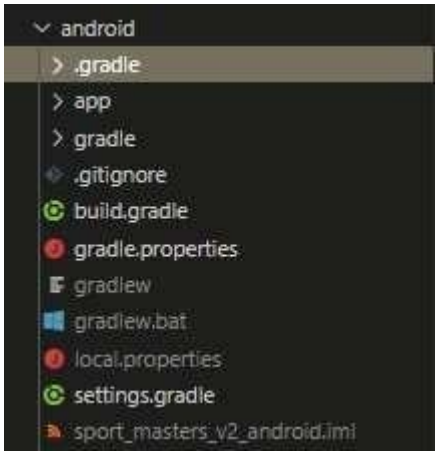
2) .idea: Ο φάκελος αυτός περιλαμβάνει συγκεκριμένες ρυθμίσεις οι οποίες αφορούν το Android Studio. Επειδή στην περίπτωση μας δεν γίνεται χρήση του Android Studio, θα αγνοήσουμε τα περιεχόμενα αυτού του φακέλου.

3) .vscode: Όπως απεικονίζεται και στην Εικόνα 10, ο φάκελος αυτός περιλαμβάνει αρχεία τα οποία χρησιμοποιεί το Visual Studio Code. Στην συγκεκριμένη περίπτωση ο φάκελος περιλαμβάνει μόνο ένα αρχείο launch.json το οποίο περιέχει ρυθμίσεις οι οποίες βοηθούν στο πιο ομαλό Debugging.



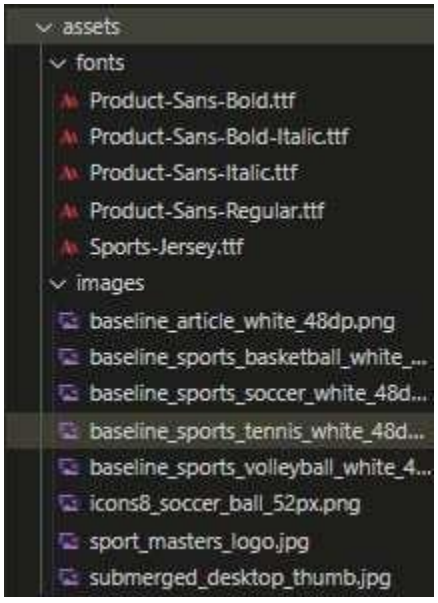
Εικόνα 10: Περιεχόμενα φακέλου .vscode

4) android: Ο επόμενος φάκελος είναι ο android όπως φαίνεται στην Εικόνα 12. Ο συγκεκριμένος φάκελος περιλαμβάνει όλα τα αρχεία τα οποία χρειάζονται για ένα κανονικό Native Android App. Ο φάκελος αυτός χρησιμοποιείται όταν γίνεται build της Flutter εφαρμογής για Android. Στην συγκεκριμένη περίπτωση ο Flutter κώδικας μεταγλωττίζεται και εισάγεται σε αυτόν τον φάκελο με αποτέλεσμα την δημιουργία μιας Native Android εφαρμογής. Στην περισσότερες περιπτώσεις δεν χρειάζεται να πειράξουμε κάτι στα αρχεία τα οποία περιλαμβάνονται μέσα στον φάκελο ή να προσθέσουμε, ωστόσο στην δικιά μας περίπτωση θα δούμε ότι χρειάστηκαν να υπάρξουν κάποιες αλλαγές, όπως θα δούμε και παρακάτω.



Εικόνα 11: Δομή Android Project

5) assets: Ο φάκελος αυτός δημιουργήθηκε για την καλύτερη οργάνωση του Project από τον χρήστη και δεν αποτελεί βασικό κομμάτι της δομής του Project. Μέσα του γίνεται η αποθήκευση διάφορων γραφικών αντικειμένων τα οποία χρησιμοποιούνται στο Project. Στην περίπτωση μας υπάρχουν δύο βασικοί φάκελοι που είναι τα Fonts και τα Images. Η δομή του μέσα στο Project φαίνεται στην Εικόνα 12



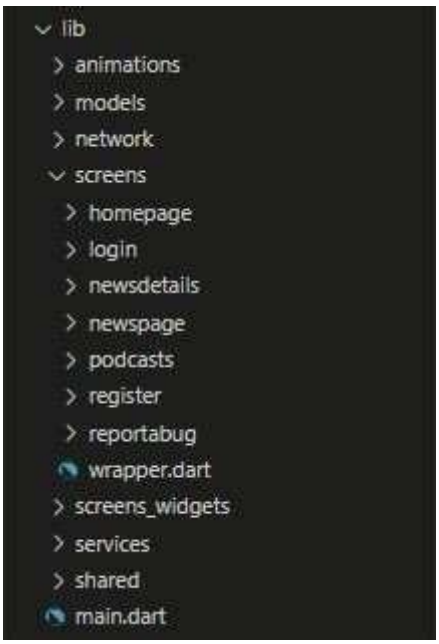
Εικόνα 12: Φάκελος Assets

6) build: Ο φάκελος αυτός περιλαμβάνει τον μεταγλωττισμένο κώδικα της Flutter εφαρμογής. Τα περιεχόμενα του φακέλου αυτού παράγονται δυναμικά, οπότε δεν χρειάζεται και δεν πρέπει να γίνει χειροκίνητη αλλαγή στα περιεχόμενα μέσα σε αυτόν τον φάκελο.

7) ios: Ο φάκελος αυτός περιλαμβάνει ένα ολοκληρωμένο XCode iOS Native Project το οποίο χρησιμοποιείται όταν γίνεται build του Flutter App για iOS. Χρησιμοποιείται με τον ίδιο τρόπο που χρησιμοποιείται και ο φάκελος android. Ο φάκελος αυτός είναι διαθέσιμος μόνο όταν γίνεται

ανάπτυξη της εφαρμογής σε iOS. Επειδή όμως θα κάνουμε την εφαρμογή διαθέσιμη και για iOS ο φάκελος αυτός επομένως είναι διαθέσιμος.

8) lib: Όπως και στην Εικόνα 13, ο φάκελος lib είναι και αυτός που θα χρησιμοποιηθεί περισσότερο. Μέσα σε αυτόν βρίσκονται όλα τα .dart αρχεία τα οποία περιέχουν τον πηγαίο κώδικα για την εφαρμογή. Όταν δημιουργείται περιλαμβάνει αυτόματα το αρχείο main.dart του οποίου την χρήση και κώδικα θα εξηγήσουμε στην συνέχεια.



Εικόνα 13: Περιεχόμενα φακέλου lib

9) test: Ο φάκελος αυτός περιέχει κώδικα ο οποίος είναι γραμμένος για την εφαρμογή ώστε να εκτελέσει αυτοματοποιημένα Tests όταν γίνεται Build

Επόμενο κομμάτι μετά τους φακέλους της δομής του Project είναι και αντίστοιχα αρχεία. Τα αρχεία αυτά είναι:

1) .flutter-plugins: Αρχείο το οποίο παράγεται αυτόματα κατά την προσθήκη Plugins στο Project. Περιέχει πληροφορίες σχετικά με τα Plugins τα οποία είναι εγκατεστημένα στο Project.

2) .flutter-plugins-dependencies: Όπως και το .flutter-plugins παράγεται αυτόματα και περιέχει πληροφορίες σχετικά με τα Dependencies των Plugins

3) .gitignore: Το αρχείο αυτό είναι ένα text αρχείο το οποίο περιέχει μια λίστα από αρχεία, extensions αρχείων και φακέλους οι οποίοι θα πρέπει να αγνοούνται όταν γίνεται χρήση του Git

4) .metadata: Το αρχείο αυτό διαχειρίζεται από το ίδιο το Flutter αυτόματα και χρησιμοποιείται για να ελέγχει κάποιες παραμέτρους του Project.

5) .packages: Παρόμοιο με το αρχείο .metadata, το αρχείο αυτό περιλαμβάνει περιεχόμενο το οποίο παράγεται αυτόματα από το SDK και περιλαμβάνει μια λίστα από Dependencies που χρησιμοποιούνται στο Project.

6) pubspec.lock: Όταν γίνεται προσθήκη ενός νέου Dependency, ο Package Manager του Flutter γράφει σε ένα lockfile για να εξασφαλίσει ότι μελλοντικές προσθήκες θα χρησιμοποιούν τις ίδιες ακριβώς εκδόσεις αυτών των Dependencies. Έτσι στο συγκεκριμένο αρχείο περιλαμβάνονται πληροφορίες σχετικές με όλα τα Dependencies του Project έτσι ώστε να υπάρχουν αποφυγές μελλοντικών σφαλμάτων σε περιπτώσεις αλλαγών.

7) pubspec.yaml: Το αρχείο αυτό είναι ένα γενικό αρχείο ρυθμίσεων για όλο το Project. Στην περίπτωση μας έγινε τροποποίηση του περιεχομένου του κάτι το οποίο θα αναλύσουμε παρακάτω.

8) README.md: Ένα τυπικό README αρχείο που παράγεται αυτόματα.

9) sport_masters_v2.iml(my_app_name.iml): Το αρχείο αυτό έχει πάντα το όνομα το οποίο δίνεται στο Project. Περιέχει περαιτέρω ρυθμίσεις για το Project και συνήθως το περιεχόμενο δεν αλλάζεται από τον χρήστη αλλά δυναμικά από το ίδιο το SDK.

3.3 Pubspec.yaml

Όπως προαναφέρθηκε το αρχείο pubspec.yaml περιέχει γενικές ρυθμίσεις για όλο το Project. Η δομή του αρχείου στο συγκεκριμένο Project έχει την μορφή που εμφανίζεται στην εικόνα παρακάτω.

```yaml
name: sport_masters_v2
description: A new Flutter project.

version: 1.0.0+1

environment:
  sdk: ">=2.2.2 <3.0.0"

dependencies:
  flutter:
    sdk: flutter

  carousel_slider: ^2.2.1
  firebase_auth: ^0.15.1
  cloud_firestore: ^0.13.7
  cached_network_image: ^2.2.0+1
  provider: ^4.3.2
  flutter_spinkit: ^4.1.2
  page_transition: ^1.1.6
  flutter_visualizers: ^0.0.1
  firebase_storage: ^3.1.6
  audioplayers: ^0.15.1
  splashscreen: ^1.2.0
  flutter_launcher_icons: ^0.7.5
  full_screen_image: ^1.0.2

dev_dependencies:
  flutter_test:
    sdk: flutter

flutter_icons:
  android: true
  ios: true
  image_path_android: assets/images/sport_masters_logo.jpg
  image_path_ios: assets/images/sport_masters_logo.jpg
  adaptive_icon_background: assets/images/sport_masters_logo.jpg
  adaptive_icon_foreground: assets/images/sport_masters_logo.jpg

flutter:
  assets:
    - assets/images/submerged_desktop_thumb.jpg
    - assets/images/icons8_soccer_ball_52px.png
    - assets/images/sport_masters_logo.jpg
    - assets/images/baseline_sports_soccer_white_48dp.png
    - assets/images/baseline_sports_basketball_white_48dp.png
    - assets/images/baseline_sports_tennis_white_48dp.png
    - assets/images/baseline_sports_volleyball_white_48dp.png
    - assets/images/baseline_article_white_48dp.png
  fonts:
    - family: Sports Jersey
      fonts:
        - asset: assets/fonts/Sports-Jersey.ttf

    - family: Product Sans
      fonts:
        - asset: assets/fonts/Product-Sans-Regular.ttf
        - asset: assets/fonts/Product-Sans-Bold.ttf
        - asset: assets/fonts/Product-Sans-Italic.ttf
        - asset: assets/fonts/Product-Sans-Bold-Italic.ttf

  uses-material-design: true
```

Εικόνα 14: Κώδικας αρχείου pubspec.yaml

Βλέπουμε ότι το αρχείο της Εικόνας 14 ακολουθεί μια μορφή του τύπου Key-Value Pair, δηλαδή για κάθε κλειδί αντιστοιχεί και μια τιμή. Συγκεκριμένα υπάρχουν τα εξής ζεύγη:

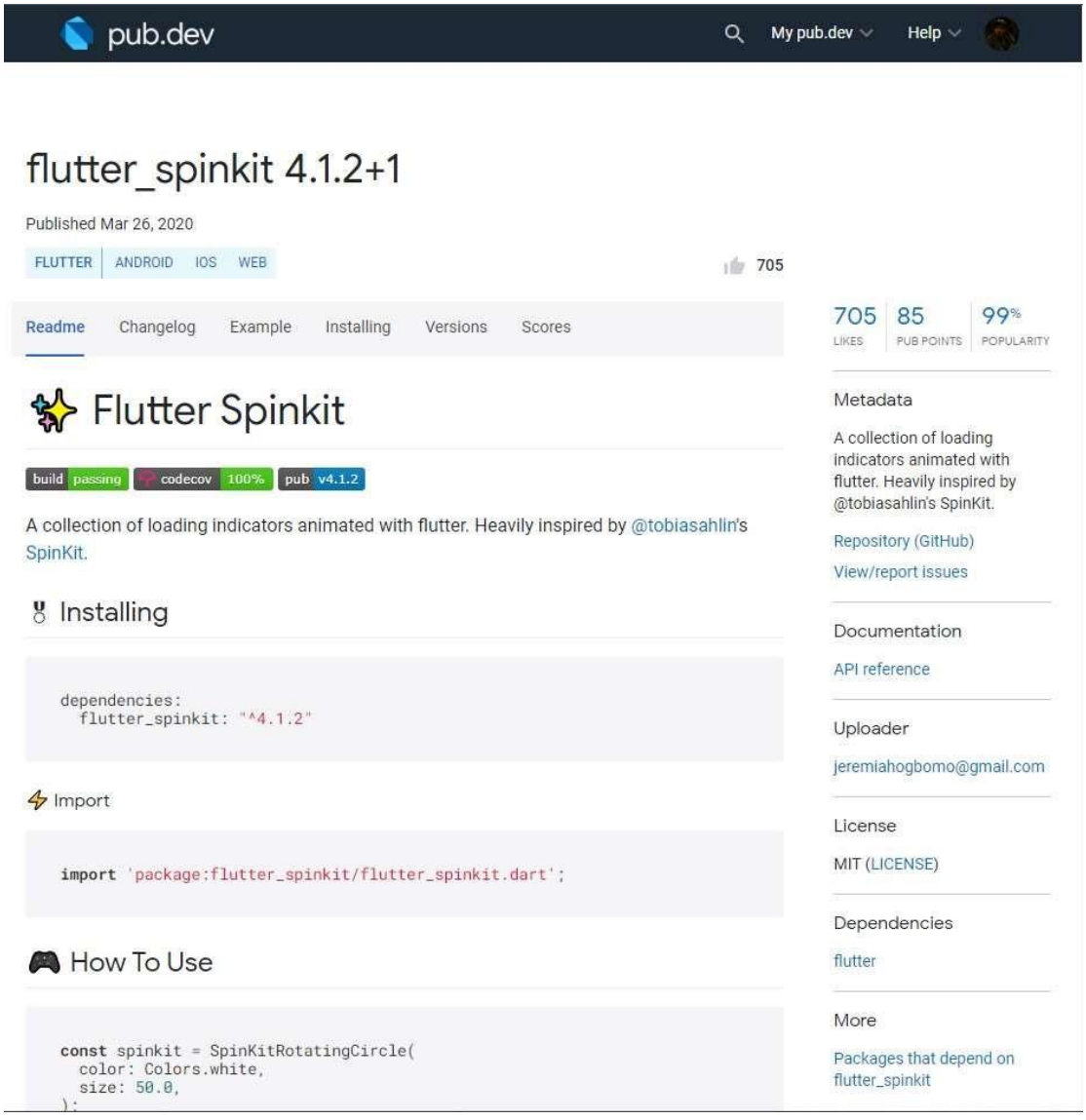
1) name: Ξεκινώντας μπορούμε να δούμε ότι το πρώτο ζευγάρι έχει ως κλειδί την τιμή “name”. Το συγκεκριμένο ζευγάρι αναφέρεται στο όνομα του Package για το τρέχον Project. Κάθε Package πρέπει να έχει ένα αντίστοιχο όνομα.

2) description: Στην συνέχεια ακολουθεί το description το οποίο είναι προαιρετικό και είναι μια περιγραφή του Package.

3) version: Κάθε Package πρέπει να έχει μια έκδοση αν ο χρήστης το δημοσιεύσει στην πλατφόρμα pub.dev. Στην περίπτωση μας δεν έχουμε κάποια τέτοια ανάγκη οπότε δεν θα εμβαθύνουμε περισσότερο.

4) dependencies: Τα Dependencies είναι Packages πάνω στα οποία εξαρτιέται το δικό μας Package που είναι και το Project. Η επιλογή των Dependencies έγινε με βάση τις ανάγκες που υπήρχαν για

να μπορέσει να υλοποιηθεί το Project. Για τον ορισμό των Dependencies το πρώτο βήμα είναι να οριστεί ως τιμή του κλειδιού “dependencies ” το ζεύγος του Dependency το οποίο αποτελείται από το όνομα του package και την διαθέσιμη έκδοση για χρήση από το Project. Τα ζεύγη αυτά μπορεί να τα βρεί κάποιος στο pub.dev όπου υπάρχει Documentation για κάθε διαθέσιμο Package και οδηγίες για την εγκατάσταση των πακέτων. Ένα παράδειγμα του πως είναι ένα Dependency στο site του pub.dev φαίνεται και στην Εικόνα 15.



Εικόνα 15: Landing Page του Flutter Spinkit Package στο pub.dev

5) dev_dependencies: Οι Dev Dependencies διαφέρουν από τα κανονικά Dependencies στο ότι τα Dependencies από τα Packages τα οποία εξαρτιέται το Project και χρησιμοποιούνται για την ανάπτυξη του Package και όχι χρήση του αγνοούνται. Αυτό σημαίνει ότι κάθε φορά που γίνεται ανάκληση για ένα Package πληροφορία η οποία δεν είναι χρήσιμη για το τελικό πακέτο δεν λαμβάνεται υπόψη αυξάνοντας έτσι την ταχύτητα των αιτημάτων.

6) flutter_icons: Το συγκεκριμένο κλειδί αποτελεί προαπαιτούμενο του Dependency flutter_launcher_icons το οποίο αφορά τα launching εικονίδια τα οποία θα εμφανίζονται στην συσκευή του χρήστη. Θα γίνει περαιτέρω αναφορά σε αυτό παρακάτω όταν θα μιλήσουμε αναλυτικότερα για τα Dependencies του Project.

7) flutter: Το κλειδί αυτό περιλαμβάνει ως τιμές τα αντίστοιχα assets τα οποία θέλουμε να χρησιμοποιήσουμε μέσα στο Project. Έτσι ορίστηκαν νέοι πίνακες ζευγών, ένας για τα Assets όπου κάθε σειρά περιλαμβάνει για τιμή το αντίστοιχο μονοπάτι μέσα στην δομή του Project για την κάθε εικόνα που χρησιμοποιήθηκε και τα fonts, στα οποία έχουν οριστεί η οικογένεια του Font καθώς και το αντίστοιχο αρχείο το οποίο αποτελεί το Font αυτό καθεαυτό.

8) uses-material-design: Το συγκεκριμένο κλειδί επιτρέπει την χρήση του προκαθορισμένου σετ Material εικονιδίων. Τα Material εικονίδια αποτελούν κομμάτι του Material Design. Το Material Design αποτελεί μια γλώσσα σχεδιασμού η οποία αναπτύχθηκε από την Google και στον πυρήνα της είναι ένα σετ από κανόνες και βασικούς άξονες οι οποίοι έχουν ως τελικό σκοπό την καλύτερη ενοποίηση των βασικών αρχών σχεδιασμού με την τεχνολογία.

3.4 Project Packages

Βασικό σημείο αυτού του Project είναι τα Packages. Το Flutter υποστηρίζει την χρήση δημόσιων Packages τα οποία μπορούν να χρησιμοποιηθούν από όλους τους χρήστες Flutter και Dart. Η χρήση των Packages αποτελεί ένα τεράστιο πλεονέκτημα καθώς επιταχύνει την διαδικασία ανάπτυξης μιας εφαρμογής αφαιρώντας την υποχρέωση του προγραμματιστή να χρειαστεί να φτιάξει το Project από το μηδέν.

Τα Packages δημοσιεύονται στην πλατφόρμα pub.dev. Εκεί κάποιος μπορεί να πλοηγηθεί και να βρεί μια ποικιλία Packages τα οποία μπορούν να εκτελούν αιτήματα στο διαδίκτυο(http) μέχρι και την δημιουργία ειδικά σχεδιασμένων στοιχείων της διεπαφής χρήστη για την κάλυψη συγκεκριμένων αναγκών.

Για να μπορέσει κάποιος να προσθέσει ένα Package πρέπει να δημιουργήσει ένα Package Dependency. Για να γίνει αυτό αρκεί να δημιουργήσει ένα ζεύγος κλειδιού-τιμής στο αρχείο pubspec.yaml με τιμές αυτές που έχουν καθοριστεί από το αντίστοιχο Documentation του Package το οποίο βρίσκεται στο pub.dev. Στο τρέχον Project έγινε η χρήση των εξής Dependencies:

- Carousel Slider: Το τρέχων Package του οποίου το παράδειγμα φαίνεται στην Εικόνα 16 προσφέρει την δυνατότητα απεικόνισης αντικειμένων με την μορφοποίηση Carousel. Το Package αυτό χρειάστηκε καθώς βοήθησε στο να αναπτυχθεί η διεπαφή του χρήστη με βάση τα Mockups.

Εικόνα 16: Δείγμα UI του Carousel Slider Package

- Firebase Auth: Package το οποίο δίνει την δυνατότητα χρήσης του Firebase Authentication API. Το API αυτό στοχεύει στο να δώσει στον χρήστη την δυνατότητα προσθήκης

  αυθεντικοποίησης με την πλατφόρμα Firebase όσο το δυνατόν πιο εύκολα και γρήγορα γίνεται.

- Cloud Firestore: Package το οποίο χρησιμοποιείται για να μπορέσει ο χρήστης να έχει πρόσβαση στο Cloud Firestore API. Με αυτό το Package θα μπορέσουμε να έχουμε πρόσβαση στις εγγράφα τα οποία χρειάζονται και περιλαμβάνουν τα δεδομένα τα οποία εμείς θα χρειαστούμε για να λειτουργήσει το App.
- Provider: Επειδή ο τρόπος με τον οποίο είναι δομημένη η ιεραρχία των Widgets είναι ιεραρχική και τα Widgets είναι εμφωλευμένα το ένα μέσα στο άλλο ο Provider θα βοηθήσει για την μεταφορά των δεδομένων από το ένα Widget στο άλλο με μεγαλύτερη ευκολία. Θα το αναλύσουμε περαιτέρω όταν θα μιλήσουμε για τον πηγαίο κώδικα του Project

- Flutter Spinkit: Το Package αυτό προσφέρει μια συλλογή από Loading Indicators, κάτι το οποίο θα χρησιμοποιήσουμε όταν γίνεται η φόρτωση των δεδομένων ως μια βιτρίνα για την καλύτερη εμπειρία του χρήστη.

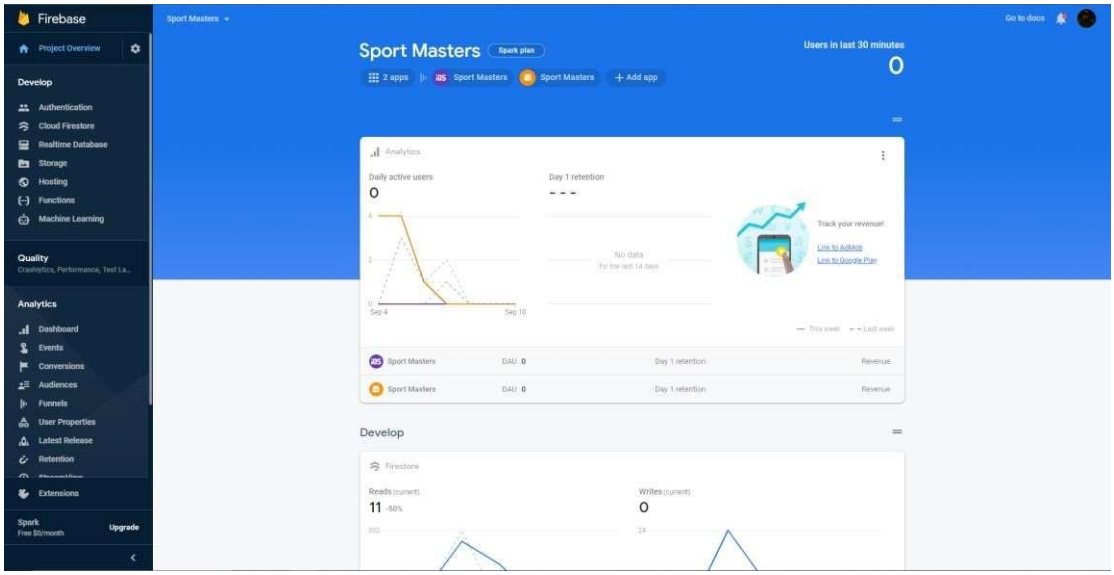
- Audioplayers: Επειδή μέσα στις λειτουργικές απαιτήσεις βρίσκεται και η αναπαραγωγή του ακουστικού περιεχομένου το οποίο υπάρχει υπό την μορφή των Podcast, στην εφαρμογή θα χρησιμοποιηθεί και αυτό το Package το οποίο προσφέρει την δυνατότητα αναπαραγωγής ακουστικού περιεχομένου μέσα στην εφαρμογή.

- Splashscreen: Κατά την έναρξη της εφαρμογής θα υπάρχει μια οθόνη η οποία θα αποτελεί την οθόνη εισόδου ή αλλιώς Splash. Για να το επιτύχουμε αυτό με μεγαλύτερη ευκολία θα χρησιμοποιήσουμε αυτό το Package.

- Flutter Launcher Icons: Το Package αυτό δίνει την δυνατότητα αυτοματοποιημένου ορισμού των Launch Icons της εφαρμογής τόσο σε Android όσο και σε iOS.

3.5 Firebase

Το Firebase είναι μια backend πλατφόρμα για την ανάπτυξη Web, Android και iOS εφαρμογών. Ο τρόπος με τον οποίο είναι δομημένη είναι τέτοιος έτσι ώστε να ελευθερώσει τους χρήστες της πλατφόρμας από της διαδικασίες που χρειάζονται για την ανάπτυξη και διαχείριση του Back-end από την αρχή, δίνοντας σε όλους την δυνατότητα να επικεντρωθούν στο να δημιουργούν ακόμα καλύτερες εμπειρίες για τον χρήστη. Με το Firebase δεν χρειάζεται να γίνει γραφή του API και αυτό γιατί το Firebase είναι ο Server, το API και το Data Storage. Πέραν των λειτουργιών που αναφέρθηκαν παραπάνω οι οποίες παρέχονται από την πλατφόρμα, το Firebase αναλαμβάνει ακόμα και λειτουργίες όπως αυθεντικοποίηση χρηστών, Hosting περιεχομένου με γρήγορες και ασφαλείς παραδόσεις, δημιουργία Analytics για μια λεπτομερής ανάλυση της κατάστασης του Back-End και πολλά άλλα. Η πλατφόρμα είναι προσβάσιμη μέσω Web και η Εικόνα 17 δείχνει την κονσόλα διαχείρισης της πλατφόρμας.



Εικόνα 17: Γραφική διεπαφή της κονσόλας του Firebase

3.6 Cloud Firestore

Μια από τις λειτουργίες του Firebase που χρησιμοποιείται σε αυτό το Project είναι το Cloud Firestore. Το Cloud Firestore είναι μια ευέλικτη και κλιμακούμενη βάση δεδομένων για την ανάπτυξη Mobile, Web και Server εφαρμογών. Οι βασικές λειτουργίες οι οποίες προσφέρει είναι:

- Ευελιξία: Δίνεται η δυνατότητα δημιουργίας ευέλικτων, ιεραρχικών δομών δεδομένων. Τα έγγραφα είναι οργανωμένα σε συλλογές και τα ίδια μπορούν να περιλαμβάνουν πολύπλοκα εμφωλευμένα αντικείμενα.

- Expressive Querying: Τα ερωτήματα τα οποία γίνονται στην βάση δεδομένων μπορούν να χρησιμοποιηθούν για να επιστρέψουν συγκεκριμένα έγγραφα ή για να επιστρέψουν όλα τα διαθέσιμα έγγραφα τα οποία ταιριάζουν με τις παραμέτρους του ερωτήματος, όπου αυτές μπορούν να περιλαμβάνουν πολλαπλά αλυσιδωτά φίλτρα και ακόμη συνδυασμό φίλτρων και ταξινόμησης. Επίσης οι καταχωρήσεις είναι ευρετηριοποιημένες, οπότε η απόδοση του

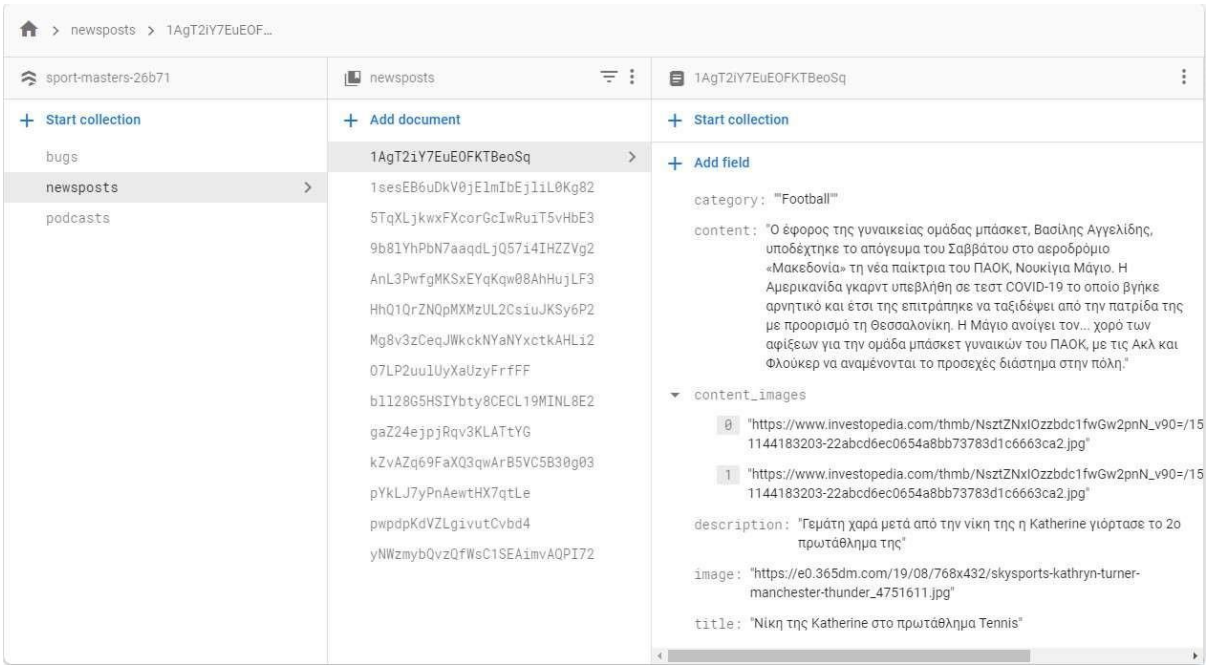
ερωτήματος δεν εξαρτάται από το μέγεθος του σετ δεδομένων αλλά από το μέγεθος του επιστρεφόμενου σετ.

- Ενημερώσεις σε πραγματικό χρόνο: Το Cloud Firestore χρησιμοποιεί συγχρονισμό δεδομένων για την ενημέρωση των δεδομένων σε οποιαδήποτε συνδεδεμένη συσκευή
- Υποστήριξη για λειτουργία εκτός σύνδεσης: Το Cloud Firestore αποθηκεύει τα δεδομένα στην Cache ώστε ο χρήστης να μπορεί να κάνει εκτελεί όποια ενέργεια επιθυμεί ακόμα και όταν είναι εκτός σύνδεσης
- Σχεδιασμένο για κλιμάκωση

3.6.1 Cloud Firestore Model

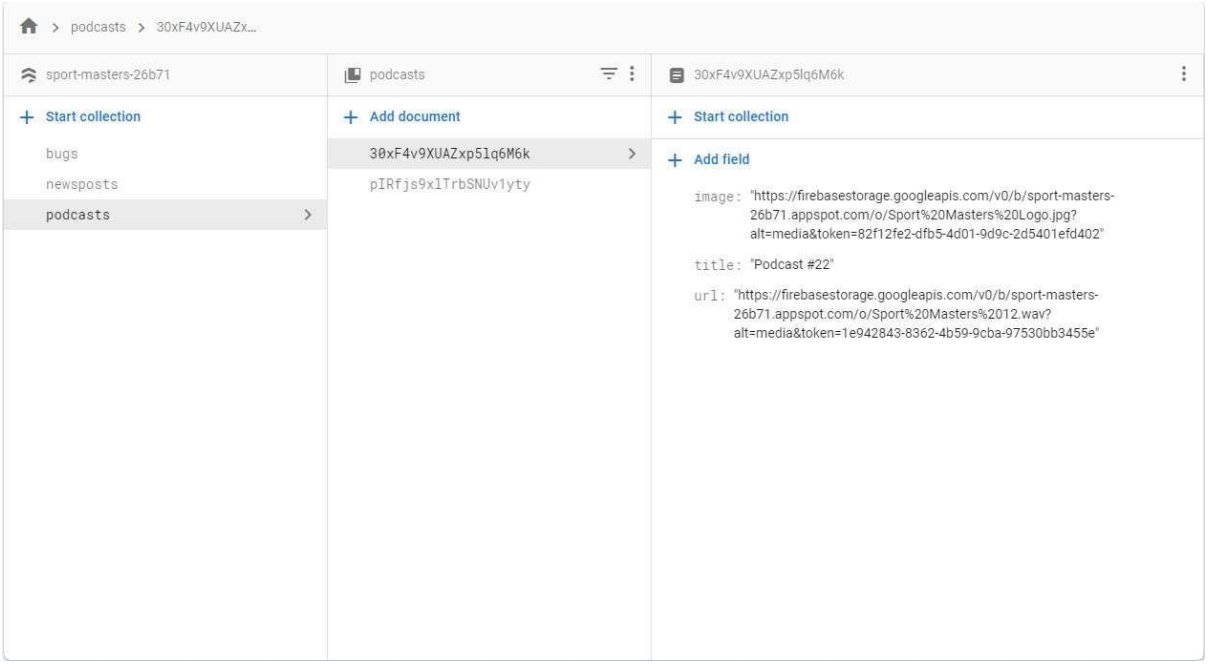
Η δομή του μοντέλου αποτελείται από τρείς βασικές οντότητες. Αυτές είναι τα newsposts, τα podcasts και τα bugs. Συγκεκριμένα:

- newsposts: Όπως φαίνεται στην Εικόνα 18, τα News Posts είναι το βασικό αντικείμενο αυτής της εφαρμογής. Τα έγγραφα τα οποία περιλαμβάνονται μέσα στην βάση θα χρησιμοποιηθούν για να απεικονιστούν οι αντίστοιχες πληροφορίες σχετικές με τα αθλητικά άρθρα τα οποία θέλουμε να εμφανίσουμε. Το κάθε έγγραφο αποτελείται από τα πεδία:
  - o title: Τίτλος του άρθρου
  - o description: Περιγραφή του άρθρου
  - o image: Η εικόνα που θα χρησιμοποιείται ως προεπισκόπηση σε όλες τις οθόνες ο
  - content: Κυρίως περιεχόμενο του άρθρου
  - o content_images: Φωτογραφίες που θα υπάρχουν στην οθόνη με το περιεχόμενο του
  - άρθρου ο category: Πεδίο το οποίο αναφέρεται στην κατηγορία του άρθρου.



Εικόνα 18: Δείγμα εγγράφου newsposts

- podcasts: Όπως φαίνεται και στην Εικόνα 19, τα Podcasts είναι το αντικείμενο που χρησιμοποιείται για να μπορέσουμε να τραβήξουμε τα αποθηκευμένα Podcasts στο Firebase Storage. Τα πεδία του εγγράφου είναι:
  - image: Εικόνα του Podcast για προεπισκόπηση
  - title: Τίτλος Podcast
  - url: Url του Podcast με μορφή .mp3 στο Firebase Storage



Εικόνα 19: Δείγμα εγγράφου podcasts

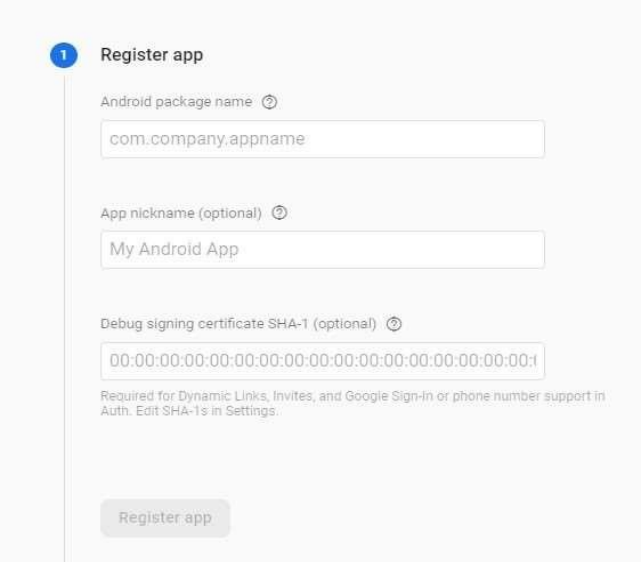
- bugs: Στην εφαρμογή θα υπάρχει η δυνατότητα αναφοράς Bug οπότε θα περιλαμβάνεται και η συγκεκριμένη δομή με τα πεδία:
  - title: Τίτλος Bug
  - description: Περιγραφή Bug

Το μοντέλο αποτελεί το βασικό κομμάτι του Back-End καθώς θα το χρησιμοποιήσουμε και στον κώδικα της εφαρμογής για να μπορέσουμε να χαρτογραφήσουμε τα δεδομένα τα οποία θα δεχόμαστε κάθε φορά από την βάση δεδομένων με βάση αυτό.

3.6.2 Δημιουργία Σύνδεσης με Firebase

Για να μπορέσει να επικοινωνήσει η εφαρμογή με τους χρήστες θα πρέπει πρώτα να γίνουν οι κατάλληλες ενέργειες για την εγκαθίδρυση της σύνδεσης ανάμεσα σε Firebase και Flutter. Θα πρέπει να σημειωθεί ότι ενώ ο κώδικας για την δημιουργία των εφαρμογών σε Android και iOS είναι κοινός όσον αφορά το Firebase, χρειάζεται να γίνει καταχώρηση και των δύο εφαρμογών για να λειτουργήσει σωστά. Για να μπορέσει να επιτευχθεί αυτό θα πρέπει να εκτελεστούν τα κατάλληλα βήματα.

Δημιουργία σε Android: Πρώτο βήμα είναι να γίνει καταχώρηση της εφαρμογής συμπληρώνοντας τα στοιχεία τα οποία φαίνονται στην Εικόνα 20.



Εικόνα 20: Φόρμα δήλωσης εφαρμογής στην πλατφόρμα Firebase

Στην συνέχεια χρειάζεται το αρχείο google-services.json. Το αρχείο δημιουργείται αυτόματα και περιέχει πληροφορίες σχετικά με την εγκαθίδρυση της σύνδεσης ανάμεσα στην εφαρμογή και το Firestore. Το αρχείο αυτό πρέπει να τοποθετηθεί μέσα στο project για να μπορέσουν να εκτελεστούν τα επόμενα βήματα.

Επόμενο βήμα είναι να γίνουν κάποιες τροποποιήσεις στα αρχεία build.gradle τα οποία βρίσκονται στο φάκελο android. Οι αλλαγές είναι αυτές που φαίνονται στην Εικόνα 21.

Project-level build.gradle (<project>/build.gradle):

```
buildscript {
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
  }
  dependencies {
    ...
    // Add this line
    classpath 'com.google.gms:google-services:4.3.3'
  }
}

allprojects {
  ...
  repositories {
    // Check that you have the following line (if not, add it):
    google()  // Google's Maven repository
    ...
  }
}
```

App-level build.gradle (<project>/<app-module>/build.gradle):

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
  // add the Firebase SDK for Google Analytics
  implementation 'com.google.firebase:firebase-analytics:17.5.0'
  // add SDKs for any other desired Firebase products
  // https://firebase.google.com/docs/android/setup#available-librari
}
```

Εικόνα 21: Πηγαίος κώδικας για προσθήκη στο αρχείο build.gradle

Στο build.gradle το οποίο βρίσκεται στο επίπεδο του Project πρέπει να γίνει έλεγχος για το αν υπάρχει η γραμμή η οποία προσφέρει σύνδεση με το Maven Repository, και για να ενεργοποιηθούν οι υπηρεσίες των Google API και Firebase πρέπει να προσθέσουμε την γραμμή "classpath: 'com.google.gms: google-services:4.3.3'". Στην συνέχεια στο build.gradle επιπέδου εφαρμογής προσθέτουμε τις γραμμές οι οποίες κάνουν εφαρμογή των plugin μέσα στο Project για να μπορέσει το Android App να τα αξιοποιήσει όταν θα γίνει build της εφαρμογής.

Παρόμοια είναι και η διαδικασία σε iOS όπου πρώτα ο χρήστης πρέπει να καταχωρήσει το iOS App χρησιμοποιώντας το iOS Bundle ID το οποίο είναι ένα μοναδικό αναγνωριστικό κάθε iOS εφαρμογή στο οικοσύστημα της Apple. Έπειτα θα χρειαστεί να γίνει η προσθήκη στον ios φάκελο του GoogleService-Info.plist αρχείου το οποίο παράγεται αυτόματα. Στην συνέχεια για να γίνει προσθήκη του Firebase SDK χρειάζεται να γίνει η προσθήκη του Cocoapods. Το Cocoapods είναι ένας Dependency Manager για το iOS. Για να μπορέσει να γίνει χρήση του πρέπει πρώτα να δημιουργηθεί ένα Podfile μέσα στο Project. Αυτό θα γίνει με την εντολή 'pod init'. Έπειτα μέσα στο Podfile θα προστεθούν οι αντίστοιχες γραμμές για την εισαγωγή των Pods και με την εντολή 'pod install' θα γίνει η εγκατάσταση των Pods στο Project. Αυτό θα δημιουργήσει ένα xcworkspace. Κανονικά με την δημιουργία ενός Project δημιουργείται ένα .xcproject αρχείο το οποίο αποτελεί το αρχείο που

διαχειρίζεται το αντίστοιχο Project. Ωστόσο με την προσθήκη των Pods δημιουργείται ένα ακόμα Project το οποίο είναι υπεύθυνο για την διαχείριση αυτών των αρχείων. Συνεπώς, παύουμε να μιλάμε για ένα Project και αρχίζουμε να διαχειριζόμαστε ένα Workspace. Τέλος για την σύνδεση χρειάζεται η προσθήκη κάποιων γραμμών στην κλάση AppDelegate του Project. Η AppDelegate κλάση συμφωνεί με το πρωτόκολλο UIApplicationDelegate και είναι υπεύθυνη για την διαχείριση των κοινών συμπεριφορών της εφαρμογής. Χρησιμοποιείται για την εκτέλεση ενεργειών κατά την εκκίνηση της εφαρμογής όπως η αρχικοποίηση των δομών δεδομένων της εφαρμογής, η ανταπόκριση σε ενέργειες οι οποίες δεν είναι συγκεκριμένες στα αντικείμενα τα οποία αποτελούν την όψη της εφαρμογής αλλά επεκτείνονται πέραν αυτού.

Δημιουργώντας λοιπόν αυτά τα αρχεία έχουμε εγκαθιδρύσει σύνδεση με το Firebase και είμαστε έτοιμοι να προχωρήσουμε στην ανάπτυξη της εφαρμογής.

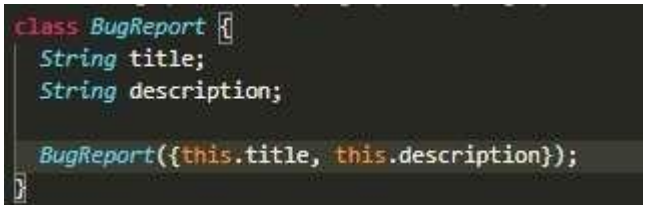
3.7 Ανάλυση Κώδικα

Το Project είναι δομημένο σε τρία βασικά τμήματα. Αρχικά θα μιλήσουμε για την διεπαφή. Η διεπαφή είναι χωρισμένη σε φακέλους όπου κάθε φάκελος είναι και η κάθε οθόνη. Έπειτα ακολουθούν τα μοντέλα τα οποία είναι αυτά που θα χρησιμοποιηθούν για την χαρτογράφηση των δεδομένων από τα αιτήματα στην Βάση Δεδομένων. Τέλος υπάρχει ο φάκελος Services στο οποίο περιλαμβάνονται τα αρχεία τα οποία ορίζουν τα αιτήματα στην Βάση Δεδομένων και τον τρόπο με τον οποίο θα γίνει η αυθεντικοποίηση των χρηστών.

3.7.1 Μοντέλα Δεδομένων

Τα μοντέλα δεδομένων αποτελούν απαραίτητο κομμάτι για να προχωρήσει η εφαρμογή. Στην τρέχουσα εφαρμογή τα μοντέλα τα οποία χρησιμοποιήθηκαν είναι τα εξής:

- Bugreport: Η κλάση αυτή περιλαμβάνει δύο πεδία τον τίτλο (title) και την περιγραφή (description) ενός Bug. Για να μπορέσει να δημιουργηθεί ένα αντικείμενο το οποίο είναι τύπου Bugreport χρειάζεται να οριστεί τουλάχιστον ένας δομητής μέσα στην κλάση. Στην γλώσσα προγραμματισμού Dart ένας δομητής κλάσης ορίζεται δημιουργώντας μια μέθοδο με όνομα το ίδιο με αυτό της κλάσης και ορίζοντας τις παραμέτρους αυτής της μεθόδους. Κανονικά για να οριστεί μια μέθοδος πέρα από το όνομα και τις παραμέτρους χρειάζεται και τις αγκύλες που αποτελούν το σώμα για τις αναθέσεις των τιμών. Ωστόσο στην Dart υπάρχει η έννοια του “Syntactic Sugar”, η οποία σημαίνει πως η ίδια η γλώσσα παρέχει τρόπους για να προκύπτει το ίδιο αποτέλεσμα με πιο εύκολο συντακτικό τρόπο. Στην περίπτωση του δομητή το σώμα αφαιρείται και η ανάθεση τιμών γίνεται αυτόματα. Το μόνο που χρειάζεται είναι το όνομα και οι παρένθεση που περιέχει τις παραμέτρους. Η δομή της φαίνεται και στην Εικόνα 22.



Εικόνα 22: Πηγαίος κώδικας κλάσης BugReport

- Podcast: Η κλάση αυτή αποτελείται από τρία πεδία, τον τίτλο του Podcast, το url του ηχητικού αρχείου και της εικόνας που αντιστοιχεί στο ηχητικό αρχείο. Ακολουθώντας την ίδια φόρμα και αυτή η κλάση αποτελείται από τις παραμέτρους και τον δομητή που της αντιστοιχεί.
- Posts: Η κλάση αυτή αποτελείται από 6 πεδία, την κατηγορία του άρθρου, τον τίτλο του άρθρου, την περιγραφή του, την εικόνα προεπισκόπησης η οποία ουσιαστικά είναι ένα String στο οποίο θα βρίσκεται το url της εικόνας, το κυρίως περιεχόμενο και μια λίστα από URL εικόνων τα οποία θα βρίσκονται στην οθόνη που θα εμφανίζεται το κυρίως περιεχόμενο του κάθε άρθρου.

Έχοντας λοιπόν ορίσει την μορφοποίηση των κλάσεων οι οποίες αποτελούν το μοντέλο της εφαρμογής μπορούμε να προχωρήσουμε στην ανάλυση του τρόπου με τον οποίο γίνεται η επικοινωνία της εφαρμογής με την βάση δεδομένων και πως είναι δομημένες οι κλάσεις για την επίτευξη της επικοινωνίας.

3.7.2 Services

Ο φάκελος Services περιλαμβάνει μέσα όλες τις απαραίτητες κλάσεις που περιέχουν τον κώδικα για ότι αφορά το Back-End κομμάτι της εφαρμογής. Τα αρχεία που περιέχονται είναι τα authentication.dart και database.dart.

Θα ξεκινήσουμε με το αρχείο database.dart. Μέσα στο αρχείο αυτό υπάρχει ο κώδικας ο οποίος περιέχει τις μεθόδους ανάκτησης των δεδομένων από την βάση. Η δομή της κλάσης είναι αυτή που φαίνεται στην Εικόνα 23.

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:sport_masters_v2/models/podcast.dart';
import 'package:sport_masters_v2/models/posts.dart';

class DatabaseService {

  //Collection Reference
  final CollectionReference newsCollection = Firestore.instance.collection('newsposts');
  final CollectionReference podcastsCollection = Firestore.instance.collection('podcasts');
  final CollectionReference bugsCollection = Firestore.instance.collection('bugs');

  List<Posts> _postsListFromSnapshot(QuerySnapshot snapshot) {
    return snapshot.documents.map((doc) {
      print(doc.data['content_images']);
      return Posts(
        category: doc.data['category'] ?? '',
        title: doc.data['title'] ?? '',
        description: doc.data['description'] ?? '',
        previewImage: doc.data['image'] ?? '',
        mainContent: doc.data['content'] ?? '',
        images: List<String>.from(doc.data['content_images'] ?? [])
      ); // Posts
    }).toList();
  }
  List<Podcast> _podcastsListFromSnapshot(QuerySnapshot snapshot) {
    return snapshot.documents.map((doc) {
      return Podcast(
        title: doc.data['title'] ?? '',
        audioUrl: doc.data['url'] ?? '',
        image: doc.data['image'] ?? ''
      );
    }).toList();
  }

  Stream<List<Podcast>> get podcasts {
    return podcastsCollection.snapshots().map(_podcastsListFromSnapshot);
  }

  Stream<List<Posts>> get posts {
    return newsCollection.snapshots().map(_postsListFromSnapshot);
  }

  Future addBugReport(String title, String description) async {
    return await bugsCollection.document().setData({
      'title': title,
      'description': description
    });
  }
}
```

Εικόνα 23: Κώδικας κλάσης DatabaseService

Στην τρέχουσα κλάση χρειαστήκαμε να χρησιμοποιήσουμε τα περιεχόμενα των κλάσεων cloud_firestore.dart, podcast.dart και posts.dart. Για να μπορέσουμε να έχουμε πρόσβαση από την μία κλάση στην άλλη χρησιμοποιούμε πριν τον ορισμό της κλάσης την εντολή import και ως τιμή ορίζουμε το μονοπάτι μέσα στο Project για το αντίστοιχο αρχείο στο οποίο θέλουμε να έχουμε πρόσβαση. Ως προς τις παραμέτρους της κλάσης, υπάρχουν οι εξής:

- newsCollection: Μια Final μεταβλητή τύπου CollectionReference. Η μεταβλητή ορίστηκε ως Final γιατί με την έναρξη της εφαρμογής δεν θα αλλάξει ξανά εφόσον αποτελεί αντικείμενο CollectionReference. Το αντικείμενο CollectionReference είναι κομμάτι του Cloud Firestore Package. Το αντικείμενο αυτό μπορεί να χρησιμοποιηθεί για την προσθήκη εγγράφων στο Cloud Firestore, για την ανάκτηση αναφορών σε έγγραφα, όπως και στην δικιά μας περίπτωση και την αναζήτηση εγγράφων. Για τιμή, αυτή η μεταβλητή έχει ένα CollectionReference αντικείμενο το οποίο δέχεται ως παράμετρο το όνομα του Collection στο οποίο θα δημιουργηθεί το Reference σε String μορφή. Για να υπάρξει πρόσβαση σε αυτή τη μέθοδο χρησιμοποιήθηκε ένα στιγμιότυπο του αντικειμένου Firestore. Το αντικείμενο Firestore αποτελεί το σημείο εισόδου για το Cloud Firestore. Μέσα στην κλάση αυτή υπάρχει ένα σετ μεθόδων εκ των οποίων μια από αυτές είναι η collection η οποία εκτελεί αυτό το οποίο

προαναφέρθηκε. Ο σκοπός της είναι να επιστρέψει μια αναφορά του Collection το οποίο περίεχει τα άρθρα με τα νέα.

- podcastsCollection: Πανομοιότυπη με την μεταβλητή newsCollection η podcastsCollection είναι μια Final μεταβλητή τύπου CollectionReference. Χρησιμοποιείται για να επιστρέψει μια αναφορά του Collection το οποίο περιέχει τα Podcasts.

- bugsCollection: Πανομοιότυπη με την μεταβλητή newsCollection η bugsCollection είναι μια Final μεταβλητή τύπου CollectionReference. Χρησιμοποιείται για να επιστρέψει μια αναφορά του Collection το οποίο περιέχει τα Bugs.

Μετά τις παραμέτρους ακολουθούν οι ορισμοί των μεθόδων. Οι μέθοδοι που υπάρχουν είναι:

- _postsListFromSnapshot: Η συγκεκριμένη μέθοδος δέχεται ως παράμετρο ένα αντικείμενο τύπου QuerySnapshot και επιστρέφει μια λίστα αντικειμένων τύπου Posts. Ένα QuerySnapshot αντικείμενο επιστρέφει τα αποτελέσματα ενός ερωτήματος. Τα αποτελέσματα αυτά είναι τύπου DocumentSnapshot. Τα DocumentSnapshot αντικείμενα περιέχουν δεδομένα από τα έγγραφα τα οποία βρίσκονται στο Cloud Firestore. Το σώμα της μεθόδου περιέχει μια εντολή η οποία επιστρέφει μια λίστα των Posts.

- _postsListFromSnapshot: Και αυτή η μέθοδος δέχεται ως παράμετρο ένα QuerySnapshot και επιστρέφει μια λίστα αλλά ο τύπος της είναι Podcast.

- podcasts (Getter): Getter μέθοδος η οποία επιστρέφει ένα Stream τύπου List<Podcast>. Αυτό το οποίο ουσιαστικά κάνει το σώμα της μεθόδου είναι ότι για κάθε στιγμιότυπο της συλλογής επιστρέφει

- posts (Getter): Getter μέθοδος η οποία επιστρέφει ένα Stream τύπου List<Posts>.

- addBugReport: Future μέθοδος η οποία δημιουργεί ένα νέο Document στο Collection των Bugs. Μια Future μέθοδος επιτρέπει την ασύγχρονη εκτέλεση κώδικα με σκοπό να ελευθερώσει Threads τα οποία δεν θα έπρεπε να περιμένουν για το αποτέλεσμα αυτής της μεθόδου ώστε να συνεχίσουν. Ένα παράδειγμα ενός τέτοιου thread είναι το UI Thread το οποίο δεν πρέπει να περιλαμβάνει τέτοιου είδους μεθόδους καθώς θα υπάρξουν δυσλειτουργίες και προβλήματα.

Συνεχίζοντας, το αρχείο authentication.dart περιλαμβάνει τον πηγαίο κώδικα για την αυθεντικοποίηση των χρηστών της εφαρμογής. Η δομή της κλάσης είναι αυτή που φαίνεται στην Εικόνα 24.

```dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:sport_masters_v2/models/user.dart';
import 'package:sport_masters_v2/services/database.dart';

class AuthService {

  final FirebaseAuth _auth = FirebaseAuth.instance;

  User _userFromFirebaseUser(FirebaseUser user) {
    return user != null ? User(uid: user.uid) : null;
  }

  // Sign In with Email & Password
  Future signInWithEmailAndPassword(String email, String password) async {
    try {
      AuthResult result = await _auth.signInWithEmailAndPassword(email: email, password: password);
      FirebaseUser user = result.user;

      return _userFromFirebaseUser(user);
    } catch(e) {
      print(e.toString());
      return null;
    }
  }

  // Register with Email & Password
  Future registerWithEmailAndPassword(String email, String password) async {
    try {
      AuthResult result = await _auth.createUserWithEmailAndPassword(email: email, password: password);
      FirebaseUser user = result.user;

      await DatabaseService(uid: user.uid).updateUserData('title1', 'description1', 'image1');
      return _userFromFirebaseUser(user);
    } catch(e) {
      print(e.toString());
      return null;
    }
  }

  // Sign Out
  Future signOut() async {
    try {
      return await _auth.signOut();
    } catch(e) {
      print(e.toString());
      return null;
    }
  }

}
```

Εικόνα 24: Κώδικας κλάσης AuthService

Ως προς τις παραμέτρους υπάρχει μόνο μία final παράμετρος τύπου FirebaseAuth. Το αντικείμενο αυτό προέρχεται από το Package Firebase Auth το οποίο είχαμε εγκαταστήσει στην αρχή. Η κλάση αυτή αποτελεί τον τρόπο για να αποκτήσουμε πρόσβαση στο Firebase Authentication SDK. Οι μέθοδοι της κλάσης αποτελούνται από τις εξής:

- _userFromFirebaseUser: Η μέθοδος αυτή είναι τύπου Stream και επιστρέφει ένα αντικείμενο τύπου User κάθε φορά που γίνεται είσοδος ή έξοδος από την εφαρμογή.

- _userFromFirebaseUser: Η μέθοδος δέχεται για παράμετρο ένα αντικείμενο τύπου FirebaseUser και επιστρέφει έναν χρήστη τύπου User.

- signInWithEmailAndPassword: Η μέθοδος αυτή είναι τύπου Future και πραγματοποιεί την είσοδο του χρήστη στο Firebase. Δέχεται για είσοδο το email και τον κωδικό του χρήστη και υπάρχουν δύο περιπτώσεις: Είτε η ταυτοποίηση είναι επιτυχής οπότε επιστρέφεται το αντικείμενο του χρήστη που έκανε επιτυχής είσοδο τύπου User είτε υπάρχει κάποια εξαίρεση και η μέθοδος επιστρέφει null.

- RegisterWithEmailAndPassword: Η μέθοδος είναι τύπου Future και πραγματοποιεί εγγραφή στο Firebase δέχοντας για είσοδο το email και τον κωδικό του χρήστη. Υπάρχουν δύο περιπτώσεις. Είτε η δημιουργία λογαριασμού είναι επιτυχής και επιστρέφεται το αντικείμενο με τον χρήστη που ήδη δημιουργήθηκε είτε υπάρχει εξαίρεση και επιστρέφεται null.

- signOut: Μέθοδος τύπου Future η οποία πραγματοποιεί την αποσύνδεση του χρήστη από το Firebase και από την εφαρμογή.

Οι μέθοδοι που δημιουργήθηκαν είναι οι απαραίτητες για την εκτέλεση των βασικών λειτουργιών μιας τυπικής Mobile εφαρμογής. Όπως θα μπορούσε να παρατηρήσει κάποιος λειτουργίες οι οποίες θα μπορούσαν να πάρουν πολύ καιρό για να γίνουν από το μηδέν με το έτοιμα SDK για Firebase και Authentication αν κάποιος ξέρει κάποια βασικά πράγματα γύρω από Flutter μπορεί να κάνει την ίδια δουλειά σε λίγα μόλις λεπτά.

3.7.3 Main.dart

Το αρχείο main.dart είναι το μέρος στο οποίο ζεί ο κώδικας της εφαρμογής μας. Το περιεχόμενο της κλάσης φαίνεται στην Εικόνα 25.

```dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:splashscreen/splashscreen.dart';
import 'package:sport_masters_v2/screens/wrapper.dart';
import 'package:sport_masters_v2/services/authentication.dart';
import 'models/user.dart';

Run | Debug
void main() => runApp(MyApp());

class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return StreamProvider<User>.value(
      value: AuthService().user,
      child: MaterialApp(
        theme: ThemeData(
          textTheme: Typography.blackCupertino,
          primarySwatch: Colors.blue,
          brightness: Brightness.light
        ), // ThemeData
        home: SplashScreen(
          photoSize: 150.0,
          image: Image.asset(
            "assets/images/sport_masters_logo.jpg"
          ), // Image.asset
          backgroundColor: Color(0xff262626),
          navigateAfterSeconds: Wrapper(),
          seconds: 3,
          loaderColor: Colors.white,
        ) // SplashScreen
      ), // MaterialApp
    ); // StreamProvider.value
  }
}
```

Εικόνα 25: Περιεχόμενα αρχείου main.dart

Στην αρχή του αρχείου περιλαμβάνονται τα imports των εξής αρχείων:

- material.dart: Βιβλιοθήκη του Flutter η οποία περιέχει Widgets τα οποία υλοποιούνται βάσει Material Design.

- Provider.dart: Package το οποίο είχαμε εγκαταστήσει στην αρχή. Θα γίνει ανάλυση παρακάτω του ποια είναι η χρήση του.
- splashscreen.dart: Package το οποίο περιέχει το Widget που θα χρησιμοποιήσουμε ως Splashscreen στην εφαρμογή μας
- wrapper.dart: Κλάση που χρειαζόμαστε για να μπορέσουμε να προχωρήσουμε στην εφαρμογή μετά το Splash. Θα γίνει εξήγηση της χρήσης της Wrapper κλάσης στην συνέχεια
- authentication.dart: Η κλάση που δημιουργήσαμε και περιέχει τις μεθόδους αυθεντικοποίησης
- user.dart: Το μοντέλο του χρήστη.

Πριν την δημιουργία της κλάσης η πρώτη γραμμή που εμφανίζεται είναι η "void main() => runApp(MyApp());" . Αν διαβάσουμε απλά την εντολή αυτό σημαίνει ότι η μέθοδος main θα τρέξει μια εφαρμογή με το όνομα MyApp. Όπως και με άλλες γλώσσες προγραμματισμού η μέθοδος main εδώ αποτελεί το σημείο έναρξης της εφαρμογής.

Η κλάση που ορίζεται κάτω από την εντολή, είναι η MyApp. Η κλάση MyApp αποτελεί ένα Stateful Widget. Για να καταλάβουμε τι είναι ένα Stateful Widget, θα πρέπει πρώτα να δούμε τι ορίζεται ως State. Το State λοιπόν είναι πληροφορία η οποία μπορεί να διαβαστεί συγχρόνως με το build ενός Widget και μπορεί να αλλάξει κατά την διάρκεια ζωής ενός Widget.

Ένα Stateful Widget λοιπόν είναι ένα Widget το οποίο περιγράφει ένα κομμάτι της διεπαφής του χρήστη χτίζοντας μια πλειάδα Widget τα οποία περιγράφουν την διεπαφή με πιο σαφή τρόπο. Τα Stateful Widgets είναι χρήσιμα όταν κομμάτι της διεπαφής του χρήστη που περιγράφεται μπορεί να αλλάξει δυναμικά. Τα Stateful Widgets είναι immutable δηλαδή η κατάσταση τους δεν μπορεί να αλλάξει, και αποθηκεύουν την Mutable (μπορεί να τροποποιηθεί) State τους σε άλλα State αντικείμενα τα οποία δημιουργούνται από την μέθοδο createState. Για να μπορέσουν να γίνουν build επομένως τα widgets χρειάζεται να γίνει η δημιουργία της μεθόδου build. Η μέθοδος build είναι υπεύθυνη για το Render του αντίστοιχου Widget στου οποίου το State βρίσκεται. Το framework καλεί την μέθοδο όταν το Widget εισάγεται μέσα στο δέντρο των Widgets τα οποία αποτελούν την εφαρμογή σε ένα συγκεκριμένο BuildContext. Το BuildContext είναι ένα αντικείμενο το οποίο δείχνει την θέση ενός Widget στο Widget Tree.

Στον κώδικα του τρέχοντος Project η μέθοδος build επιστρέφει ένα αντικείμενο τύπου Widget.
Συγκεκριμένα επιστρέφει ένα αντικείμενο τύπου StreamProvider. Το αντικείμενο αυτό ακούει ένα Stream και επιστρέφει τα περιεχόμενα του στο Widget το οποίο έχει οριστεί ως child καθώς και σε όλα τα υπόλοιπα Widgets τα οποία βρίσκονται κάτω από αυτό. Το βασικό σενάριο εφαρμογής αυτού του αντικειμένου είναι η παροχή του περιεχομένου του Stream, όπως ένα Firebase ερώτημα στα
Widgets κάτω από αυτό χωρίς να χρειαστεί να αντιδράσει αναλόγως στην περίπτωση κάποιου Event. Δίπλα από τον ορισμό του αντικειμένου υπάρχουν και οι αγκύλες για τα Generics. Στην συγκεκριμένη περίπτωση δεν ορίζουμε ότι το αντικείμενο είναι Generic αλλά καθορίζουμε τον ακριβή τύπο του. Στην περίπτωση μας είναι User. Αυτό γίνεται γιατί η τιμή που επιστρέφεται είναι ίδια με τον τύπο της τιμής που ορίζεται στην παράμετρο Value που εδώ είναι τύπου User καθώς η μέθοδος που έχουμε θέσει και ορίσει είναι αυτή που είχαμε πεί ότι είναι Getter τύπου User. Έτσι έχουμε διασφαλίσει ότι με την εκτέλεση του αντίστοιχου αιτήματος όλα τα Widgets τα οποία βρίσκονται μέσα στο πεδίο child θα έχουμε πρόσβαση στο Response του αιτήματος.

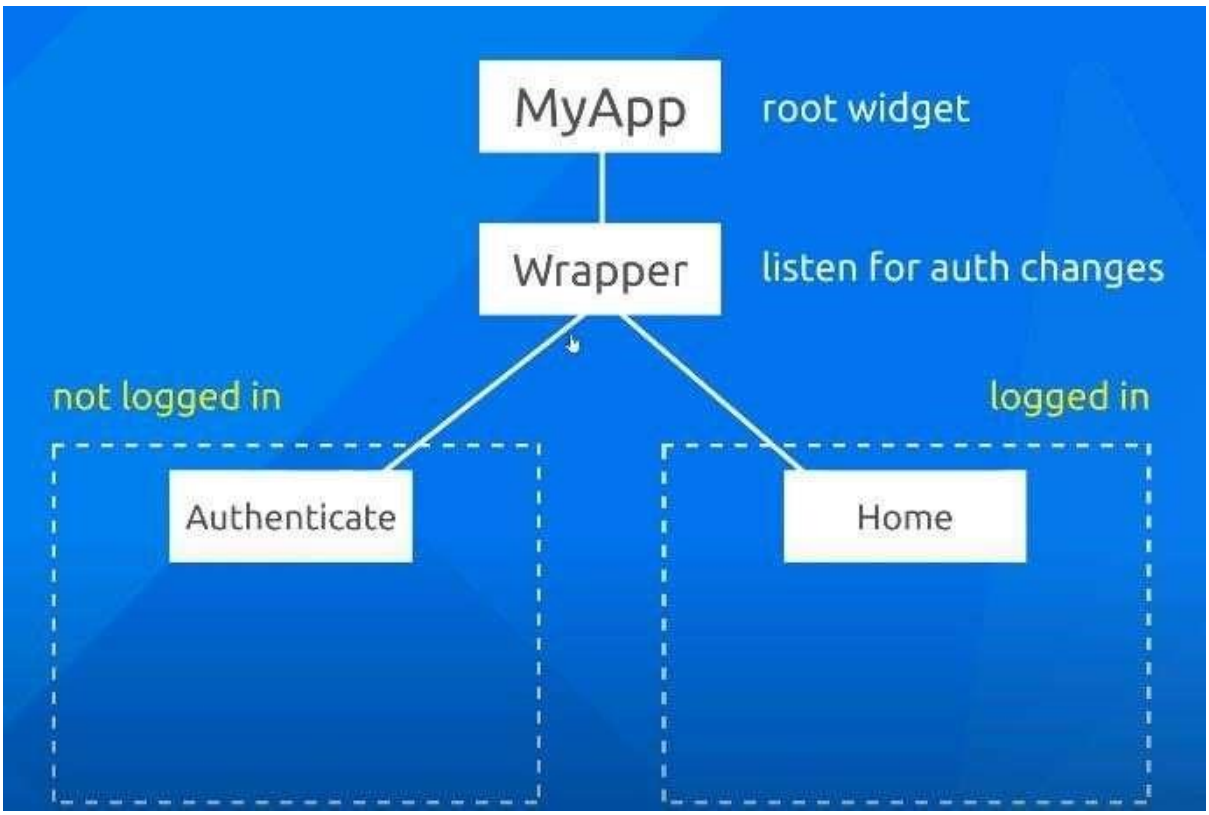
Όσον αφορά το child αποτελείται αρχικά από ένα MaterialApp. Το MaterialApp είναι μια εφαρμογή η οποία χρησιμοποιεί Material Design. Το Material App ρυθμίζει πολλές παραμέτρους που αφορούν την διεπαφή της εφαρμογής ως προς την ολότητα της. Στην περίπτωση μας έχουμε ορίσει τις παραμέτρους:

- home: Το Default Widget το οποίο θα εμφανιστεί. Στην περίπτωση μας είναι το SplashScreen. Το Widget αυτό θα χρησιμοποιηθεί ως το Splash Screen της εφαρμογής. Οι παράμετροι που ορίστηκαν σε αυτό το Widget είναι:
    - photoSize: Το μέγεθος της εικόνας που απεικονίζεται στην οθόνη. ο image: Η εικόνα που θα εμφανίζεται στην οθόνη. Όσον αφορά την εικόνα γίνεται χρήση της μεθόδου Image.asset η οποία δέχεται ως παράμετρο το μονοπάτι του αρχείου που θα απεικονιστεί σε μορφή ImageStream και αν αυτό υπάρχει απεικονίζει την εικόνα. Είναι σημαντικό να διευκρινίσουμε ότι για να μπορέσει να λειτουργήσει το Path πρέπει να οριστεί και σαν Asset στο αρχείο pubspec.yaml κάτι το οποίο εμείς έχουμε κάνει νωρίτερα.
    - backgroundColor: Το χρώμα του Background της οθόνης
    - navigateAfterSeconds: Η παράμετρος αυτή ορίζει το Widget στο οποίο θα κάνει Navigate η εφαρμογή μετά από τον αριθμό των δευτερολέπτων που έχουν οριστεί από την παράμετρο seconds.
    - seconds: Ορίζει τα δευτερόλεπτα που θα χρειαστεί να περάσουν για να εκτελεστεί η οδηγία που θα δωθεί από την παράμετρο navigationAfterSeconds.
    - loaderColor: Το χρώμα του Loader της οθόνης

Έχοντας ορίσει λοιπόν το Widget το οποίο θα τοποθετηθεί πρώτο στο δέντρο των Widgets και αφού ορίσαμε το Widget Material App το οποίο θα αποτελέσει το πλαίσιο για να χτίσουμε το υπόλοιπο περιεχόμενο της εφαρμογής, πρέπει να δούμε τι ακριβώς είναι η κλάση Wrapper και ποιος είναι ο σκοπός της.

3.7.4 Wrapper

Όταν η εφαρμογή ξεκινάει βασίζεται σε έναν σημαντικό παράγοντα. Την κατάσταση σύνδεσης του χρήστη. Όταν ο χρήστης ξεκινάει την εφαρμογή για πρώτη φορά, δεν έχει κάνει κανένα είδος ταυτοποίησης ή εγγραφής, επομένως η εφαρμογή δεν έχει καμία πληροφορία σχετική με το ποιος είναι. Από την άλλη όταν ο χρήστης είτε έχει κάνει εγγραφή, είτε έχει συνδεθεί στο Firebase, έχει με κάποιον τρόπο αλληλεπιδράσει με την εφαρμογή. Αυτές οι δύο περιπτώσεις που προαναφέρθηκαν, είναι δύο ξεχωριστά μονοπάτια και πρέπει να διαχειριστούν με κάποιον τρόπο. Εδώ έρχεται η κλάση Wrapper η οποία ουσιαστικά τυλίγει και τα δύο Widgets τα οποία ορίζουν δύο ξεχωριστά μονοπάτια τα οποία θα ακολουθήσει η εφαρμογή για να συνεχίσει και μέσω αυτής γίνεται έλεγχος της κατάστασης στην οποία βρίσκεται ο χρήστης και ανάλογα εκτελούνται οι αντίστοιχες ενέργειες.



Εικόνα 26: Διάγραμμα Δομής Wrapper

Όσον αφορά τον κώδικα η δομή του Project είναι αυτή που φαίνεται στην παρακάτω εικόνα:

```dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sport_masters_v2/screens/homepage/homepage.dart';
import 'package:sport_masters_v2/screens/login/login.dart';
import 'package:sport_masters_v2/models/user.dart';

class Wrapper extends StatefulWidget {
  @override
  _WrapperState createState() => _WrapperState();
}

class _WrapperState extends State<Wrapper> {

  @override
  Widget build(BuildContext context) {
    final user = Provider.of<User>(context);
    print(user);

    if (user == null) {

      return Login();
    } else {

      setState(() {
      });

      return HomePage();
    }
  }
}
```

Εικόνα 27: Κώδικας αρχείο Wrapper.dart

Η κλάση Wrapper όπως και η MyApp είναι ένα StatefulWidget του οποίου η λειτουργία είναι αυτή που εξηγήθηκε νωρίτερα. Το State αντικείμενο της κλάσης περιλαμβάνει την μέθοδο build μέσα στην οποία ορίζεται μια final μεταβλητή τύπου user η οποία ισούται με Provider.of<user>(context). Το αντικείμενο στην ουσία με το οποίο ισούται είναι τύπου Provider. Το αντικείμενο είναι υπεύθυνο να διαχειρίζεται τον κύκλο ζωής της τιμής την οποία παρέχει στα Widget τα οποία βρίσκονται κάτω από αυτό (Child Widgets). Στην συγκεκριμένη γραμμή γίνεται χρήση της μεθόδου of η οποία είναι μια Generic μέθοδος. Αυτό το οποίο κάνει αυτή η μέθοδος είναι ότι βρίσκει τον πιο κοντινό Provider πάνω από αυτήν και τραβάει την τιμή του. Επειδή έχουμε ορίσει με τις αγκύλες ότι ο τύπος θα είναι User η μέθοδος θα ψάξει για τιμή η οποία είναι τύπου User. Έτσι, με βάση και τον κώδικα της main μπορούμε να συμπεράνουμε ότι ουσιαστικά ο λόγος που γίνεται αυτή η διαδικασία είναι για να ελεγχθεί η κατάσταση του χρήστη, αν δηλαδή έχει ταυτοποιηθεί ή είναι καινούργιος στην εφαρμογή. Το περιεχόμενο της μεταβλητής ελέγχεται με μια συνθήκη όπου γίνεται η ερώτηση αν η μεταβλητή είναι null ή όχι και ανάλογα το αποτέλεσμα στην μέθοδο build το Widget το οποίο επιστρέφεται για να γίνει Render είναι είτε το Login, είτε το HomePage.

Έχοντας εξασφαλίσει έλεγχο για την έγκυρη αυθεντικοποίηση του χρήστη προχωράμε στην ανάλυση του πρώτο μονοπατιού της εφαρμογής το οποίο είναι η είσοδος.

3.7.5 Login

Το πρώτο βήμα ενός νέου ή εγγεγραμμένου χρήστη θα είναι να κάνει είσοδο στην εφαρμογή. Σε αυτήν την περίπτωση η εφαρμογή εμφανίζει το Widget Login. Το Widget Login είναι υπεύθυνο να εμφανίσει στον χρήστη την κατάλληλη φόρμα για ταυτοποίηση, καθώς και τις αντίστοιχες επιλογές σε περίπτωση που αυτή η περίπτωση δεν είναι αυτό το οποίο επιθυμεί να εκπληρώσει. Ο κώδικας του Widget είναι αυτός ο οποίος φαίνεται στις παρακάτω εικόνες.

```dart
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:sport_masters_v2/screens/register/register.dart';
import 'package:sport_masters_v2/services/authentication.dart';
import 'package:sport_masters_v2/shared/loading.dart';

class Login extends StatefulWidget {
  @override
  _LoginState createState() => _LoginState();
}

class _LoginState extends State<Login> {

  AuthService _auth = AuthService();
  final _formKey = GlobalKey<FormState>();
  bool loading = false;

  // Text Field State
  String email = '';
  String password = '';
```

Εικόνα 28: Κώδικας login.dart (1)

```dart
@override
Widget build(BuildContext context) {
  return loading ? Loading() : Scaffold(
    body: Stack(
      children: <Widget>[
        Container(
          height: double.infinity,
          width: double.infinity,
          decoration: BoxDecoration(
            color: Color(0xFF262626)
          ), // BoxDecoration
        ), // Container
        Container(
          height: double.infinity,
          child: SingleChildScrollView(
            physics: AlwaysScrollableScrollPhysics(),
            padding: EdgeInsets.symmetric(
              horizontal: 40.0,
              vertical: MediaQuery.of(context).size.height / 10,
            ), // EdgeInsets.symmetric
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: <Widget>[
                Text(
                  'SPORT MASTERS',
                  style: TextStyle(
                    color: Colors.white,
                    fontFamily: 'Sports Jersey',
                    fontSize: 40.0,
                  ), // TextStyle
                ), // Text
```

Εικόνα 29: Κώδικας login.dart (3)

```dart
SizedBox(height: MediaQuery.of(context).size.height / 20),
Form(
  key: _formKey,
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
      Container(
        height: MediaQuery.of(context).size.height / 13,
        alignment: Alignment.centerLeft,
        child: TextFormField(
          style: TextStyle(
            color: Colors.white
          ), // TextStyle
          onChanged: (val) {
            email = val;
          },
          decoration: InputDecoration(
            prefixIcon: Icon(
              Icons.account_box,
              color: Colors.white,
            ), // Icon
            filled: true,
            fillColor: Colors.transparent,
            hintText: 'Email',
            hintStyle: TextStyle(color: Colors.white),
            enabledBorder: OutlineInputBorder(
              borderRadius: BorderRadius.all(Radius.circular(10)),
              borderSide: BorderSide(width: 2, color: Colors.white)
            ), // OutlineInputBorder
          ) // InputDecoration
        ) // TextFormField
      ), // Container
      SizedBox(height: MediaQuery.of(context).size.height / 22),
      Container(
        height: MediaQuery.of(context).size.height / 13,
        alignment: Alignment.centerLeft,
        child: TextFormField(
          obscureText: true,
          style: TextStyle(
            color: Colors.white
          ), // TextStyle
          onChanged: (val) {
            password = val;
          },
          decoration: InputDecoration(
            prefixIcon: Icon(
              Icons.lock,
              color: Colors.white,
            ), // Icon
            filled: true,
            fillColor: Colors.transparent,
            hintText: 'Κωδικός Χρήστη',
            hintStyle: TextStyle(color: Colors.white),
            enabledBorder: OutlineInputBorder(
              borderRadius: BorderRadius.all(Radius.circular(10)),
              borderSide: BorderSide(width: 2, color: Colors.white)
            ), // OutlineInputBorder
          ) // InputDecoration
        ) // TextFormField
      ), // Container
```

Εικόνα 30: Κώδικας login.dart (2)

Το Widget αυτό αρχικά είναι ένα Stateful Widget και αυτό γιατί βασικό κριτήριο χρήσης ενός Stateful Widget είναι ότι τα δεδομένα τα οποία διαχειρίζεται αλλάζουν και στην περίπτωση μας αυτό ισχύει. Ένα παράδειγμα αλλαγής είναι η συνεχής αλλαγή της τιμής των πεδίων ονόματος χρήστη και κωδικού με την κάθε φορά που ο χρήστης θα πληκτρολογεί έναν χαρακτήρα. Οι παράμετροι του Widget είναι:

- _auth: Αντικείμενο τύπου AuthService. Με αυτό το αντικείμενο θα γίνει η διαδικασία ταυτοποίησης του χρήστη χρησιμοποιώντας το email και κωδικό το οποίο έχει πληκτρολογήσει στα αντίστοιχα πεδία.

- _formKey: Final μεταβλητή τύπου GlobalKey. Όταν ορίζεται ένα Widget τύπου Form συνίσταται να δημιουργείται και ένα GlobalKey και ο λόγος είναι διότι μέσω αυτού του αντικειμένου θα γίνει επαλήθευση των τιμών των πεδίων.

- loading: bool μεταβλητή η οποία χρησιμοποιείται για τον έλεγχο εμφάνισης Widget. Θα εξηγηθεί παρακάτω πως ακριβώς χρησιμοποιείται

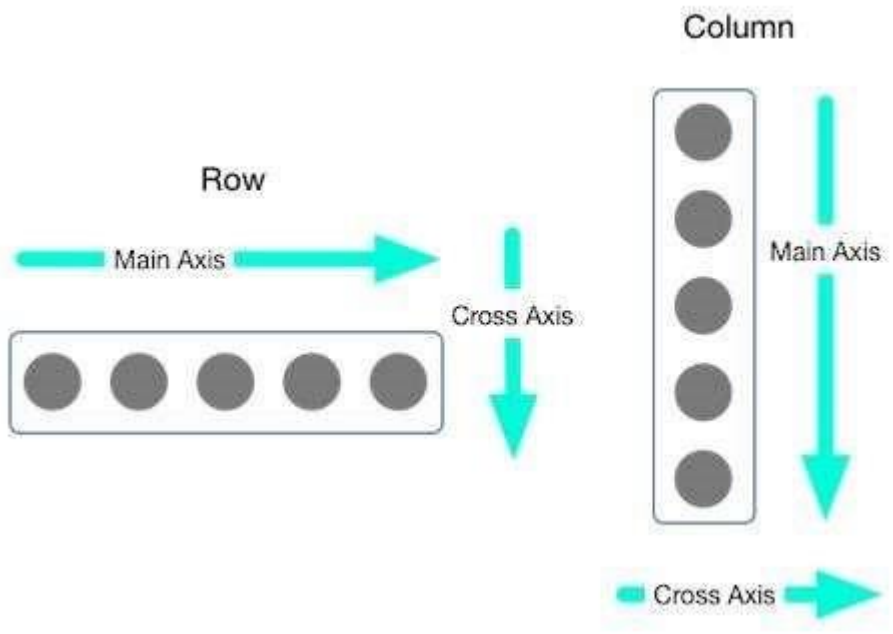
- email: String μεταβλητή στην οποία θα αποθηκεύεται το περιεχόμενο του πεδίου για το email.

- password: String μεταβλητή στην οποία θα αποθηκεύεται το περιεχόμενο του πεδίου για το password.

Στην συνέχεια όσον αφορά μεθόδους υπάρχουν:

- build: Η build στην συγκεκριμένη περίπτωση αποτελείται από μία πλειάδα Widgets. Αρχίζοντας από την πρώτη γραμμή κώδικα βλέπουμε την γραμμή “return loading ? Loading() : Scaffold(…). Η γραμμή αυτή ουσιαστικά είναι μια if-else συνθήκη συμπιεσμένη σε μία γραμμή. Αυτό το οποίο ουσιαστικά ελέγχεται είναι ότι αν το περιεχόμενο της μεταβλητής Loading είναι true τότε εμφάνισε το Widget Loading το οποίο έχουμε ορίσει ως έναν Loader της εφαρμογής σε περίπτωση που χρειάζεται χρόνος για την επιστροφή ενός αποτελέσματος από ένα αίτημα, ενώ αν είναι False εμφάνισε το αντίστοιχο περιεχόμενο του Login Widget. Όσον αφορά το περιεχόμενο, αποτελείται αρχικά από ένα Scaffold. Ένα Scaffold Widget υλοποιεί την βασική δομή ενός Material Design App και αποτελεί το πλαίσιο για να μπούν τα υπόλοιπα Widgets. Ως σώμα αυτού του Widget ορίζεται στην συνέχεια ένα Stack Widget. Ένα Stack Widget ορίζει ένα κουτί στο οποίο αυτό έχει τον έλεγχο και δημιουργεί μια στοίβα αντικειμένων τα οποία έχουν ως δικό τους χώρο αυτό το κουτί και κάθονται το ένα πάνω στο άλλο. Ο πίνακας αυτών των Widget αποτελείται αρχικά από ένα Container. Ένα Container Widget χρησιμοποιείται βασικό βάψιμο, τοποθέτηση και ορισμό του μεγέθους των Widgets. Στην περίπτωση μας αυτό το Container αποτελεί ουσιαστικά το βασικό χρώμα της οθόνης γιαυτό και οι παράμετροι height και width έχουν οριστεί ως infinity double που ουσιαστικά σημαίνει όσο μεγάλο όσο με αφήνει το Widget γονέας που στην συγκεκριμένη περίπτωση είναι το Scaffold άρα πιάνουμε όλη την επιφάνεια του Scaffold. Έπειτα ορίζουμε ένα ακόμη Container με ύψος όσο επιτρέπει το Scaffold και με παιδί ένα SingleChildScrollView. Το βασικό χαρακτηριστικό ενός SingleChildScrollView είναι ότι το Widget το οποίο ορίζεται ως child του είναι Scrollable, δηλαδή μπορεί ο χρήστης να κάνει Scroll. Ο λόγος που το κάναμε έτσι είναι διότι όταν ο χρήστης ανοίξει το πληκτρολόγιο ένα κομμάτι της οθόνης θα χαθεί με αποτέλεσμα αν δεν μπορεί να γίνει Scroll του Widget ο χρήστης να μην μπορεί και να το δεί

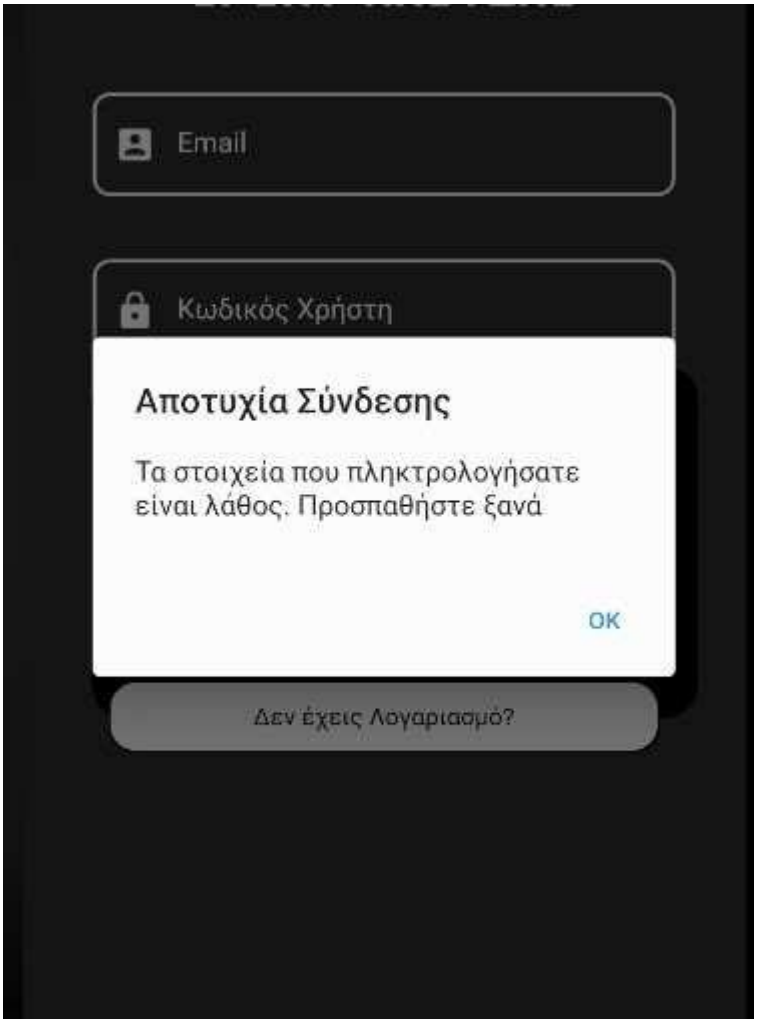
αν το έχει ανοιχτό. Παιδί αυτού του Widget αποτελεί ένα Column ή μια στήλη. Όπως λέει και το όνομα του το Widget αυτό είναι ένα Widget το οποίο έχει μια συγκεκριμένη μορφοποίηση για να θέτει την θέση των αντικειμένων σαν μια στήλη. Η παράμετρος mainAxisAlignment ορίζει την θέση των Widgets στον κάθετο άξονα.



Εικόνα 31: Απεικόνιση της κατεύθυνσης του Main και Cross Axis

Η τιμή MainAxisAlignment.center ορίζει ότι τα Widgets θα τοποθετηθούν έτσι ώστε να βρίσκονται στο κέντρο της στήλης. Ακολουθούν τα Widgets παιδιά του Column τα οποία είναι αρχικά ένα Text Widget το οποίο έχει ένα συγκεκριμένο style. Μέσα σε αυτά το σημαντικό είναι το fontFamily και ο λόγος είναι διότι το Family αυτό δεν είναι imported από το Flutter αλλά είναι το Custom Font το οποίο δηλώσαμε εμείς στο αρχείο pubspec.yaml. Παρακάτω βλέπουμε ένα SizedBox Widget. Το συγκεκριμένο Widget εδώ χρησιμοποιήθηκε για να δημιουργήσει ένα Custom Spacing ανάμεσα στο Text και στο Form Widget. Όσον αφορά για το Form Widget χρησιμοποιείται για την ομαδοποίηση των Text Fields που αποτελούν το κομμάτι μιας φόρμας. Στην παράμετρο Form θα θέσουμε ως key αυτής της φόρμας την μεταβλητή _formKey το οποίο θα χρησιμοποιήσουμε για Validation. Η φόρμα περιέχει δύο TextFormField, τα οποία ορίζονται για να μπορέσουμε να ανακτήσουμε τα δεδομένα μέσω της φόρμας, ένα για το όνομα και ένα για τον κωδικό. Το TextFormField παρέχει την δυνατότητα ανάκτησης του περιεχομένου κάθε φορά που το περιεχόμενο αλλάζει με την χρήση της παραμέτρου onChanged. Μέσα σε αυτήν την callback μέθοδο, η οποία μας επιστρέφει την τιμή του πεδίου αναθέτουμε το περιεχόμενο των πεδίων στις αντίστοιχες μεταβλητές email και password για τα αντίστοιχα πεδία. Έτσι έχουμε αποθηκευμένο το περιεχόμενο στις μεταβλητές για να μπορέσουμε να το χειριστούμε αναλόγως. Επόμενο κομμάτι είναι το κουμπί της εισόδου. Αυτό είναι τύπου RaisedButton. Κάθε button έχει την παράμετρο onPressed η οποία εκτελείται κάθε φορά που ο χρήστης κάνει Tap στο κουμπί και μέσα σε αυτή την παράμετρο μπορεί να εκτελεστεί κώδικας. Στην περίπτωση μας αυτό το οποίο θα κάνουμε είναι να εκτελέσουμε την ταυτοποίηση. Για αυτό θα χρειαστούμε την παράμετρο _auth την οποία δημιουργήσαμε νωρίτερα. Η μέθοδος για να μην επιβαρύνει το Thread του UI προφανώς και θα γίνει ασύγχρονα, για αυτό χρησιμοποιούμε την λέξη async πριν το σώμα της μεθόδου. Αρχικά

γίνεται ένας έλεγχος των πεδίων της φόρμας για το αν είναι γεμάτα. Αν η συνθήκη αυτή είναι αληθής τότε η γίνεται set του State που σημαίνει ότι το UI θα ξαναχτιστεί, η μεταβλητή loading αποκτά τιμή true και εκτελείται η γραμμή "dynamic result = await _auth.signInWithEmailAndPassword(email, password);" Η οποία ουσιαστικά επιστρέφει το αποτέλεσμα της εισόδου του χρήστη στο Firebase. Έπειτα γίνεται έλεγχος του result. Αν αυτό είναι null τότε εμφανίζεται ένα Alert Dialog. Τα Alert Dialogs είναι Widgets τα οποία χρησιμοποιούνται για την ενημέρωση του χρήστη και επι της ουσίας είναι Pop-up Dialogs. Για να μην χρειάζεται να γίνουν Custom Animations και διαδικασίες για την εμφάνιση του Dialog το Material Package παρέχει την μέθοδο showDialog η οποία αναλαμβάνει όλες αυτές τις διαδικασίες. Έτσι λοιπόν στην τρέχουσα περίπτωση εάν το result είναι null τότε εμφανίζεται το εξής Alert Dialog:



Εικόνα 32: Popup αποτυχίας σύνδεσης

Αν από την άλλη το περιεχόμενο δεν είναι null αυτό σημαίνει ότι επιστράφηκε το αντικείμενο του χρήστη οπότε η μεταβλητή loading ισούται με false. Αυτό σημαίνει ότι πλέον υπάρχει ταυτοποιημένος χρήστης στο αντικείμενο της Wrapper κλάσης. Άρα πλέον φεύγουμε από το Login σενάριο και μπαίνουμε στο Homepage.

Τελευταίο κομμάτι αποτελεί το κουμπί “Δεν έχεις Λογαριασμό?” το οποίο κάνοντας Tap πάνω μεταφερόμαστε στην οθόνη της εγγραφής δηλαδή στο Register Widget. Για να μεταφερθούμε χρησιμοποιείται το Navigator Widget. Το Widget αυτό είναι υπεύθυνο για τον χειρισμό Widgets ως μια στοίβα. Σε ένα Android ή iOS App πάντα υπάρχει ένας τρόπος για να μπορέσουμε να δρομολογηθούμε ανάμεσα στην διάφορες οθόνες χρησιμοποιώντας αντίστοιχους Navigators. Στο Flutter το Navigator Widget ουσιαστικά αποτελεί αυτόν τον τρόπο για προσανατολισμό στην εφαρμογή. Για να μπορέσουμε να αξιοποιήσουμε το Widget αυτό και να εμφανίσουμε το Widget το οποίο επιθυμούμε αρκεί να χρησιμοποιήσουμε την μέθοδο push. Αυτή η μέθοδος ουσιαστικά θα προσθέσει το Widget της επιλογής μας στο Stack του Navigator. Για να μπορέσουμε να εμφανίσουμε το Widget ομαλά το εμφωλεύουμε μέσα σε ένα MaterialPageRoute. Με το MaterialPageRoute ουσιαστικά αντικαθιστάται όλη η οθόνη με το Widget το οποίο θα ορίσουμε ότι θα χτιστεί στην παράμετρο builder με ένα συγκεκριμένο Transition. Το Transition είναι ανάλογο της πλατφόρμας στην οποία τρέχει η εφαρμογή, έτσι για iOS, το Route αντικείμενο θα γλιστρήσει από δεξιά προς τα αριστερά και αν ποτέ του δωθεί η εντολή να βγεί από την οθόνη, θα το κάνει εκτελώντας το Transition αυτό, αλλά ανάποδα. Έτσι λοιπόν με τις εντολές που έχουμε γράψει λέμε πως όταν πατήσει στο κουμπί της εγγραφής, ο χρήστης θα μεταφερθεί στην οθόνη της Εγγραφής.

3.7.6 Register

Επόμενη περίπτωση την οποία μπορεί να συναντήσει ο χρήστης είναι η δημιουργία ενός νέου λογαριασμού, αν δεν έχει κάποιον ήδη. Αυτήν την περίπτωση καλύπτει το Widget Register. Η μορφή του κώδικα του είναι αυτή στις παρακάτω εικόνες:

```dart
import 'package:flutter/material.dart';
import 'package:sport_masters_v2/services/authentication.dart';
import 'package:sport_masters_v2/shared/loading.dart';

class Register extends StatefulWidget {
  @override
  _RegisterState createState() => _RegisterState();
}

class _RegisterState extends State<Register> {

final AuthService _auth = AuthService();
  final _formKey = GlobalKey<FormState>();
  bool loading = false;

  TextEditingController emailController = TextEditingController();
  TextEditingController passwordController = TextEditingController();

  final GlobalKey<_RegisterInputBoxState> _emailTextFieldState = GlobalKey();
  final GlobalKey<_RegisterInputBoxState> _passwordTextFieldState = GlobalKey();
```

Εικόνα 33: Κώδικας register.dart (1)

```dart
@override
Widget build(BuildContext context) {

  String email = "";
  String password = "";

  return loading ? Loading() : SafeArea(
    child: GestureDetector(
      onTap: () {
        FocusScopeNode currentFocus = FocusScope.of(context);

        if (!currentFocus.hasPrimaryFocus) {
          currentFocus.unfocus();
        }
      },
      child: Scaffold(
        body: SingleChildScrollView(
          physics: BouncingScrollPhysics(),
          child: Stack(
            children: [
              Column(
                crossAxisAlignment: CrossAxisAlignment.center,
                children: [
                  Padding(
                    padding: const EdgeInsets.only(left: 30, top: 50),
                    child: Text(
                      'Δημιουργία Λογαριασμού',
                      style: TextStyle(
                        fontFamily: 'Roboto',
                        fontSize: 35,
                        fontWeight: FontWeight.w800
                      ), // TextStyle
                    ) // Text
                  ), // Padding
                  SizedBox(
                    height: 30
                  ), // SizedBox
                  Form(
                    key: _formKey,
                    child: Column(
                      children: [
                        RegisterInputBox(
                            key: _emailTextFieldState,
                            controller: emailController,
                            label: 'Email',
                            inputHint: 'example@example.com',
                            isPasswordField: false,
                        ), // RegisterInputBox
                        RegisterInputBox(
                          key: _passwordTextFieldState,
                          controller: passwordController,
                          label: 'Κωδικός Χρήστη',
                          inputHint: '',
                          isPasswordField: true,
                        ), // RegisterInputBox
                        SizedBox(
                          height: 30
                        ), // SizedBox
```

Εικόνα 34: Κώδικας register.dart (2)

```
Text(
  "Δημιουργώντας έναν νέο λογαριασμό συμφωνείς \nμε τους όρους χρήσης της εφαρμογής",
  textAlign: TextAlign.center,
  style: TextStyle(
    fontFamily: 'Roboto',
    fontSize: 15.5,
    fontWeight: FontWeight.bold,
    color: Color(0xff8f9db5).withOpacity(0.75),
  ), // TextStyle
), // Text
SizedBox(
  height: 50,
), // SizedBox
GestureDetector(
  onTap: () async {
    if(_formKey.currentState.validate()) {
      loading = true;

      email = _emailTextFieldState.currentState.emailTextFieldController.text;
      password = _passwordTextFieldState.currentState.passwordTextFieldController.text;

      dynamic result = _auth.registerWithEmailAndPassword(email, password);

      if (result == null) {
        showDialog(
            context: context,
            builder: (_) => AlertDialog(
              title: Text("Αποτυχία Εγγραφής"),
              content: Text("Κάτι πήγε στραβά. Προσπαθήστε ξανά"),
              actions: [FlatButton(
                onPressed: () {
                  Navigator.pop(context);
                  setState(() { loading = false; });
                }, child: Text("OK")
                ) // FlatButton
              ],
            ), // AlertDialog
            barrierDismissible: true
        );
      } else {
        loading = false;
        showDialog(
            context: context,
            builder: (_) => AlertDialog(
              title: Text("Επιτυχής Δημιουργία Λογαριασμού"),
              content: Text("Ο λογαριασμός σας δημιουργήθηκε επιτυχώς. Πατήστε OK για να συνεχίσετε στην εφαρμογή"),
              actions: [FlatButton(
                onPressed: () {
                  Navigator.of(context).pop();
                  Navigator.pop(context);
                  setState(() { loading = false; });
                }, child: Text("OK")
                ) // FlatButton
              ],
            ), // AlertDialog
            barrierDismissible: true
        );
      }
```

Εικόνα 35: Κώδικας register.dart (3)

```
child: Container(
  margin: EdgeInsets.symmetric(vertical: 20),
  width: MediaQuery.of(context).size.width * 0.85,
  height: 75,
  decoration: BoxDecoration(
    color: Color(0xff0962ff),
    borderRadius: BorderRadius.circular(20),
  ), // BoxDecoration
  child: Center(
    child: Text(
      'Δημιουργία νέου Λογαριασμού',
      style: TextStyle(
        fontFamily: 'ProductSans',
        fontSize: 18,
        fontWeight: FontWeight.bold,
        color: Colors.white
      ), // TextStyle
    ), // Text
  ), // Center
) // Container
```

Εικόνα 36: Κώδικας register.dart (4)

Το Register Widget είναι ένα Stateful Widget. Οι παράμετροι του είναι:

- _auth: AuthService αντικείμενο το οποίο αυτή τη φορά θα χρησιμοποιηθεί για την δημιουργία λογαριασμού.

- _formKey: GlobalKey τύπου FormState για να μπορέσουμε να κάνουμε Validation της φόρμας.

- loading: bool μεταβλητή η οποία θα έχει παρόμοια χρήση με αυτήν της Login καθώς βάσει αυτής θα εμφανίζονται τα αντίστοιχα Widgets.

Ως προς τις μεθόδους η κλάση έχει μόνο μία την build. Μέσα σε αυτήν περιλαμβάνονται δύο τοπικές μεταβλητές email και password οι οποίες θα χρησιμοποιηθούν για την κλήση της μεθόδου για αίτημα στο Firebase. Η μέθοδος επιστρέφει ανάλογα με το περιεχόμενο της μεταβλητής loading είτε ένα Loader είτε ένα SafeArea αντικείμενο. Το SafeArea αντικείμενο δημιουργείται καθώς υπάρχει η ανάγκη υπολογισμού του κομματιού της οθόνης η οποία μπορεί να αξιοποιηθεί. Σε συσκευές όπως iPhone X, όπου η οθόνη δεν είναι ένα τετράγωνο αλλά αποτελείται από πολλές γωνίες και η δομή της οθόνης περιλαμβάνει κιάλλα στοιχεία πέρα από την εφαρμογή, χρειάζεται να ορίζεται το σωστό κομμάτι της οθόνης το οποίο θα χρησιμοποιηθεί για να μπορέσει να λειτουργήσει σωστά ο τρόπος με τον οποίο εμφανίζονται τα Widgets. Εδώ είναι που χρησιμεύει το Widget SafeArea. Ως child περιέχει το GestureDetector Widget. Το Widget αυτό χρησιμοποιείται όταν χρειάζεται να αναγνωριστούν χειρονομίες. Η χειρονομία την οποία χειριζόμαστε στον κώδικα μας είναι το tap με την παράμετρο onTap και την χειριζόμαστε έτσι ώστε όταν ο χρήστης κάνει tap στην οθόνη να γίνει το κατέβασμα του Keyboard αν αυτό είναι ανεβασμένο. Η δομή του Widget είναι παρόμοια με του Login Widget με την μόνη διαφορά ότι εδώ το αίτημα που γίνεται είναι της εγγραφής.

3.7.7 Homepage

Το HomePage Widget αποτελεί το κυρίως κομμάτι της εφαρμογής. Μέσα σε αυτό βρίσκεται όλο το περιεχόμενο και η πλειονότητα των Widgets της εφαρμογής. Μέσα σε αυτό περιλαμβάνονται οι οθόνες των News Posts καθώς και των Podcasts.

Το HomePage αποτελεί ένα Stateful Widget του οποίου ο κώδικας είναι αυτός στις παρακάτω εικόνες:

```dart
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:sport_masters_v2/screens/podcasts/podcasts.dart';
import 'package:sport_masters_v2/screens/reportabug/reportabug.dart';
import 'package:sport_masters_v2/screens_widgets/customappbar.dart';
import 'package:sport_masters_v2/screens/homepage/homepagebody.dart';
import 'package:sport_masters_v2/screens/newspage/newspage.dart';
import 'package:sport_masters_v2/services/authentication.dart';

class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  int bottomSelectedIndex = 0;

  static final AuthService _auth = AuthService();

  PageController pageController = PageController(
    initialPage: 0,
    keepPage: true
  );

  void pageChanged(int index) {
    setState(() {
      bottomSelectedIndex = index;
    });
  }

  void bottomTapped(int index) {
    setState(() {
      bottomSelectedIndex = index;
      pageController.animateToPage(index, duration: Duration(milliseconds: 500), curve: Curves.ease);
    });
  }

  Future<bool> _onBackPressed() async {
    SystemChannels.platform.invokeMethod('SystemNavigator.pop');
    return true;
  }
```

Εικόνα 37: Κώδικας homepage.dart (1)

```dart
@override
Widget build(BuildContext context) {
  return new WillPopScope(
    onWillPop: () {
      return _onBackPressed();
    },
    child: SafeArea(
      child: GestureDetector(
        onTap: () {
          FocusScopeNode currentFocus = FocusScope.of(context);
          if (!currentFocus.hasPrimaryFocus) {
            currentFocus.unfocus();
          }
        },
        child: Scaffold(
        backgroundColor: Color(0xffffffff),
        drawer: Drawer(
          elevation: 16.0,
          child: Container(
            color: Color(0xff262626),
            child: ListView(
              padding: EdgeInsets.all(10.0),
              children: [
                ListTile(
                  title: Text(
                    "Settings",
                    style: TextStyle(
                      color: Colors.white
                    ), // TextStyle
                  ), // Text
                  onTap: () => {

                  }
                ), // ListTile
                ListTile(
                  title: Text(
                    "Report a Bug",
                    style: TextStyle(
                      color: Colors.white
                    ), // TextStyle
                  ), // Text
                  onTap: () => {
                    Navigator.push(context, MaterialPageRoute(builder: (context) => ReportABug()))
                  }
                ), // ListTile
                ListTile(
                  title: Text(
                    "Sign Out",
                    style: TextStyle(
                      color: Colors.white
                    ), // TextStyle
                  ), // Text
                  onTap: () async {
                    Navigator.of(context).pop();
                    await _auth.signOut();

                  }
                ), // ListTile
              ],
            ), // ListView
```

Εικόνα 38: Κώδικας homepage.dart (2)

```
        appBar: CustomAppBar(),
        body: PageView(
          controller: pageController,
          onPageChanged: (index) {
            pageChanged(index);
          },
          children: <Widget>[
            HomePageBody(),
            NewsPage(),
            Podcasts()
          ], // <Widget>[]
        ), // PageView
        bottomNavigationBar: Container(
          decoration: BoxDecoration(
            borderRadius: BorderRadius.only(
              topLeft: Radius.circular(200),
              topRight: Radius.circular(200)
            ) // BorderRadius.only
          ), // BoxDecoration
          child: BottomNavigationBar(
            onTap: (index) {
              bottomTapped(index);
            },
            currentIndex: bottomSelectedIndex,
            backgroundColor: Color(0xFF262626),
            unselectedItemColor: Colors.white,
            items: const <BottomNavigationBarItem> [
              BottomNavigationBarItem(
                icon: Icon(Icons.home),
                title: Text('Home'),
              ), // BottomNavigationBarItem
              BottomNavigationBarItem(
                icon: Icon(Icons.cast),
                title: Text('Νέα'),
              ), // BottomNavigationBarItem
              BottomNavigationBarItem(
                icon: Icon(Icons.mic),
                title: Text('Podcast'),
              ), // BottomNavigationBarItem
            ] // <BottomNavigationBarItem>[]
          ), // BottomNavigationBar
        ), // Container
      ) // Scaffold
    ) // GestureDetector
  ) // SafeArea
  ); // WillPopScope
 }
}
```

Εικόνα 39: Κώδικας homepage.dart (2)

Όσον αφορά τα imports βλέπουμε ότι περιέχονται:

- material.dart: Το import αυτό έχει χρησιμοποιηθεί ξανά και αφορά τα Material Components. Χρησιμοποιείται και σε αυτό το Widget
- services.dart: Το import αυτό χρησιμοποιείται στην προσθήκη την δυνατότητα εξόδου από την εφαρμογή σε περίπτωση που ο χρήστης πατήσει το Back Button του OS για Android
- podcasts.dart: Κομμάτι του HomePage είναι και η οθόνη των Podcasts επομένως έγινε και το ανάλογο import
- reportabug.dart: Στο Drawer της εφαρμογής που θα εξηγήσουμε στην συνέχεια τι ακριβώς είναι, περιλαμβάνεται η σελίδα για την αναφορά Bugs το Widget αυτό περιέχεται στο αρχείο το οποίο κάναμε import.
- customappbar.dart: Το import αυτό περιλαμβάνει το Widget το οποίο θα ορίσουμε ως App

Bar.

- homepagebody.dart: Το κομμάτι του HomePage που είναι η αρχική σελίδα.
- newspage.dart: Η σελίδα των κατηγοριοποιημένων νέων
- authentication.dart: Το import αυτό το χρησιμοποιούμε σε περίπτωση που ο χρήστης επιλέξει να αποσυνδεθεί για να εκτελεστεί το αίτημα στο Firebase.

Όσον αφορά το State αντικείμενο του Widget, οι παράμετροι του είναι:

- bottomSelectedIndex: Η μεταβλητή αυτή κρατάει τον τρέχον δείκτη του BottomNavigationBar. Μέσω αυτής υπάρχει επικοινωνία ανάμεσα στο PageView και BottomNavigationBar.
- _auth: Το αντικείμενο αυτό θα χρησιμοποιηθεί για την αποσύνδεση του χρήστη
- pageController: Το αντικείμενο αυτό είναι ένας χειριστής για το PageView. Μέσω αυτού του χειριστή θα μπορέσουμε να μεταφερόμαστε ανάμεσα στις διάφορες σελίδες με μεγαλύτερη ευκολία.

Οι μέθοδοι της κλάσης είναι 4:

- pageChanged: Η μέθοδος είναι void και με παράμετρο μια int μεταβλητή, γίνεται αρχικά η ανακατασκευή της διεπαφής του χρήστη και ορίζεται ως νέα τιμή της μεταβλητής bottomSelectedIndex η τιμή της παραμέτρου.
- bottomTapped: Void μέθοδος η οποία με παράμετρο μια int μεταβλητή,
- _onBackPressed: Η μέθοδος αυτή επιστρέφει μια Future τύπου bool τιμή. Συγκεκριμένα αυτό που κάνει είναι ότι με βάση το services.dart αρχείο, αξιοποιεί κανάλια του συστήματος και στην συγκεκριμένη περίπτωση αξιοποιείται το κανάλι το οποίο είναι υπεύθυνο για το κλείσιμο της εφαρμογής
- Build: Η build μέθοδος αποτελείται αρχικά από ένα WillPopScope Widget. Το Widget αυτό ανιχνεύει κάθε φορά που γίνεται Tap στο Back Button του λειτουργικού συστήματος. Όποτε συμβαίνει αυτό με την παραμέτρο onWillPop η οποία επιστρέφει μια callback μέθοδο, εκτελείται η _onBackPressed και όταν συμβεί αυτό τότε κλείνει η εφαρμογή.

  Έπειτα ακολουθεί το Scaffold του Widget. Το συγκεκριμένο περιλαμβάνει και drawer. Το Drawer είναι το αντικείμενο που μπορούμε να πούμε και ως Side Menu. Εκεί μέσα έχει οριστεί μία ListView η οποία περιέχει κάποια ListTiles. Τα Tiles αυτά είναι το Report a Bug και Sign Out. Αν ο χρήστης επιλέξει το πρώτο τότε θα μεταφερθεί στην οθόνη αναφοράς Bug. Από εκεί θα μπορεί να γυρίσει πίσω στην HomePage. Αν επιλέξει το δεύτερο, τότε θα εκτελεστεί το αίτημα αποσύνδεσης και θα γυρίσει πίσω στην οθόνη εισόδου.

  Μετά το Drawer, έχει οριστεί και το App Bar. Το App Bar ορίζει την μπάρα στο πάνω μέρος της εφαρμογής που περιέχεται σε κάθε κλασσικό Mobile App. Για App Bar έχει οριστεί ένα νέο Widget, το CustomAppBar.

3.7.8 Custom App Bar

Το περιεχόμενο αυτού του Widget είναι η δομή του App Bar το οποίο θα εμφανίζεται στην εφαρμογή. Ο κώδικας του είναι αυτός που εμφανίζεται στις παρακάτω εικόνες:

```dart
import 'package:flutter/material.dart';

class CustomAppBar extends StatefulWidget implements PreferredSizeWidget {

  CustomAppBar({Key key}) : preferredSize = Size.fromHeight(60), super(key:key);

  @override
  final Size preferredSize;

  @override
  _CustomAppBarState createState() => _CustomAppBarState();
}

class _CustomAppBarState extends State<CustomAppBar> {

  @override
  Widget build(BuildContext context) {
    return PreferredSize(
      preferredSize: widget.preferredSize,
      child: AppBar(
        centerTitle: true,
        leading: Padding(
          padding: EdgeInsets.only(top:5, left:10),
          child: IconButton(
            icon: Icon(Icons.menu),
            onPressed: () => Scaffold.of(context).openDrawer(),
          ), // IconButton
        ), // Padding
        iconTheme: IconThemeData(color: Colors.white),
        backgroundColor: Color(0xff262626),
        title: Padding(
          padding: const EdgeInsets.only(top: 5),
          child: Text(
            "SPORT MASTERS",
            style: TextStyle(
              fontFamily: 'Sports Jersey',
              fontSize: 35
            ), // TextStyle
          ), // Text
        ), // Padding
      ) // AppBar
    ); // PreferredSize
  }
}
```

Εικόνα 40: Κώδικας custombar.dart

Το CustomAppBar είναι ένα StatefulWidget το οποίο υλοποιεί το Interface PreferredSizeWidget. Ένα Interface είναι μια Abstract κλάση η οποία αν ο χρήστης ορίσει ότι η κλάση του υλοποιεί αυτήν την κλάση θα πρέπει και να υλοποιήσει και τις μεθόδους της. Η κλάση CustomAppBar εδώ θα πρέπει να υλοποιήσει την παράμετρο preferredSize. Ο λόγος υλοποίησης της PreferredSizeWidget είναι διότι το Scaffold Widget εξαρτάται από το προτιμόμενο μέγεθος ενός AppBar. Έτσι για να μπορέσουμε να ορίσουμε ένα Custom App Bar θα πρέπει και να δηλώσουμε τα παραπάνω. Ο δομητής της κλάσης ουσιαστικά ορίζει την τιμή της παραμέτρου preferredSize της κλάσης με μια στατική τιμή μεγέθους Size.fromHeight(60).

Το State αντικείμενο της κλάσης αποτελείται αποκλειστικά από την Build μέθοδο η οποία επιστρέφει ένα PreferredSize Widget, το οποίο ουσιαστικά καλύπτει την ανάγκη του Scaffold την οποία εξηγήσαμε προηγουμένως. Ακολουθεί το AppBar Widget ως child το οποίο δημιουργεί ένα Material Design AppBar. Η παράμετρος centerTitle ορίζει αν θα υπάρχει κεντραρισμένος τίτλος στο AppBar κάτι το οποίο έχει τιμή True στην συγκεκριμένη περίπτωση. Μετά ορίζεται η μεταβλητή leading, η οποία αφορά το τι θα εμφανίζεται πριν του centerTitle ή αλλιώς αριστερά του. Η τιμή του είναι ένα IconButton Widget το οποίο είναι εμφωλευμένο σε ένα Padding Widget. Ένα IconButton είναι ένα Widget το οποίο έχει την δυνατότητα απεικόνισης εικόνας και μπορεί να λειτουργήσει ως ένα κουμπί. Έτσι ορίζουμε το icon του με το Menu Icon από την συλλογή των διαθέσιμων Material Design Icons του Flutter και όταν κάποιος το πατήσει τότε χρησιμοποιούμε την μέθοδο openDrawer του Scaffold η οποία ελέγχει αν υπάρχει διαθέσιμο Drawer και αν υπάρχει τότε εκτελείται το Animation για την εμφάνιση του. Το Padding Widget παρέχει δυνατότητα περιορισμού του περιεχομένου και δημιουργία χώρου γύρω από αυτό. Στην συνέχεια ορίζουμε το θέμα το οποίο θα ακολουθήσουν τα Icons το οποίο ορίζει μόνο το χρώμα ως άσπρο, θέτουμε το χρώμα του Background και ορίζουμε τον τίτλο με ένα Text Widget.

Συνεχίζοντας με τη δομή του HomePage, βλέπουμε την παράμετρο body. Αυτή η παράμετρος ορίζει το σώμα ή αλλιώς το κυρίως περιεχόμενο του Scaffold. Στην περίπτωση μας έχουμε ένα PageView.
Το PageView είναι μια λίστα σελίδων που είναι Scrollable και παίρνει ως είσοδο μια λίστα Widgets. Τα Widgets εδώ που ορίζουμε για λίστα είναι:

- HomePageBody: Το Widget της αρχικής σελίδας.
- NewsPage: Το Widget των κατηγοριοποιημένων ειδήσεων.
- Podcasts: Το Widget για τα Podcasts.

Επόμενο κομμάτι της διάταξης είναι το Bottom Bar. Πέρα από την επιλογή για Top Navigation Bar οι χρήστες μπορούν να ορίσουν και Bottom Navigation Bar. Η επιλογή αυτή καλύπτει τις ανάγκες μας καθώς θα χρησιμοποιούμε τον Top Navigation ως Navigation για όλη την εφαρμογή και τον Bottom για την πλοήγηση στο PageView Widget. Έχουμε ορίσει σε περίπτωση που θα γίνει Tap στο Bar να εκτελεστεί η μέθοδος bottomTapped. Η συγκεκριμένη μέθοδος καλεί την μέθοδο setState και στην ολοκλήρωση της θέτει την μεταβλητή bottomSelectedIndex ίση με την τιμή της παραμέτρου index και εκτελεί την μέθοδο animateToPage του pageController στο Widget που βρίσκεται στο αντίστοιχο index του PageView. Ο τρέχων δείκτης, δηλαδή το ποια σελίδα εμφανίζεται, ορίζεται από την παράμετρο currentIndex της οποίας η τιμή είναι η τιμή της μεταβλητής bottomSelectedIndex. Έπειτα ορίζονται το χρώμα του προσκήνιου και το χρώμα που ένα αντικείμενο έχει όταν δεν είναι επιλεγμένο. Τέλος, ορίζεται και η λίστα των αντικειμένων τα οποία είναι τύπου
BottomNavigationBarItem, το αντίστοιχο Widget για την δημιουργία αντικειμένων τα οποία μπορούν να μπούν στο BottomNavigationBar. Τα αντικείμενα μας εδώ είναι τρία, για την αρχική οθόνη, την οθόνη των νέων και των Podcast αντίστοιχα. Και έτσι κλείνει η δομή του Widget homepage.

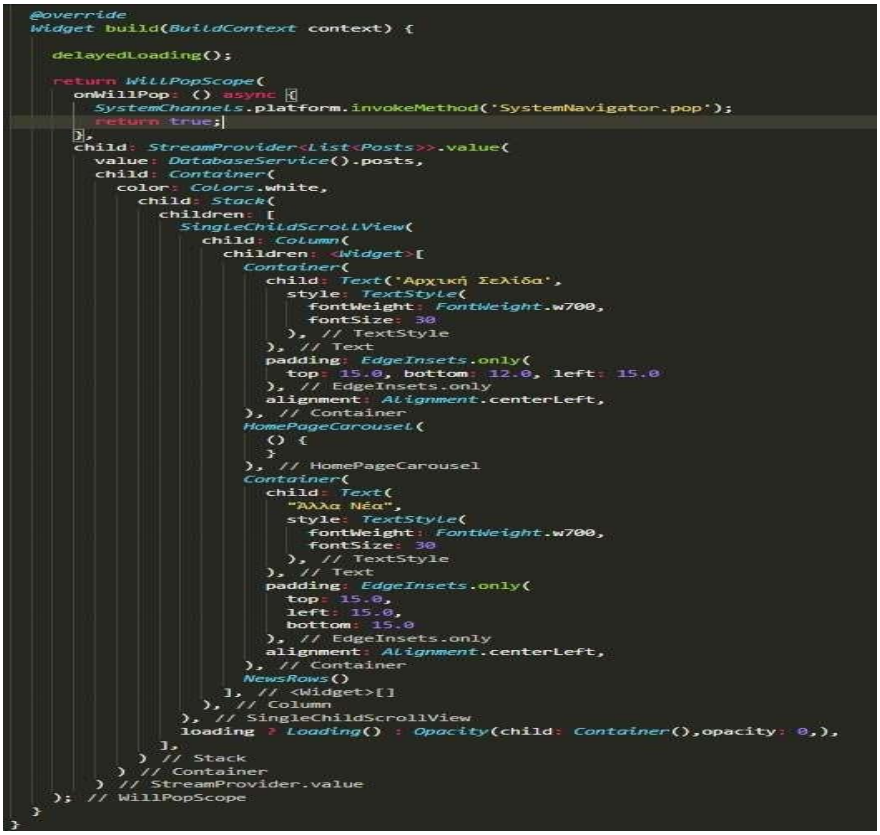
Με αυτό το Widget πετύχαμε την δημιουργία μιας δομής με την οποία ο χρήστης μπορεί να μεταφερθεί από την μια στην άλλη οθόνη με μια πολύ εύκολη πλοήγηση. Παρακάτω θα μιλήσουμε για την δομή της κάθε οθόνης του HomePage Widget ξεχωριστά ξεκινώντας από το HomePageBody.

3.7.9 HomePageBody

Το Widget αυτό ουσιαστικά είναι αυτό που ορίζει και την αρχική οθόνη της εφαρμογής. Ο κώδικας παρουσιάζεται στις παρακάτω εικόνες:

```dart
class _HomePageBodyState extends State<HomePageBody> {
  static bool loading = true;

  static CarouselController _carouselController = new CarouselController();

  static List<NewsTile> carouselitems = [];

  @override
  void initState() {
    super.initState();
  }

  @override
  void dispose() {
    super.dispose();
  }

  static CarouselOptions _carouselOptions = CarouselOptions(
    onPageChanged: (index, reason) {

    },
    autoPlay: true,
    height: 200,
    viewportFraction: 0.8,
    enlargeCenterPage: true,
    initialPage: 0,
    enableInfiniteScroll: true,
    scrollDirection: Axis.horizontal,
  );

  CarouselSlider carousel = CarouselSlider(
    options: _carouselOptions,
    carouselController: _carouselController,
    items: carouselitems,
  );

  void delayedLoading() async {
    Future.delayed(
      Duration(seconds: 5),
      () {
        setState(() {
          loading = false;
        });
      }
    ); // Future.delayed
  }
```

Εικόνα 41: Κώδικας homepagebody.dart (1)



```
@override
Widget build(BuildContext context) {

  delayedLoading();

  return WillPopScope(
    onWillPop: () async {
      SystemChannels.platform.invokeMethod('SystemNavigator.pop');
      return true;
    },
    child: StreamProvider<List<Posts>>.value(
      value: DatabaseService().posts,
      child: Container(
        color: Colors.white,
          child: Stack(
            children: [
              SingleChildScrollView(
                child: Column(
                  children: <Widget>[
                    Container(
                      child: Text('Αρχική Σελίδα',
                        style: TextStyle(
                          fontWeight: FontWeight.w700,
                          fontSize: 30
                        ), // TextStyle
                      ), // Text
                      padding: EdgeInsets.only(
                        top: 15.0, bottom: 12.0, left: 15.0
                      ), // EdgeInsets.only
                      alignment: Alignment.centerLeft,
                    ), // Container
                    HomePageCarousel(
                      () {
                      }
                    ), // HomePageCarousel
                    Container(
                      child: Text(
                        "Άλλα Νέα",
                        style: TextStyle(
                          fontWeight: FontWeight.w700,
                          fontSize: 30
                        ), // TextStyle
                      ), // Text
                      padding: EdgeInsets.only(
                        top: 15.0,
                        left: 15.0,
                        bottom: 15.0
                      ), // EdgeInsets.only
                      alignment: Alignment.centerLeft,
                    ), // Container
                    NewsRows()
                  ], // <Widget>[]
                ), // Column
              ), // SingleChildScrollView
              loading ? Loading() : Opacity(child: Container(),opacity: 0,),
            ],
          ) // Stack
        ) // Container
      ) // StreamProvider.value
    ); // WillPopScope
  }
}
```

Εικόνα 42: Κώδικας homepagebody.dart (2)

Όσον αφορά τα imports τα πράγματα είναι περίπου τα ίδια σε σχέση με αυτά που συναντήσαμε στα προηγούμενα Widgets. Αυτό το οποίο είναι καινούργιο και έχει σημασία είναι Carousel Slider. Το Carousel Slider χρησιμοποιείται καθώς αποτελεί κομμάτι της αρχικής οθόνης βάσει του Mockup και αποτελεί το πρώτο σημείο της εφαρμογής. Μέσω αυτού ο χρήστης θα μπορεί να βλέπει τα πιο πρόσφατα νέα και θα έχει την δυνατότητα να μπαίνει στην σελίδα με τις λεπτομέρειες του κάθε άρθρου.

Όσον αφορά τις παραμέτρους του State αντικειμένου, υπάρχουν:

- loading: bool μεταβλητή η οποία θα ορίσει τα Widgets τα οποία θα εμφανίζονται.
- _carouselController: Controller για το Carousel που θα δημιουργήσουμε
- carouselItems: Μια λίστα αντικειμένων τύπου NewsTile.

Πριν προχωρήσουμε θα πρέπει να εξηγήσουμε την δομή των NewsTile για να μπορέσουμε να καταλάβουμε τι αντικείμενα θα περιέχει το Carousel καθώς και οι μετέπειτα οθόνες.

3.7.10: NewsTile

Το NewsTile είναι ένα Widget το οποίο χρησιμοποιείται τόσο στο Carousel όσο και στην οθόνη NewsPage. Αποτελείται από τον εξής κώδικα:

```dart
import "package:flutter/material.dart";

class NewsTile extends StatefulWidget {
  final String title;
  final String url;
  final String description;
  final double height;

  NewsTile(this.title, this.description, this.url, this.height, {Key key}) : super(key: key);

  @override
  _NewsTileState createState() => _NewsTileState();
}

class _NewsTileState extends State<NewsTile> {
  static String title;
  static String link;
  static String desc;
  static double height;

  void initState() {
    title = widget.title ?? "";
    desc = widget.description ?? "";
    link = widget.url ?? "";
    height = widget.height ?? 200;
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return new Container(
      height: height,
      decoration: BoxDecoration(
        borderRadius: BorderRadius.circular(10),
        image: DecorationImage(
          image: NetworkImage(link),
          fit: BoxFit.fill
        ) // DecorationImage
      ), // BoxDecoration
      margin: EdgeInsets.all(10.0),
      child: Stack(
        children: <Widget>[
          Align(
            alignment: const Alignment(1.0, 1.0),
              child: Container(
                height: 170,
                decoration: BoxDecoration(
                  borderRadius: BorderRadius.only(
                    bottomLeft: const Radius.circular(8.0),
                    bottomRight: const Radius.circular(8.0)
                  ), // BorderRadius.only
                  gradient: LinearGradient(
                      begin: FractionalOffset.topCenter,
                      end: FractionalOffset.bottomCenter,
                      colors: [
                        Colors.transparent,
                        Colors.grey[850].withOpacity(1),
                      ],
                      stops: [0, 1.0],
                    ), // LinearGradient
                  ), // BoxDecoration
                ) // Container
              ), // Align
```

Εικόνα 43: Κώδικας newstile.dart (1)

```
Container(
  padding: EdgeInsets.all(10.0),
  child: Container(
    height: 180,
    child: Column(
      mainAxisSize: MainAxisSize.min,
      mainAxisAlignment: MainAxisAlignment.end,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
        Container(
          padding: EdgeInsets.only(bottom:5.0),
          child: Text(
            title,
            style: TextStyle(
              fontSize: 17,
              fontWeight: FontWeight.bold,
              color: Colors.white
            ), // TextStyle
          ) // Text
        ), // Container
        Text(
          desc,
          style: TextStyle(
            color: Colors.white,
            fontSize: 13
          ), // TextStyle
        ), // Text
      ], // <Widget>[]
    ), // Column
  ), // Container
) // Container
```

Εικόνα 44: Κώδικας newstile.dart (2)

Το Widget είναι Stateful και περιλαμβάνει στην κλάση του τις εξής παραμέτρους:

- title: Final String μεταβλητή στην οποία θα βρίσκεται ο τίτλος του άρθρου
- url: Final String μεταβλητή στην οποία θα βρίσκεται το url για την εικόνα προεπισκόπηση
- description: Final String μεταβλητή στην οποία θα περιλαμβάνεται η περιγραφή του άρθρου
- height: Final Double μεταβλητή η οποία θα ορίζει το μέγεθος του Widget

Στο δομητή περιλαμβάνονται αυτά τα πεδία και βλέπουμε ότι δεν χρειάζεται να ορίσουμε την ανάθεση τιμών καθώς η Dart αναλαμβάνει το έργο αυτό η ίδια. Το State αντικείμενο περιλαμβάνει στην συνέχεια τις εξής παραμέτρους:

- title: Static String μεταβλητή στην οποία θα αποθηκευτεί ο τίτλος του άρθρου
- link: Static String μεταβλητή στην οποία θα αποθηκευτεί το link της φωτογραφίας προεπισκόπησης
- desc: Static String μεταβλητή στην οποία θα αποθηκευτεί η περιγραφή του άρθρου
- height: Static String μεταβλητή στην οποία θα αποθηκευτεί το ύψος του Widget.

Στην συνέχεια οι μέθοδοι είναι:

- initState: Γνωρίζουμε την χρήση της μεθόδου από τα προηγούμενα Widgets. Η μόνη διαφορά εδώ είναι ότι θα ορίσουμε την τιμή των μεταβλητών του State αντικειμένου με τις τιμές των παραμέτρων του αντικειμένου με το οποίο έγινε και αρχικοποίηση του αντικειμένου.

- build: Εδώ η δομή του Widget που επιστρέφεται αποτελείται από ένα Container το οποίο ουσιαστικά χρησιμοποιούμε για να περιορίσουμε το Widget στο ύψος το οποίο θα λάβουμε ως είσοδο από την παράμετρο height. Έπειτα ορίζουμε την διακόσμηση του και συγκεκριμένα ορίζουμε το borderRadius δηλαδή το πόσο στρογγυλοποιημένο θα είναι το περίγραμμα και ορίζουμε την εικόνα που θα διακοσμεί το Container. Ο τρόπος που το υλοποιούμε αυτό είναι κάνοντας ένα Network Request με το Widget NetworkImage το οποίο παίρνει ως παράμετρο το URL της εικόνας και επιστρέφει την εικόνα. Πάνω σε αυτό το Container τοποθετούμε ένα Stack το οποίο περιλαμβάνει ένα στρώμα το οποίο είναι ένα Gradient Layer για να μπορέσουμε να δημιουργήσουμε μια επιφάνεια στην οποία τα γράμματα του τίτλου θα κάθονται και θα δημιουργείται μια επιθυμητή αντίθεση για την ευαναγνωσία του τίτλου καθώς τα χρώματα των εικονών είναι δυναμικά και μπορεί να υπάρξει η περίπτωση τα χρώματα αυτά να δυσκολέψουν τον χρήστη να διαβάσει αυτό το οποίο υπάρχει στην οθόνη. Έπειτα κομμάτι του Stack αποτελεί ένα ακόμα Container το οποίο περιλαμβάνει μέσα ένα Column Widget. Μέσα σε αυτό το Column θα ορίσουμε τον πίνακα των Widgets τα οποία αποτελούνται από τα Text Widgets του τίτλου και της περιγραφής. Έτσι καταλήγουμε στο αποτέλεσμα το οποίο φαίνεται στην παρακάτω εικόνα:

Εικόνα 45: Υπόδειγμα εμφάνισης ενός NewsTile Widget

Έχοντας το Widget το οποίο θα αποτελέσει το βασικό αντικείμενο του Carousel συνεχίζουμε στον τρόπο με τον οποίο θα γεμίσουμε δεδομένα αυτά τα αντικείμενα. Ο τρόπος είναι γνωστός και γίνεται με την χρήση του StreamProvider. Αυτήν την φορά η τιμή που χρησιμοποιείται είναι τα Posts αντικείμενα. Άρα αυτόματα και τα Widgets παιδιά έχουν πρόσβαση σε αυτήν την τιμή. Συνεχίζοντας με την διάταξη του Widget βλέπουμε ότι η οθόνη αποτελείται από ένα Stack το οποίο έχει ως children δύο Widgets. Το ένα από αυτά αποτελεί το υπόλοιπο κομμάτι της οθόνης. Το δεύτερο αποτελεί την Loading οθόνη η οποία με βάση την τιμή της μεταβλητής loading θα εμφανίζεται ή θα εξαφανίζεται αναλόγως. Το άλλο αντικείμενο αποτελεί ένα SingleChildScrollView την λειτουργία του οποίου έχουμε εξηγήσει σε προηγούμενα Widgets. Μέσα σε αυτό περιλαμβάνονται ένα Container το οποίο περιέχει το Text “Αρχική Σελίδα” και ένα ακόμα για τα “Άλλα Νέα”. Τα καινούργια κομμάτια εδώ είναι τα HomePageCarousel και NewsRows.

3.7.11: HomePageCarousel

Το HomePageCarousel αποτελεί το Custom Carousel το οποίο θα χρησιμοποιήσουμε για την υλοποίηση του Carousel στην αρχική οθόνη. Αποτελείται από τον εξής κώδικα:

```dart
import 'package:carousel_slider/carousel_slider.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sport_masters_v2/animations/bottomtotopslide.dart';
import 'package:sport_masters_v2/models/posts.dart';
import 'package:sport_masters_v2/screens/newsdetails/newsdetails.dart';
import 'package:sport_masters_v2/screens_widgets/newstile.dart';

class HomePageCarousel extends StatefulWidget {

  HomePageCarousel({Key key}) : super(key: key);

  @override
  _HomePageCarouselState createState() => _HomePageCarouselState();
}

class _HomePageCarouselState extends State<HomePageCarousel> {

  static int carouselcounter = _carouselOptions.initialPage;

  static List<Widget> carouselItems = [];

  static CarouselOptions _carouselOptions = CarouselOptions(

    onPageChanged: (index, reason) {
      carouselcounter = index;
    },
    autoPlay: false,
    height: 200,
    viewportFraction: 0.8,
    enlargeCenterPage: true,
    initialPage: 0,
    enableInfiniteScroll: true,
    scrollDirection: Axis.horizontal,
  );

  void initState() {
    super.initState();
  }
```

Εικόνα 46: Κώδικας homepagecarousel.dart (1)

```dart
@override
Widget build(BuildContext context) {

  final posts = Provider.of<List<Posts>>(context);

  for (var items in posts) {
    posts.add(items);
    carouselItems.add(
      NewsTile(
        items.title,
        items.description,
        items.previewImage,
        200,
      )
    );
  }

  return GestureDetector(
    child: Hero(
      tag: 'carouselHero',
      child: CarouselSlider(
        items: carouselItems,
        options: _carouselOptions
      ) // CarouselSlider
    ), // Hero
    onTap: () {
      Navigator.push(context, BottomToTopSlide(builder: (context) => NewsDetails(posts[carouselcounter])));
    },
  ); // GestureDetector
 }
}
```

Εικόνα 47: Κώδικας homepagecarousel.dart (2)

Όσον αφορά το State αντικείμενο οι μεταβλητές είναι:

- carouselcounter: Static int μεταβλητή η οποία χρησιμοποιείται για να αποθηκεύεται το τρέχω index του carousel.
- carouselItems: Static λίστα Widgets η οποία χρησιμοποιείται για να αποθηκεύονται τα αντικείμενα του Carousel
- postsItems: Static λίστα στην οποία θα γίνεται η αποθήκευση των Posts.
- _carouselOptions: Static μεταβλητή τύπου CarouselOptions της οποίας ορίζουμε τις εξής παραμέτρους:
    - onPageChanged: Callback μέθοδος η οποία εκτελείται όταν η σελίδα του Carousel αλλάξει. Στην συγκεκριμένη περίπτωση εκτελείται η ανάθεση του index του Carousel στην μεταβλητή carouselcounter
    - autoPlay: Ορίζει αν το carousel θα εκτελεί κίνηση Scroll αυτόματα
    - height: Ύψος του Carousel o viewportFraction: Κομμάτι του Carousel το οποίο θα καταλαμβάνει το κάθε αντικείμενο
    - enlargeCenterPage: Ορίζεται αν θα μεγενθύνεται το μεσαίο αντικείμενο ή όχι o initialPage: Ορίζει την αρχική σελίδα

- enableInfiniteScroll: Ορίζει αν το Carousel όταν φτάσει στο τέλος θα γυρνάει στο τέλος κάνοντας κύκλο
- scrollDirection: Κατεύθυνση στην οποία θα εκτελείται το Scrolling

Όσον αφορά την build μέθοδο αρχικά βλέπουμε μια τοπική μεταβλητή posts η οποία χρησιμοποιεί το αντικείμενο Provider για να τραβήξει την λίστα των Posts. Στην συνέχεια με μια for συνθήκη για κάθε άρθρο στην λίστα γίνεται η προσθήκη στην λίστα των postsItems των Posts του Provider και προσθήκη των carouselitems ενός NewsTile με δεδομένα το αντίστοιχο Posts αντικείμενο. Αυτό που επιστρέφεται από αυτήν την μέθοδο είναι αρχικά ένα GestureDetector αντικείμενο το οποίο έχει ως child το Carousel μαζί με όλες τις ρυθμίσεις που του χρειάζονται και όταν γίνεται Tap πάνω στο carousel γίνεται η μεταφορά στην οθόνη NewsDetails.

Επόμενο κομμάτι της οθόνης αποτελούν τα NewsRows.

3.7.12: NewsRows

Το NewsRows Widget ουσιαστικά είναι η στήλη των σειρών στο κάτω μέρος της οθόνης. Ο κώδικας είναι ο εξής:

```dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sport_masters_v2/models/posts.dart';
import 'package:sport_masters_v2/screens/newsdetails/newsdetails.dart';
import 'package:sport_masters_v2/screens_widgets/newsrow.dart';

class NewsRows extends StatefulWidget {
  @override
  _NewsRowsState createState() => _NewsRowsState();
}

class _NewsRowsState extends State<NewsRows> {

  static List<Widget> postsLists;

  static int counter = 0;

  @override
  Widget build(BuildContext context) {
    final posts = Provider.of<List<Posts>>(context);

    postsLists = [];

    counter = 0;

    for (var item in posts) {
      if (counter > 5) {
        postsLists.add(
          GestureDetector(
            child: NewsRow(
              item.title,
              item.previewImage
            ), // NewsRow
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => NewsDetails(item)
                ) // MaterialPageRoute
              );
            },
          ) // GestureDetector
        );
      }
      counter++;
    }

    return Column(
      children: postsLists,
    );
  }
}
```

Εικόνα 48: Κώδικας newsrows.dart

Ως προς τις παραμέτρους του StatefulWidget έχουμε:

- postsLists: Static List<String> μεταβλητή στην οποία θα αποθηκευτεί η λίστα των άρθρων
- counter: int μεταβλητή την οποία χρησιμοποιούμε για να πάρουμε τα Posts τα οποία είναι μετά τα 5 πιο πρόσφατα.

Η μόνη μέθοδος εδώ είναι η build στην οποία ορίζεται η final μεταβλητή posts στην οποία αποθηκεύεται η τιμή του Provider. Στην συνέχεια ορίζεται ο κενός πίνακας postsLists και η μεταβλητή counter παίρνει τιμή 0 και με μια for συνθήκη για κάθε αντικείμενο της μεταβλητής posts ελέγχεται αρχικά αν το counter είναι μεγαλύτερο του 5 και αν η συνθήκη είναι αληθής, τότε προστίθεται στην λίστα postsLists ένα νέο Widget το οποίο είναι ένα GestureDetector Widget με παιδί ένα NewsRow το οποίο δέχεται ως παραμέτρους τα αντίστοιχα δεδομένα του αντικειμένου της λίστας posts. Όταν γίνει tap στο αντικείμενο τότε γίνεται μεταφορά στην οθόνη NewsDetails η οποία παίρνει ως παράμετρο ένα αντικείμενο τύπου Posts. Μετά την if συνθήκη η τιμή της counter αυξάνεται κατά 1. Στο τέλος αυτό το οποίο επιστρέφει η μέθοδος είναι ένα Column Widget με παιδιά την λίστα των NewsRow αντικειμένων.

3.7.12.1: NewsRow

Από την μέθοδο build του NewsRows το μόνο το οποίο δεν γνωρίζαμε ήταν το Widget NewsRow. Το Widget αυτό είναι η κάθε σειρά που εμφανίζεται στο κάτω μέρος της οθόνης. Ο κώδικας του είναι ο εξής:

```dart
import 'package:flutter/material.dart';

class NewsRow extends StatefulWidget {

  final String title;
  final String image;

  NewsRow(this.title, this.image, {Key key}): super(key: key);

  @override
  _NewsRowState createState() => _NewsRowState();
}

class _NewsRowState extends State<NewsRow> {

  String title;
  String url;

  void initState() {
    super.initState();
    title = widget.title;
    url = widget.image;
  }
```

Εικόνα 49: Κώδικας newsrow.dart (1)

```dart
@override
Widget build(BuildContext context) {
  return Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: <Widget>[
      Container(
        decoration: BoxDecoration(
          border: Border(
            bottom: BorderSide(width: 2, color: Colors.white)
          ) // Border
        ), // BoxDecoration
        child: Image(
          fit: BoxFit.cover,
          alignment: Alignment.center,
          image: NetworkImage(url),
          height: MediaQuery.of(context).size.height * 0.15,
          width: MediaQuery.of(context).size.width * 0.35,
        ), // Image
      ), // Container
      Stack(
        children:<Widget>[
          Container(
            decoration: BoxDecoration(
              border: new Border(
                left: BorderSide(
                  color: Colors.white,
                  width: 3
                ), // BorderSide
                bottom: BorderSide(
                  color: Colors.white,
                  width: 2
                ) // BorderSide
              ), // Border
              color: Colors.blueAccent,
            ), // BoxDecoration
            alignment: Alignment.centerLeft,
            padding: EdgeInsets.only(
              top: 10,
              left: 7,
              right: 10
            ), // EdgeInsets.only
            height: MediaQuery.of(context).size.height * 0.15,
            width: MediaQuery.of(context).size.width * 0.65,
            child: Text(
              title,
              style: TextStyle(
                color: Colors.white,
                fontSize: 15,
                fontWeight: FontWeight.w600
              ), // TextStyle
            ) // Text
          ), // Container
        ] // <Widget>[]
      ) // Stack
    ], // <Widget>[]
  ); // Row
}
```

Εικόνα 50: Κώδικας newsrow.dart (2)

Οι παράμετροι του Widget είναι:

- title: Τίτλος του άρθρου
- description: Περιγραφή του άρθρου

Όσον αφορά την κλάση του State αντικειμένου, αποτελείται και αυτή από δύο παραμέτρους, μέσα στις οποίες θα αποθηκευτούν ο τίτλος και η περιγραφή του άρθρου αντίστοιχα. Ως προς τις μεθόδους της κλάσης περιλαμβάνεται η initState στην οποία γίνεται η ανάθεση των τιμών και η build η οποία επιστρέφει ένα νέο Row Widget. Ένα Row Widget έχει μια συγκεκριμένη μορφοποίηση η οποία το κάνει να λειτουργεί και να φαίνεται σαν σειρά. Μέσα στο Row τα Widgets παιδιά είναι αρχικά ένα Container το οποίο διακοσμούμε χρησιμοποιώντας το αντικείμενο BoxDecoration και θέτουμε ως child ένα Image Widget το οποίο θα χρησιμοποιήσει το NetworkImage Widget για να ζητήσει την εικόνα από το url που θα του δώσουμε. Στην συνέχεια εμφανίζεται ένα Stack του οποίου ορίζουμε μερικές από τις ιδιότητες και θέτουμε ως child του ένα Text Widget το οποίο περιλαμβάνει τον τίτλο του άρθρου.

Έτσι λοιπόν ορίζεται και το NewsRow και αυτό ολοκληρώνει το Layout της αρχικής οθόνης. Επόμενο κομμάτι της εφαρμογής αποτελεί το Widget NewsPage το οποίο αποτελεί την οθόνη των κατηγοριοποιημένων νέων.

3.7.13: NewsPage

Το NewsPage αποτελεί το Widget το οποίο είναι η σελίδα την οποία χρησιμοποιούμε στο PageView για την εμφάνιση των άρθρων ανά κατηγορία. Ο κώδικας επαναλαμβάνεται σε ορισμένα σημεία για αυτό θα δούμε ορισμένα σημεία του.

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:provider/provider.dart';
import 'package:sport_masters_v2/models/posts.dart';
import 'package:sport_masters_v2/screens/newsdetails/basketnews.dart';
import 'package:sport_masters_v2/screens/newsdetails/footballnews.dart';
import 'package:sport_masters_v2/screens/newsdetails/othernews.dart';
import 'package:sport_masters_v2/screens/newsdetails/tennisnews.dart';
import 'package:sport_masters_v2/screens/newsdetails/volleynews.dart';
import 'package:sport_masters_v2/services/database.dart';
```

Εικόνα 51: Κώδικας newspage.dart (1)

Όσον αφορά τα imports τα περιέχονται imports τα οποία έχουμε ξαναχρησιμοποιήσει. Τα καινούργια είναι αυτά τα οποία ανήκουν στον φάκελο newsdetails. Αυτά τα imports είναι το περιεχόμενο της κάθε κατηγορίας νέων σε ξεχωριστά Widget. Ο λόγος που έγινε αυτός ο διαχωρισμός είναι κυρίως για λόγους οργάνωσης.

Όσον αφορά τον κώδικα, το Widget έχει την κλασσική δομή που ξέρουμε με την createState μέθοδο.

```
class NewsPage extends StatefulWidget {
  @override
  _NewsPageState createState() => _NewsPageState();
}
```

Εικόνα 52: Κώδικας newspage.dart (3)

Στην συνέχεια η κλάση του State αντικειμένου έχει τρείς παραμέτρους:

- contentType: String μεταβλητή την οποία θα αξιοποιήσουμε για την αλλαγή της κατηγορίας και του περιεχομένου μέσα στο Widget.
- contentWidget: Widget μεταβλητή η οποία θα ορίζει το περιεχόμενο της σελίδας με βάση την κατηγορία.

```
class _NewsPageState extends State<NewsPage> {

  String contentType = "Football";

  Widget contentWidget = FootballNews();
```

Εικόνα 53: Κώδικας newspage.dart (4)

Η μόνη μέθοδος της κλάσης εδώ είναι η build στην οποία επιστρέφει ένα WillPopScope αντικείμενο την λειτουργία του οποίου έχουμε ξαναεξηγήσει. Το child αυτού του Widget είναι ένα StreamProvider αντικείμενο του οποίου το value είναι μια λίστα από Posts. Το value θα επιστρέφεται από την getter μεταβλητή posts της DatabaseService κλάσης. Μέσα στην builder παράμετρο του StreamProvicer γίνεται μια Switch συνθήκη με την οποία ελέγχεται ποιο είναι το περιεχόμενο της μεταβλητής contentType και ανάλογα το περιεχόμενο της γίνεται η αντίστοιχη ανάθεση τιμής στην contentWidget.

```
switch (contentType) {
  case "Football":
    contentWidget = FootballNews();
    break;
  case "Basket":
    contentWidget = BasketNews();
    break;
  case "Volley":
    contentWidget = VolleyNews();
    break;
  case "Tennis":
    contentWidget = TennisNews();
    break;
  case "Other":
    contentWidget = OtherNews();
    break;
  default:
    break;
}
```

Εικόνα 54: Κώδικας newspage.dart (2)

Αφού ολοκληρωθεί αυτή η διαδικασία εκτελείται η return εντολή η οποία επιστρέφει ένα Container το οποίο περιέχει ένα SingleChildScrollView με ένα Column για αντίστοιχο child. Μέσα στο Column περιλαμβάνονται το Text “Τα νέα σήμερα”, που είναι ο τίτλος της οθόνης, στην συνέχεια ένα Container με ένα Row μέσα. Αυτό το Row περιλαμβάνει τα buttons για τις 5 κατηγορίες σε Containers τα οποία είναι εμφωλευμένα μέσα σε GestureDetector Widgets για να μπορέσουμε να αναγνωρίζουμε τα Taps και να εκτελούμε τη setState κάθε φορά όπου θα ορίζουμε το περιεχόμενο της μεταβλητής contentType και αυτόματα θα αλλάζουμε το περιεχόμενο της οθόνης. Έτσι για παράδειγμα, αν γίνει tap στο Container του Tennis τότε αυτόματα ο GestureDetector θα αναγνωρίσει το Tap και θα εκτελέσει την

setState όπου θα ορίσει την τιμής της μεταβλητής contentType σε “Tennis” και στο επόμενο build του Widget το περιεχόμενο της contentWidget θα είναι η TennisNews.

```
GestureDetector(
  child: Column(
    children: <Widget>[
      Container(
        height: MediaQuery.of(context).size.height * 0.08,
        width: MediaQuery.of(context).size.height * 0.08,
        decoration: BoxDecoration(
          color: Color(0xff262626),
          borderRadius: BorderRadius.circular(30)
        ), // BoxDecoration
        child: Padding(
          padding: EdgeInsets.all(10),
            child: Image.asset(
            "assets/images/baseline_sports_soccer_white_48dp.png",
            height: MediaQuery.of(context).size.height * 0.02,
            width: MediaQuery.of(context).size.height * 0.02,
          ) // Image.asset
        ), // Padding
      ), // Container
      Padding(
        padding: EdgeInsets.only(top: 10),
        child: Text(
          "Ποδόσφαιρο",
          style: new TextStyle(
            fontSize: 13
          ), // TextStyle
        ) // Text
      ) // Padding
    ], // <Widget>[]
  ), // Column
  onTap: () {
    setState(() {
      contentType = "Football";
    });
  },
), // GestureDetector
```

Εικόνα 55: Κώδικας newspage.dart (6)

Έχοντας ορίσει την δομή του γενικού Layout θα προχωρήσουμε στην ανάλυση ενός από τα 5 Widgets τα οποία ορίζουν το ποια άρθρα θα εμφανιστούν και πως. Επειδή υπάρχουν πολλές ομοιότητες μεταξύ τους για να μην επαναληφθεί η ίδια πληροφορία πολλές φορές θα αναλυθεί η δομή μόνο ενός από αυτά και θα εξηγήσουμε τι αλλάζει στο κάθε Widget.

3.7.13.1: FootballNews

Ένα από τα 5 Widgets τα οποία χρησιμοποιούνται στην NewsPage είναι το FootballNews. Ο κώδικας του είναι ο εξής:

```dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sport_masters_v2/models/posts.dart';
import 'package:sport_masters_v2/screens/newsdetails/newsdetails.dart';
import 'package:sport_masters_v2/screens_widgets/newstile.dart';

class FootballNews extends StatefulWidget {
  @override
  _FootballNewsState createState() => _FootballNewsState();
}

class _FootballNewsState extends State<FootballNews> {

  static bool loaded;

  List<Widget> items = [];

  @override
  void initState() {
    super.initState();
    loaded = false;
  }


  @override
  Widget build(BuildContext context) {
    final posts = Provider.of<List<Posts>>(context);

    if (!loaded) {
      for (var item in posts) {
        if (item != null && item.category == "Football") {
          items.add(
            GestureDetector(
              child: NewsTile(
                item.title, item.description, item.previewImage, 200
              ), // NewsTile
              onTap: () {
                Navigator.push(context, MaterialPageRoute(builder: (context) => NewsDetails(item)));
              }
            ) // GestureDetector
          );
        }
      }

      loaded = !loaded;
    }

    return Column(
      children: items
    );
  }
}
```

Εικόνα 56: Κώδικας footballnews.dart

Όσον αφορά το State αντικείμενο υπάρχουν δύο μεταβλητές:

- loaded: Static bool μεταβλητή η οποία θα χρησιμοποιηθεί για να μην υπάρχει φόρτωση των δεδομένων παραπάνω από μια φορές.

- items: Οι λίστα των Widgets η οποία θα εμφανίζεται.

Στην initState ορίζουμε την μεταβλητή με false για να πούμε ότι δεν έχουν φορτωθεί ακόμα τα δεδομένα στην οθόνη.

Η build μέθοδος έχει μια Provider final μεταβλητή posts η οποία θα εκτελέσει την λειτουργία του παρόχου στα child Widgets. Στην συνέχεια με μια συνθήκη if ελέγχεται αν τα δεδομένα δεν έχουν φορτωθεί ακόμα. Αν η συνθήκη ισχύει τότε με μια for συνθήκη ελέγχεται για κάθε item της μεταβλητής Posts αν δεν είναι null και αν η κατηγορία του είναι ίση με Football και αν αυτή η συνθήκη ισχύει τότε προστίθεται στην λίστα των items ένα νέο GestureDetector Widget το οποίο έχει ως child ένα NewsTile Widget το οποίο δέχεται για δεδομένα τα στοιχεία του τρέχοντος item στην for συνθήκη και όταν ο χρήστης πατήσει πάνω στο Widget έχει οριστεί να μεταφέρεται στην σελίδα NewsDetails. Τέλος η εντολή return επιστρέφει ένα Column το οποίο περιέχει τα items της κατηγορίας ποδοσφαίρου.

Αν μεταφερόμασταν στα άλλα 4 Widgets η μόνη διαφορά που θα υπήρχε στον κώδικα θα ήταν στην if συνθήκη για την κατηγορία του Item όπου θα ελεγχόταν αν η τιμή του θα ήταν “Basket”, “Volley”, “Tennis” ή “Other” αντίστοιχα και ανάλογα την κατηγορία θα επέστρεφαν τα αντίστοιχα items. Έτσι λοιπόν ολοκληρώνεται και η διάταξη της σελίδας NewsPage. Επόμενο κομμάτι είναι η οθόνη NewsDetails.

3.7.14: NewsDetails

Η News Details αποτελεί την οθόνη στην οποία κάθε Tap πάνω σε ένα άρθρο καταλήγει. Σε αυτό το Widget ο χρήστης θα βλέπει όλες τις πληροφορίες του άρθρου.

Ο κώδικας του Widget είναι ο εξής:

```dart
import 'package:flutter/material.dart';
import 'package:sport_masters_v2/models/posts.dart';

class NewsDetails extends StatefulWidget {
  final Posts podcastItem;

  NewsDetails(Posts item, {Key key}) :
  podcastItem = item,
  super(key: key);

  @override
  _NewsDetailsState createState() => _NewsDetailsState();
}

class _NewsDetailsState extends State<NewsDetails> {

  Posts post;

  @override
  void initState() {
    post = widget.podcastItem;
    super.initState();
  }

  @override
  Widget build(BuildContext context) {

    List<Widget> imagesWidgets = [
      Text(
        "Εικόνες",
        style: TextStyle(
          color: Colors.black,
          fontSize: 20,
          fontWeight: FontWeight.w900
        ), // TextStyle
      ), // Text
      SizedBox(
        height: MediaQuery.of(context).size.height / 30,
      )
    ];
```

Εικόνα 57: Κώδικας newsdetails.dart (1)

```dart
for (var image in post.images) {
  imagesWidgets.add(
    SizedBox(
      width: MediaQuery.of(context).size.width,
      child: Image.network(
        image,
        fit: BoxFit.fill,
        loadingBuilder: (BuildContext context, Widget child,
            ImageChunkEvent loadingProgress) {
          if (loadingProgress == null) return child;
          return Center(
            child: CircularProgressIndicator(
              value: loadingProgress.expectedTotalBytes != null
                  ? loadingProgress.cumulativeBytesLoaded /
                      loadingProgress.expectedTotalBytes
                  : null,
            ), // CircularProgressIndicator
          ); // Center
        },
      ), // Image.network
    ), // SizedBox
  );
  imagesWidgets.add(SizedBox(height: 20,));
}
```

Εικόνα 58: Κώδικας newsdetails.dart (2)

```
return WillPopScope(
  onWillPop: () async {
    Navigator.of(context).pop();
    return true;
  },
  child: Scaffold(
    appBar: AppBar(
      backgroundColor: Color(0xff262626),
    ), // AppBar
    body: ListView(
      children: [
        Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            Center(
              child: Stack(
                children: [
                  Container(
                    height: MediaQuery.of(context).size.height / 4,
                    decoration: BoxDecoration(
                      image: DecorationImage(
                        image: NetworkImage(post.previewImage),
                        fit: BoxFit.cover
                      ) // DecorationImage
                    ), // BoxDecoration
                  ), // Container
                  Container(
                    height: MediaQuery.of(context).size.height / 4,
                    decoration: BoxDecoration(
                      gradient: LinearGradient(
                          begin: FractionalOffset.center,
                          end: FractionalOffset.bottomCenter,
                          colors: [
                            Colors.transparent,
                            Colors.grey[850].withOpacity(0.8),
                          ],
                          stops: [0, 1],
                        ), // LinearGradient
                      ), // BoxDecoration
                  ), // Container
                  Padding(
                    padding: const EdgeInsets.only(top: 130, left: 20),
                    child: Text(
                      post.title,
                      style: TextStyle(
                        color: Colors.white,
                        fontSize: 20,
                        fontWeight: FontWeight.w800
                      ), // TextStyle
                    ), // Text
                  ) // Padding
                ],
              ), // Stack
            ), // Center
            SizedBox(
              height: 20,
            ), // SizedBox
            Padding(
              padding: const EdgeInsets.all(20),
              child: Text(
                post.description,
                style: TextStyle(
```

Εικόνα 59: Κώδικας newsdetails.dart (3)

```
                fontSize: 20,
                fontWeight: FontWeight.w800
              ), // TextStyle
              textAlign: TextAlign.left,
            ), // Text
          ), // Padding
          Padding(
            padding: const EdgeInsets.all(20),
            child: Text(
              post.mainContent
            ), // Text
          ), // Padding
        ],
      ), // Column
      Column(
        children: imagesWidgets,
      ) // Column
    ]
  ), // ListView
  // Scaffold
  // WillPopScope
```

Εικόνα 60: Κώδικας newsdetails.dart

Τα imports είναι δύο και περιλαμβάνουν το material Package και το μοντέλο των Posts.

Στην συνέχεια στην κλάση του Widget περιλαμβάνεται μια final Posts μεταβλητή η οποία χρησιμοποιείται στον δομητή και αποτελεί το αντικείμενο του άρθρου το οποίο θα αξιοποιήσει το State αντικείμενο.

Στην κλάση του State αντικειμένου περιλαμβάνεται μια παράμετρος post τύπου Posts η οποία χρησιμοποιείται στην initState και αποκτά τιμή την τιμή της podcastItem.

Η μέθοδος build ξεκινάει με τον ορισμό της παραμέτρου imagesWidgets η οποία είναι μια λίστα από Widgets τα οποία θα χρησιμοποιηθούν στο τέλος της διάταξης της οθόνης. Ορίζονται από ένα Text Widget το οποίο είναι ο τίτλος και στην συνέχεια με μία for συνθήκη για κάθε αντικείμενο στην παράμετρο του post αντικειμένου images προστίθεται στην λίστα ένα νέο SizedBox το οποίο έχει ως child μια εικόνα η οποία λαμβάνεται από το δίκτυο χρησιμοποιώντας την μέθοδο network της κλάσης Image. Για να μην μένει κενή η οθόνη κατά την διάρκεια εκπλήρωσης του αιτήματος στο διαδίκτυο υπάρχει η παράμετρος loadingBuilder η οποία όσο το loadingProgress έχει τιμή επιστρέφει το Widget CircularProgressIndicator το οποίο αποτελεί τον γνωστό κυκλικό Loader. Στην συνέχεια η return επιστρέφει ένα νέο WillPopScope αντικείμενο το οποίο σε περίπτωση που ο χρήστης πατήσει κάποιο Back button θα αφαιρέσει το Widget από το Stack του Navigator. Το child του WillPopScope είναι ένα Scaffold το οποίο περιλαμβάνει AppBar και Body. Μέσα στο Body περιέχεται ένα ListView. Το ListView μας δίνει την δυνατότητα το body να είναι Scrollable και έχει ως children ένα Column το οποίο αποτελεί το κομμάτι της οθόνης το οποίο περιλαμβάνει την εικόνα στην οποία επάνω με την χρήση του Stack βρίσκεται ο τίτλος του άρθρου. Από κάτω, σε ένα ξεχωριστά Text Widget βρίσκονται η περιγραφή του άρθρου και το κυρίως περιεχόμενο του. Κλείνοντας η λίστα imageWidgets την οποία είχαμε ρυθμίσει στην αρχή προστίθεται για να δείξουμε και τις σχετικές φωτογραφίες του κάθε άρθρου.

Έτσι λοιπόν κλείνουμε με την ανάλυση του Widget NewsDetails και συνεχίζουμε με το τελευταίο κομμάτι της εφαρμογής μας το οποίο είναι η οθόνη των Podcasts.

3.7.15: Podcasts

Τελευταίο κομμάτι της εφαρμογής αποτελεί η οθόνη των Podcasts. Η οθόνη αυτή αποτελεί μια ψηφιακή βιβλιοθήκη για τους χρήστες έτσι ώστε όταν αυτοί θα εισέρχονται στην εφαρμογή να μπορούν να ακούσουν τα ηχητικά αρχεία των εκπομπών που ίσως να μην προλάβανε να ακούσουν ζωντανά ή ίσως θέλουν να ξαναεπισκεφτούν.

Ο κώδικας του Widget αποτελείται από την παρακάτω εικόνα:

```dart
import 'dart:async';
import 'package:flutter/material.dart';
import 'package:audioplayers/audioplayers.dart';
import 'package:flutter/foundation.dart';
import 'package:provider/provider.dart';
import 'package:sport_masters_v2/models/podcast.dart';
import 'package:sport_masters_v2/screens/podcasts/podcasts_carousel.dart';
import 'package:sport_masters_v2/services/database.dart';

class Podcasts extends StatefulWidget {
  @override
  _PodcastsState createState() => _PodcastsState();
}

class _PodcastsState extends State<Podcasts> {

  static final playerState = GlobalKey<_PlayerWidgetState>();

  static List<Podcast> _podcastUrls = [
    Podcast(
      audioUrl: "",
      image: "",
      title: ""
    )
  ];

  PlayerWidget playerWidget = PlayerWidget(
    key: playerState,
    url: _podcastUrls[0].audioUrl
  );

  @override
  Widget build(BuildContext context) {
    return StreamProvider<List<Podcast>>.value(
      value: DatabaseService().podcasts,
      child: Container(
        height: MediaQuery.of(context).size.height / 2,
        child: Column(
          children: [
            SizedBox(height: 20),
            PodcastsCarousel(
              (index, podcasts) {
                _podcastUrls = podcasts;
                playerState.currentState.url = _podcastUrls[index].audioUrl;
              }
            ), // PodcastsCarousel
            SizedBox(
              height: 100
            ), // SizedBox
            playerWidget
          ]
        ) // Column
      ) // Container
    ); // StreamProvider.value
  }
}
```

Εικόνα 61: Κώδικας podcasts.dart

Όσον αφορά τα imports το νέο ανάμεσα σε όλα αυτά είναι το package audioplayers. Το συγκεκριμένο plugin επιτρέπει την δημιουργία ενός Audio Player για Android και iOS, κάτι το οποίο θα χρησιμοποιήσουμε για την αναπαραγωγή των Podcasts από το Firebase. Όσον αφορά τον κώδικα και συγκεκριμένα το State αντικείμενο, οι παράμετροι του είναι:

- playerState: Μια Static final παράμετρος τύπου GlobalKey. Χρησιμοποιείται για τον χειρισμό του Audio Player τον οποίο θα δημιουργήσουμε
- _podcastUrls: Μια Static λίστα Podcast η οποία θα χρησιμοποιείται για την αποθήκευση των Podcasts

- playerWidget: Το PlayerWidget αντικείμενο το οποίο θα χρησιμοποιηθεί για την αναπαραγωγή των ηχητικών αρχείων

Η build μέθοδος επιστρέφει ένα StreamProvider μιας λίστας Podcast ο οποίος παίρνει την τιμή του από την getter παράμετρο podcasts του αντικειμένου DatabaseService. Το child αυτού είναι ένα Container το οποίο περιέχει ένα Column του οποίου τα childs είναι δύο SizedBox για την δημιουργία κενού ανάμεσα στα Widgets, ένα PodcastsCarousel Widget και το playerWidget.

Όσον αφορά το PodcastsCarousel για κάθε φορά που θα γίνεται αλλαγή στην σελίδα θα καλείται η callback με παραμέτρους τις index και podcasts και θα γίνεται η ανάθεση των διαθέσιμων podcasts στην παράμετρο _podcastUrls και θα ορίζουμε το url του αντικειμένου playerState ίσο με το αντίστοιχο url το οποίο βρίσκεται στο index του πίνακα.

3.7.15.1: PodcastsCarousel

Το PodcastsCarousel αποτελεί το Custom Carousel για τα Podcasts. Ο κώδικας του είναι ο εξής:

```dart
import 'package:carousel_slider/carousel_slider.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sport_masters_v2/models/podcast.dart';
import 'package:sport_masters_v2/screens/podcasts/podcasts_carousel_item.dart';

class PodcastsCarousel extends StatefulWidget {
  final Function(int, List<Podcast>) onCarouselPageChanged;

  PodcastsCarousel(this.onCarouselPageChanged, {Key key}) : super(key: key);

  @override
  _PodcastsCarouselState createState() => _PodcastsCarouselState();
}

class _PodcastsCarouselState extends State<PodcastsCarousel> {

  static Function(int, List<Podcast>) callback;

  static List<Podcast> _podcastItems = [];

  static CarouselOptions _carouselOptions = CarouselOptions(
    onPageChanged: (index, reason) {
      callback(index, _podcastItems);
    },
    autoPlay: false,
    scrollDirection: Axis.vertical,
    enlargeCenterPage: true,
    viewportFraction: 1.0,
    enableInfiniteScroll: false
  );

  void initState() {
    callback = widget.onCarouselPageChanged;
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    final podcasts = Provider.of<List<Podcast>>(context);

    for (var items in podcasts) {
      _podcastItems.add(
        Podcast(
          title: items.title,
          image: items.image,
          audioUrl: items.audioUrl
        )
      );
    }

    return CarouselSlider(
      items:
      _podcastItems.map((item) => PodcastsCarouselItem(
        title: item.title,
        imageUrl: item.image,
      )).toList(), // PodcastsCarouselItem
      options: _carouselOptions
    ); // CarouselSlider
  }
}
```

Εικόνα 62: Κώδικας podcastscarousel.dart

Όσον αφορά τις μεταβλητές του Widget περιέχεται μόνο μια Final Function η οποία επιστρέφει έναν int και μια λίστα τύπου Podcast και χρησιμοποιείται στον δομητή για την αρχικοποίηση.

Στην συνέχεια η κλάση του State αντικειμένου περιλαμβάνει ως προς τις μεταβλητές της τις παρακάτω:

- callback: Η μεταβλητή που θα χρησιμοποιηθεί για πρόσβαση την callback Function
- _podcastItems: Εδώ θα αποθηκεύονται τα Podcasts. Η μεταβλητή είναι μια λίστα τύπου Podcast
- _carouselOptions: Τα options του Carousel. Αυτό το οποίο έχει σημασία είναι η παράμετρος onPageChanged στην οποία όταν γίνει το γεγονός καλούμε την Callback συνάρτηση μας και παίρνουμε το index του Carousel και την αντίστοιχη λίστα με τα Podcasts.

Οι μέθοδοι του αντικειμένου περιλαμβάνουν την initState στην οποία γίνεται η ανάθεση της callback και την build η οποία τραβάει τα Podcasts με την χρήση μια Provider μεταβλητής και στην συνέχεια επιστρέφει ένα CarouselSlider στο οποίο με την μέθοδο map την μεταβλητής _podcastItems για κάθε item μέσα στην λίστα γίνεται δημιουργία ενός PodcastsCarouselItem και όλο αυτό μετατρέπεται σε μία λίστα με την μέθοδο toList. Το CarouselSlider έχει επίσης για Options την μεταβλητή _carouselOptions.

3.7.15.1.1: PodcastsCarouselItem

Το PodcastCarouselItem είναι ένα Custom Widget το οποίο αποτελεί το αντικείμενο του Carousel. Ο κώδικας του είναι αυτός που ακολουθεί:

```dart
import 'package:flutter/material.dart';

class PodcastsCarouselItem extends StatefulWidget {

  final String title;
  final String imageUrl;

  PodcastsCarouselItem({Key key, this.title, this.imageUrl}) : super(key: key);

  @override
  _PodcastsCarouselItemState createState() => _PodcastsCarouselItemState();
}

class _PodcastsCarouselItemState extends State<PodcastsCarouselItem> {

  static String url;

  @override
  void initState() {
    super.initState();
    url = widget.imageUrl;
  }

  @override
  void dispose() {
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Align(
      alignment: Alignment.center,
          child: Container(
        height: MediaQuery.of(context).size.height / 6,
        decoration: BoxDecoration(
          borderRadius: BorderRadius.all(Radius.circular(20)),
          color: Colors.blueAccent,
        ), // BoxDecoration
        margin: EdgeInsets.symmetric(
          horizontal: MediaQuery.of(context).size.width / 15
        ), // EdgeInsets.symmetric
        width: MediaQuery.of(context).size.width / 0.2,
        child: Align(
          alignment: Alignment.center,
          child: Row(
            crossAxisAlignment: CrossAxisAlignment.center,
            mainAxisAlignment: MainAxisAlignment.start,
            children: [
              FadeInImage.assetNetwork(
                placeholder: 'assets/images/icons8_soccer_ball_52px.png',
                image: url,
                width: MediaQuery.of(context).size.height / 6,
                height: MediaQuery.of(context).size.height / 6,
              ), // FadeInImage.assetNetwork
              SizedBox(
                width: MediaQuery.of(context).size.width / 30
              ), // SizedBox
```

Εικόνα 63: Κώδικας podcastscarouselitem.dart (1)

```dart
Flexible(
  child: Text(
    widget.title,
    style: TextStyle(
      color: Colors.white,
      fontWeight: FontWeight.bold

    ), // TextStyle
    maxLines: 2,
  ), // Text
) // Flexible
```

Εικόνα 64: Κώδικας podcastscarouselitem.dart (2)

Συγκεκριμένα, οι παράμετροι του αντικειμένου είναι δύο final String μεταβλητές η title και η imageUrl οι οποίες θα περιλαμβάνουν τον τίτλο του Podcast και το url της εικόνας αντίστοιχα. Στην κλάση του State αντικειμένου αυτό το οποίο γίνεται στην build μέθοδο είναι ότι επιστρέφεται ένα Row Widget το οποίο περιέχει μέσα του ένα FadeInImage Widget το οποίο επιτυγχάνει την φόρτωση της εικόνας με Fade In Animation, και ένα Text Widget το οποίο με το Flexible Widget ουσιαστικά προσαρμόζεται αυτόματα για να μην βγαίνει εκτός των ορίων του Parent Widget. Έτσι λοιπόν έχουμε και το τελικό Widget για την εφαρμογή.

Κλείνοντας αυτό το κεφάλαιο, κάναμε μια μεγάλη και αναλυτική περιγραφή του κομματιού της ανάπτυξης του πηγαίου κώδικα στο περιβάλλον που προσφέρει το Flutter. Μέσα από το κεφάλαιο αυτό κανείς θα μπορούσε να συμπεράνει πως ο κώδικας που δημιουργήθηκε χαρακτηρίζεται από μια αρχιτεκτονική η οποία βασίζεται σε μεγάλο βαθμό στο κομμάτι της διεπαφής του χρήστη. Τα Widgets, όντας ένα από τα εργαλεία-πυρήνας, μας οδηγεί στο συμπέρασμα ότι η διεπαφή χρήστη αποτελεί ένα από τα πρωτεύοντα ζητήματα που τίθονται προς επίλυση στο περιβάλλον του Flutter.

Κεφάλαιο 4ο: Συμπεράσματα

Η εφαρμογή αυτή ήταν ένας τρόπος για να μπορέσουμε να καταλάβουμε το πως ένα SDK μπορεί να δημιουργήσει ένα πλαίσιο για τους προγραμματιστές να λειτουργούνε με ευκολία και χωρίς κάποιο συμβιβασμό ως προς το πως μπορούν να αναπτύξουν μια εφαρμογή για Mobile συσκευές.

Η κανονικότητα της αγοράς εργασίας επιβάλει ότι θα χρειαστεί ένα σύνολο ατόμων για να μπορέσει να γίνει η ανάπτυξη μιας εφαρμογής και αυτό πρέπει να εφαρμόζεται σε κάθε πλατφόρμα καθώς είναι γνωστό ότι κάθε πλατφόρμα έχει τις δικές της μοναδικές ιδιομορφίες και ιδιαιτερότητες κάτι το οποίο μας λέει ότι οι εφαρμογές που θα πρέπει να αναπτυχθούν είναι τελείως διαφορετικές. Έτσι λοιπόν κανείς μπορεί να καταλάβει ότι σχετικά με τον τρόπο με τον οποίο θα αναπτυχθούν οι εφαρμογές τόσο από τα μέλη των ομάδων, όσο και από τις ομάδες μεταξύ τους, μπορούμε να συμπεράνουμε ότι κάτι τέτοιο πέρα απο χρονοβόρο, αποτελεί και μια δύσκολη διαδικασία, η οποία δεν φέρνει πάντα τα επιθυμητά αποτελέσματα. Αν όμως λύσεις όπως αυτή του Flutter έρθουν και ενσωματωθούν στην αγορά εργασίας και στον καθιερωμένο τρόπο με τον οποίο λειτουργεί η πραγματικότητα, τότε αυτό σίγουρα θα αποτελέσει μια ευκαιρία για τους επιχειρηματίες καθώς και τους προγραμματιστές να αναθεωρήσουν τον τρόπο με τον οποίο η όλη διαδικασία ανάπτυξης εφαρμογών για κινητά λειτουργεί. Κάτι τέτοιο σίγουρα δεν θα αποτελέσει μια εύκολη λύση και είναι σχεδόν σίγουρο ότι δεν θα μπορέσει να υιοθετηθεί ευθύς αμέσως καθώς μια νέα τεχνολογία όπως αυτή του Flutter σίγουρα δεν έχει δοκιμαστεί και δεν αποτελεί ένα σίγουρο βήμα μπροστά για να μπορέσει να γίνει η διαδικασία μεταφοράς και αλλαγής των ήδη υφιστάμενων προτύπων. Σίγουρα τα αρνητικά δεν αποκλείονται και είναι κάτι το οποίο δεν μπορεί να παραβλεφθεί, ωστόσο με την συνεχή υποστήριξη και εισαγωγή νέων εργαλείων καθώς και την βελτιστοποίηση των ήδη υπάρχοντων, κανείς θα μπορούσε να είναι αισιόδοξος και να πεί ότι το μέλλον του Flutter υπάρχει και σιγά σιγά ίσως γίνει και κομμάτι της πραγματικότητας της παγκόσμιας αγοράς.

Βασικό συμπέρασμα αυτού του Project είναι τελικά πως μέσω του Flutter ο οποιοσδήποτε έχει βασικές γνώσεις προγραμματισμού και μια αντίληψη του πως ακριβώς μπορεί να αναπτυχθεί μια διεπαφή χρήστη μπορεί να δημιουργήσει μια εφαρμογή η οποία να καλύπτει τις ανάγκες του τόσο για την πλατφόρμα Android όσο και για iOS, πολύ εύκολα και σε πολύ μικρό χρόνο όντας πλήρως ανεξάρτητος. Ένα τέτοιο σύνολο από εργαλεία για να γίνει ευρέως χρησιμοποιούμενο, χρειάζεται αρκετό καιρό και την κατάλληλη στήριξη. Έτσι λοιπόν φαίνεται πως το SDK αυτό πολύ πιθανόν να είναι κάτι το οποίο θα μας απασχολήσει στα επερχόμενα χρόνια.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

[1] Eric Windmill, Flutter in Action. Manning Publication Co, 2020

[2] Ed Freitas, Flutter Succinctly. Ebooks Series, 2019

Internet Site

[3] Flutter Official Website, Google, 2020. Available: https://flutter.dev

[4] Firebase Official Website, Google, 2020. Available: https://firebase.google.com/

[5] Pub.dev Official Website, Google, 2020. Available: https://pub.dev/

[6] Visual Studio Code Official Website, Microsoft, 2021. Available: https://code.visualstudio.com/

[7] Android Official Website, Google, 2021. Available: https://www.android.com/

[8] iOS Official Website, Apple, 2021. Available: https://www.apple.com/ios

[9] User Interface Design, Wikipedia, 2021. Available: https://en.wikipedia.org/wiki/User_interface_design

[10] Open Source, Wikipedia, 2021. Available: https://en.wikipedia.org/wiki/Open_source

[11] Flutter Architectural Overview, Google, 2021. Available: https://flutter.dev/docs/resources/architectural-overview

ΠΑΡΑΡΤΗΜΑ Α : ΕΓΚΑΤΑΣΤΑΣΗ FLUTTER

Όσον αφορά την υλοποίηση του Project, αυτή έγινε σε περιβάλλον Windows με την χρήση του Editor Visual Studio. Έτσι η πλατφόρμα στην οποία ως επι το πλείστον αναπτύχθηκε η εφαρμογή είναι σε Android. Αυτό όμως όπως αναφέρθηκε και προηγουμένως δεν εμποδίζει την ανάπτυξη και δημιουργία ενός τελικού προϊόντος σε iOS. Στην συνέχεια θα γίνει επίδειξη και των απαραίτητων βημάτων σε iOS ώστε να γίνει το build της εφαρμογής.

Για την εγκατάσταση του Flutter με βάση την επίσημη ιστοσελίδα χρειάζεται η εγκατάσταση του SDK στο οποίο μπορεί ο οποιοσδήποτε να αποκτήσει πρόσβαση είτε κατεβάζοντας το ως ZIP αρχείο είτε δημιουργώντας ένα Git Repository στον προσωπικό του υπολογιστή. Στην τρέχουσα περίπτωση έγινε χρήση του .zip αρχείου. Τα βήματα τα οποία πρέπει να γίνουν είναι:

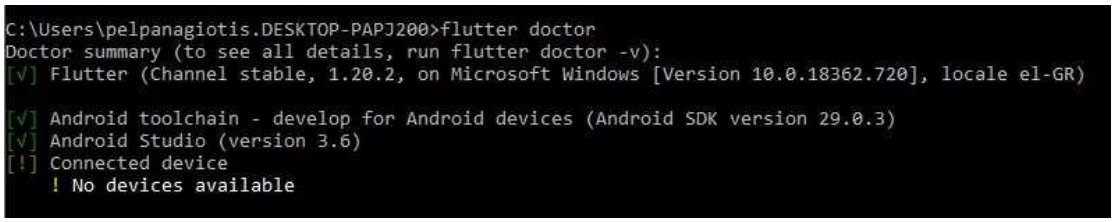
1) Κατέβασμα του .zip αρχείου flutter_windows_1.20.3-stable.zip.

2) Εξαγωγή των αρχείων από το .zip αρχείο

3) Τοποθέτηση των αρχείων στο επιθυμητό φάκελο στο σύστημα (π.χ. C:\flutter)

4) Ενημέρωση της μεταβλητής περιβάλλοντος PATH με την προσθήκη της διαδρομής αρχείου στην μεταβλητή αν αυτή ήδη υπάρχει ή την δημιουργία της και τον ορισμό της τιμής της με την συγκεκριμένη διαδρομή αρχείου.

Αφού λοιπόν έχουν γίνει σωστά τα παραπάνω βήματα το Flutter πλέον είναι διαθέσιμο για χρήση. Πριν γίνει το ξεκίνημα της ανάπτυξης μιας εφαρμογής συνίσταται να γίνεται το τρέξιμο της εντολής “flutter doctor”. Με την συγκεκριμένη εντολή ο χρήστης αν όλα τα κριτήρια για την χρήση του Flutter, όπως απαραίτητα IDE’s, συσκευές και άλλα, πληρούνται και μπορεί να ξεκινήσει η ανάπτυξη εφαρμογών.



Εικόνα 8: Υπόδειγμα αποτελέσματος εκτέλεσης της εντολής “flutter doctor”

Εικόνα : Υπόδειγμα αποτελέσματος εκτέλεσης της εντολής “flutter doctor”

Όταν όλα αυτά τα πεδία στο αποτέλεσμα έχουν στο αριστερό τους μέρος το σύμβολο του τικ, δηλαδή ότι όλα είναι εντάξει τότε είμαστε έτοιμοι να προχωρήσουμε στην έναρξη ανάπτυξης της εφαρμογής μας.