ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΔΙ.ΠΑ.Ε. - Πανεπιστημιούπολη Σερρών

Τμήμα Μηχανικών Πληροφορικής, Υπολογιστών και Τηλεπικοινωνιών

Κατεύθυνση Μηχανικών Λογισμικού

Farm Zone Manager

Διαχείριση Τμημάτων Αγροκτήματος

Ονοματεπώνυμο:	Αριθμός Μητρώου:	Επιβλέπων Καθηγητής:
Σίμων-Ουμπέρτο Μόσχος	**3741**	**Δρ. Θεόδωρος Λάντζος**

Πίνακας Περιεχομένων

Εισαγωγή Διπλωματικής Εργασίας

Η πτυχιακή αυτή ασχολείται με την μελέτη, σχεδίαση και δημιουργία ενός συστήματος το οποίο θα μπορέσει να υποστηρίξει την διαχείριση τμημάτων ενός αγροκτήματος.

Η γεωργία ακριβείας αποσκοπεί στην δυνατότητα παρακολουθησης και διαχειρισης φυτών όπως αυτά είναι μοναδικά στον αγρό. Όμως το κόστος υποστήριξης της τεχνολογίας, κρατά μεγάλο αριθμό αγροτών μακριά από την υιοθέτηση της τεχνολογίας. Προς σκοπό αυτό, η πτυχιακή αυτή προσπαθεί να δημιουργήσει ένα αυτόνομο πληροφοριακό σύστημα κινητής τηλεφωνίας το οποίο θα μπορέσει να δώσει υπηρεσίες διαχείρισης αγροκτήματος και των τμημάτων αυτών με ένα μοναδικό τρόπο.

Η διπλωματική αυτή παρουσιάζει την ανάλυση, σχεδίαση και υλοποίηση της εφαρμογής σε κινητό τηλέφωνο Android για χαρτογράφηση χωραφιών με σύγχρονη τεχνολογία και ανάπτυξη τμημάτων αυτών. Παρέχει επίβλεψη, χαρτογράφηση, ημερολογιακή υποστήριξη, αρχείο εργασιών και προγραμματισμό αυτών και εξαγωγή δεδομένων.

Στα επόμενα κεφάλαια παρουσιάζεται με λεπτομέρεια η ανάλυση των απαιτήσεων, ο τρόπος σχεδίασης, τα εργαλεία ανάπτυξης λογισμικού και το τελικό προϊόν μετα απο μεγάλους ελέγχους και εφαρμογή.

1. Θεματολογία Διπλωματικής Εργασίας και Ανάλυση των Απαιτήσεων

1.1. Θεματολογία Διπλωματικής Εργασίας

“Χαρτογράφηση αγροκτήματος με συμμετοχή θέσης, ορίων και παρεμβολής με χάρτες”

Σκοπός της πτυχιακής αυτής είναι σχεδίαση μια εφαρμογής σε κινητό τηλέφωνο η οποία θα επιτρέπει στους αγρότες να χαρτογραφούν τα αγροκτήματα τους. Η Εφαρμογή θα επιτρέπει τον χρήστη όχι μόνο να ορίσει την τοποθεσία στο google maps, αλλά να ορίσει τα σύνορα και την δυνατότητα παρεμβολής χαρτών με αγροτική υπηρεσία και ΤΟΕΒ.

Αποτέλεσμα αυτής της εφαρμογής θα είναι ένα προϊόν το οποίο θα δίνει μια ξεκάθαρη εικόνα στον αγρότη για το χαρτογραφικό υπόβαθρο του αγροκτήματος του, καθώς και η μεταφορά δεδομένων σε άλλα συστήματα όπως εξαγωγή χάρτη σε άλλες εφαρμογές, που θα του επιτρέψει την έγκυρη επίλυση προβλημάτων με οργανισμούς και διαχείρισης αγροκτήματος.

Η εφαρμογή θα επιτρέπει την έτοιμη εισαγωγή και εξαγωγή χωραφιών από το κτήμα καθώς και τις δυνατότητες διαίρεσης και πρόσθεσης γειτονικών κτημάτων.

1.2. Ανάλυση Απαιτήσεων Εφαρμογής

Ως πρώτο βήμα πρέπει να αναλύσουμε τι χρειάζεται το project για την σχεδίαση και υλοποίηση του. Θα χρειαστούμε:

- Ένα **ολοκληρωμένο προγραμματιστικό περιβάλλον (IDE)** για ανάπτυξη εφαρμογών σε πλατφόρμα Android.
- Ένα **πρόγραμμα επεξεργασίας γραφικών** για την δημιουργία γραφικών πόρων όπως εικονίδια ή διανυσματικά background.
- Ένα **γεωγραφικό πληροφοριακό σύστημα (GIS)** για την απεικόνιση χαρτών σε πλατφόρμες Android.
- Μια **τοπική βάση δεδομένων** για πλατφόρμες Android, όπου θα αποθηκεύονται τα δεδομένα του χρήστη.
- Βοηθητικές **βιβλιοθήκες ανάγνωσης και εγγραφής αρχείων** για πλατφόρμες Android, όπου θα βοηθάνε την προσκόμιση και εκτύπωση δεδομένων της εφαρμογής σε αρχεία.
- Μια **βιβλιοθήκη πράξεων γεωγραφικών συνόρων**, για την υλοποίηση των πράξεων ένωσης, τομής και διαφοράς μεταξύ δύο γεωγραφικά σύνορα.
- Μια **βιβλιοθήκη ημερολογίου** για πλατφόρμες Android, όπου θα παρουσιάζονται δεδομένα που έχουν σχέση με κάποιες ημερομηνίες.
- Μια **βιβλιοθήκη προβολής εικόνας** που χρησιμοποιεί gestures για επεξεργασία των διαστάσεων του, για την προβολή μιας εικόνας που να περιγραφεί τα σύνορα του αγροκτήματος σαν εικόνα επικάλυψης στων χάρτη.

1.3. Ανάλυση Απαιτήσεων Διαχείρισης Αγροκτημάτων

Έχοντας αναλύσει τις απαιτήσεις σχεδίασης και υλοποίησης της εφαρμογής, θα καταγράψουμε τις απαιτήσεις διαχείρισης αγροκτημάτων από τους γεωργούς ή από την ίδια εφαρμογή. Τα User Stories είναι τα εξής:

A. Σαν **<u>γεωργός</u>** θέλω να:

1. Δημιουργώ αγρούς με διάφορους τίτλους, να ορίζω τα γεωγραφικά σύνορα τους, να βάζω μερικές ετικέτες, κ.α.
2. Εισάγω γεωγραφικά σύνορα από ένα γεωγραφικό αρχείο.
3. Τροποποιώ ή να διαγράψω έναν αγρό.
4. Φιλτράρω αγρούς με βάση τις ετικέτες τους.
5. Εξάγω αγρούς σε ένα γεωγραφικό αρχείο.
6. Δημιουργώ τμήματα στα αγροκτήματα μου με διάφορους τίτλους, να ορίζω τα γεωγραφικά σύνορα τους, να βάζω μερικές ετικέτες, κ.α.
7. Τροποποιώ ή να διαγράψω ένα τμήματα αγροκτήματος.
8. Φιλτράρω τμήματα αγροκτημάτων με βάση τις ετικέτες τους.
9. Μπορώ να επαναφέρω έναν αγρό με τα επιμέρους τμήματα του από ένα ιστορικό γεγονός.
10. Δημιουργώ εργασίες που αφορούν έναν αγρό ή ένα τμήμα αγρού.
11. Τροποποιώ ή να διαγράψω μια εργασία.
12. Δημιουργώ κατηγορίες εργασιών.
13. Τροποποιώ ή να διαγράψω μια κατηγορία εργασιών.
14. Ορίζω κατηγορίες εργασιών στις εργασίες μου.
15. Φιλτράρω τις εργασίες του αγρότη με βάση τις κατηγορίες τους.
16. Βλέπω τις εργασίες μου σε ένα ημερολόγιο ή μία λίστα.
17. Βλέπω όλους τους αγρούς με τα επιμέρους τμήματα τους σε έναν χάρτη.
18. Βλέπω τις εργασίες πού αφορούν κάποιον αγρό ή τμήμα αγρού.
19. Εξάγω όλα τα δεδομένα σε ένα αρχείο Open XML.
20. Εισάγω δεδομένα από ένα αρχείο Open XML.

B. Σαν **<u>εφαρμογή</u>** θέλω να:

1. Διατηρώ ιστορικά γεγονότα που αφορούν τους αγρούς και τα επιμέρους τμήματα τους, μαζί με την ημερομηνία και των τύπο τις ενέργειας που έκανε ο **<u>γεωργός</u>**.
2. Ειδοποιώ των **<u>γεωργό</u>** για κάποια προγραμματισμένη εργασία.
3. Χωρίζω τους αγρούς και τα επιμέρους τμήματα τους ανά έτος δήλωσης.
4. Εισάγω τους αγρούς και τα επιμέρους τμήματα τους από την περσινή δήλωσης στην τωρινή χρόνια.

2. Το περιβάλλον εργασίας και σχεδίασης

Έχοντας αναλύσει τις λειτουργίες της εφαρμογής, ήρθε η ώρα να ρίξουμε μια ματιά στα εργαλεία που θα χρειαστούμε για την σχεδίαση και την υλοποίηση της εφαρμογής.

2.1. Περιβάλλον Εργασίας

Σε αυτή την ενότητα θα δούμε τα προγράμματα που χρησιμοποιήθηκαν για την σχεδίαση και υλοποίηση του project μας.

2.1.1. Android Studio

Το Android Studio είναι ένα δωρεάν ολοκληρωμένο προγραμματιστικό περιβάλλον IDE της Google και της JetBrains' IntelliJ IDEA, για ανάπτυξη εφαρμογών σε πλατφόρμες Android σε γλώσσες προγραμματισμού Java, C, C++ και Kotlin.

2.1.1.1. Υποδομή Project σε Android Studio

- **Αρχεία Res**, είναι τα αρχεία που περιγράφουν κύριος γραφικούς πόρους, όπως:
  - Εικόνες και διανυσματικά σχέδια.
  - UI Layout και Μενού.
  - Σταθερές μεταβλητές ή πίνακες τιμών όπως: διαστάσεις, θέματα και χρώματα, αλφαριθμητικά μαζί με τις μεταφράσεις τους, κ.α.
- **Αρχεία Κώδικα**, είναι τα αρχεία που περιλαμβάνουν των πηγαίο κώδικα της εφαρμογής σε C, C++, Java και Kotlin. Μπορούν να περιγράψουν την σχεδίαση ολόκληρης ή κομμάτι της οθόνης, υπηρεσίες παρασκηνίου εφαρμογής , μερικές λειτουργίες της εφαρμογής αλλά και μεθόδους, αντικείμενα και διεπαφές της εφαρμογής.
- **Αρχείο Manifest**, είναι το αρχείο που χρησιμοποιείται για την δήλωση αδειών, υπηρεσιών και οθονών της εφαρμογής.
  - Άδειες εφαρμογής, εδώ καταγράφουμε τις άδειες που θα ζητήσουμε από των χρήστη για να έχουμε πρόσβαση σε πόρους του συστήματος, όπως:
    - Ανάγνωση ή/και εγγραφή αρχείων στον αποθηκευτικό χώρο της συσκευής.
    - Άδεια επικοινωνίας μέσου Internet ή/και Bluetooth.
    - Χρήση τοποθεσίας συσκευής.
    - Ανάγνωση ή/και διαχείριση επαφών.
    - Ανάγνωση ή/και αποστολή μηνυμάτων
    - Ανάγνωση κατάστασης τηλεφωνικού δικτύου, κ.α.
  - Οθόνες εφαρμογής, εδώ δηλώνουμε τις οθόνες της εφαρμογής μαζί με τις παραμέτρους τους, αυτά μπορεί να είναι:
    - Ένα παράθυρο για Είσοδο και Εγγραφή λογαριασμού στην εφαρμογή μας.
    - Ένα παράθυρο ανάγνωσης αρχείου που θα ξεκινάει από το σύστημα όταν προσπαθήσει να ανοίξει κάποιο συγκεκριμένο τύπο αρχείου.
    - Ένα παράθυρο αποστολής ή/και παράδοσης μηνυμάτων
    - Ένα παράθυρο που να δουλεύει σαν καμβά για σχέδιο, κ.α.
  - Υπηρεσίες εφαρμογής, είναι λειτουργίες που να δουλεύουν στο παρασκήνιο όπως:
    - Συγχρονισμός βάσης δεδομένων.
    - Δημιουργία και επεξεργασία ειδοποιήσεων εφαρμογής στο παρασκήνιο.
    - Αναπαραγωγής μουσικής στο παρασκήνιο.
    - Αποστολή ή λήψη αρχείου σε server, κ.α.

2.1.1.2. Λειτουργίες και Frameworks στο Android Studio

Το Android Studio παρέχει έτοιμες λειτουργίες και frameworks που απλοποιούν την δουλειά σε έναν προγραμματιστή, μερικά από αυτά είναι:

- **Συμπλήρωση**, **ανακατασκευή** και **ανάλυσης** κώδικα στων code editor.
- **Εργαλεία Testing** που μας βοηθάνε για την δοκιμή και επαλήθευσης του κώδικα μας.
- **Ενσωματωμένες υπηρεσίες** όπως:
  - **Firebase and Cloud Services**, είναι ένα σύνολο υπηρεσιών και frameworks της Google και της Firebase, όπως:
    - Εκτέλεση μεθόδων μέσου server,
    - NoSQL βάση δεδομένων,
    - Αποθηκευτικό χώρο στο Google Cloud,
    - Υπηρεσίες Hosting μαζί με CDN,
    - Απομακρυσμένες ρυθμίσεις σε κάποιον χρήστη,
    - Google Analytics και Crashlytics,
    - Google, Twitter, Facebook, GitHub Authentication, κ.α.
  - **Github integration**, το android studio έχει από μόνο του την δυνατότητα ελέγχου και διαχείρισης εκδόσεων της εφαρμογής στο Github έχοντας μονό εγκατεστημένο το GIT.
  - **Layout Editor**, είναι ένας οπτικός επεξεργαστής αρχείων layout για την πιο εύκολη δημιουργία και επεξεργασία αρχείων διάταξης layout.
  - **Layout Viewer**, είναι μια ενότητα που μας παρουσιάζει πως θα φαίνεται ένα αρχείο layout σε διάφορες οθόνες και συσκευές.
  - **Layout Inspector**, είναι μια ενότητα που μας παρουσιάζει αναλυτικά όλα τα αντικείμενα που βρίσκονται στην οθόνη της συσκευής, μαζί με όλες τις διαστάσεις, ρυθμίσεις άλλα και την ιεραρχία τους.
  - **Device Logcat**, είναι μια ενότητα για την προβολή των καταγεγραμμένων μηνυμάτων του συστήματος και προβολή σφαλμάτων εφαρμογών (logcat).
  - **Profiler Inspector**, για των έλεγχο πόρων της συσκευής και απόδοσης της εφαρμογής.
  - **Database και Background Service Inspector**, είναι μια ενότητα που μας παρουσιάζει σε μια καρτέλα την κατάσταση των υπηρεσιών στο παρασκήνιο της εφαρμογής μας μαζί με το αποτέλεσμα τους και σε άλλη καρτέλα την τοπική βάση δεδομένων της εφαρμογής μας.
  - **Translations Editor**, είναι μια ενότητα για την υλοποίηση κάποιας μετάφρασης στην εφαρμογή μας
  - **Android Emulator**, είναι ένα emulator που μπορεί να πάρει μορφή οποιασδήποτε συσκευής υποστηρίζει android μαζί με όλες τις εκδόσεις του android, όπως: κινητά, tablet, smartwatch κ.α.

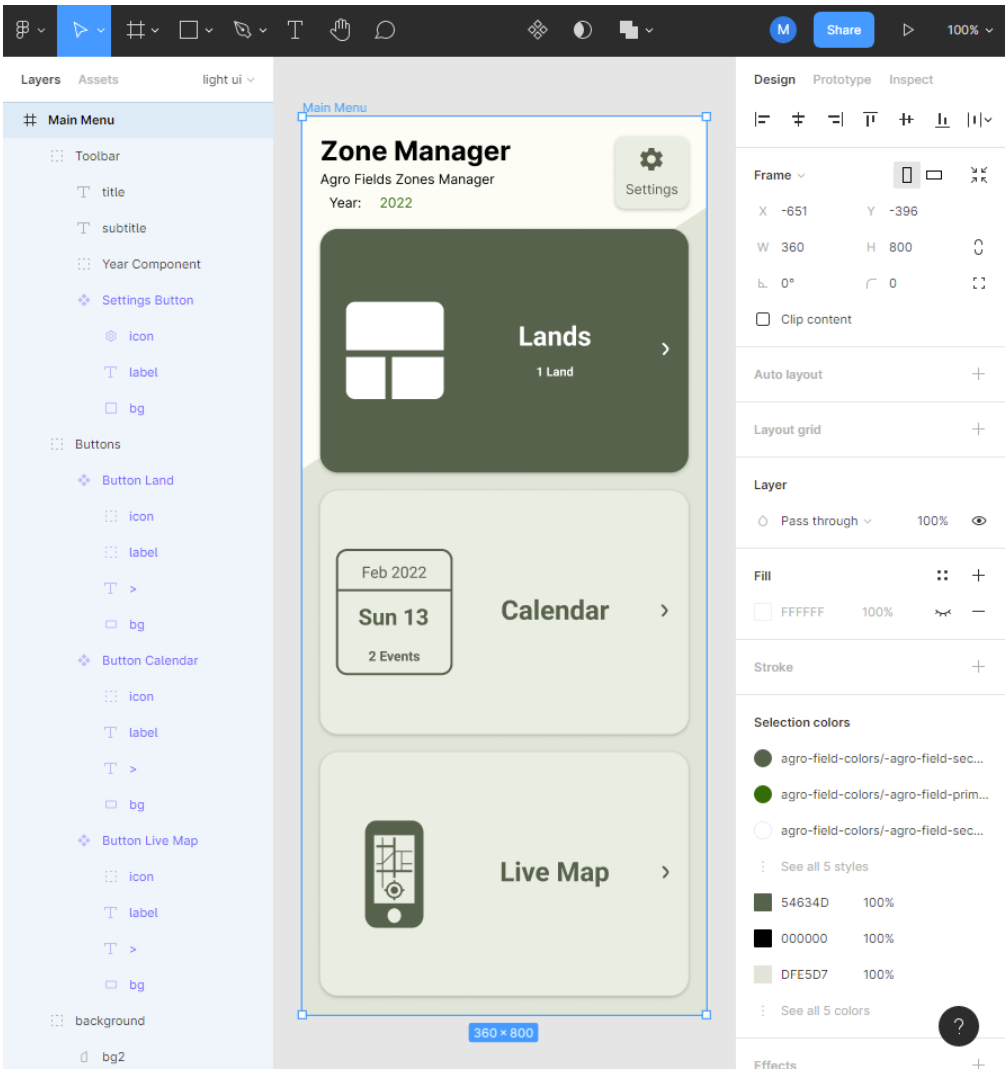
Για κλείσιμο, το Android Studio είναι ένα πολύ καλό IDE για ανάπτυξη εφαρμογής Android, από την απλή υποδομή του project, στην πληθώρα υπηρεσιών που παρέχει, είναι ο λόγος που επιλέχθηκε για το συγκεκριμένο project.

Link για λεπτομέρειες του Android Studio: https://developer.android.com/studio

2.1.2. Figma

Το Figma είναι μία δωρεάν εφαρμογή για επεξεργασίας διανυσματικών γραφικών και πρωτοτύπων, το οποίο χρησιμοποιείται και για σχεδίαση διεπαφών χρηστών (UI/UX). Παρέχει εύκολη προβολή χαρακτηριστικών των διανυσματικών αντικειμένων που δημιουργεί όπως διαστάσεις αντικειμένων, απλά ή σύνθετα χρώμα, ρυθμίσεις κειμένων και άλλα. Ενα τετοιο πρόγραμμα θα μας βοηθήσει στο στάδιο της σχεδίασης αλλά και στην δημιουργία εικονιδίων στην εφαρμογή.

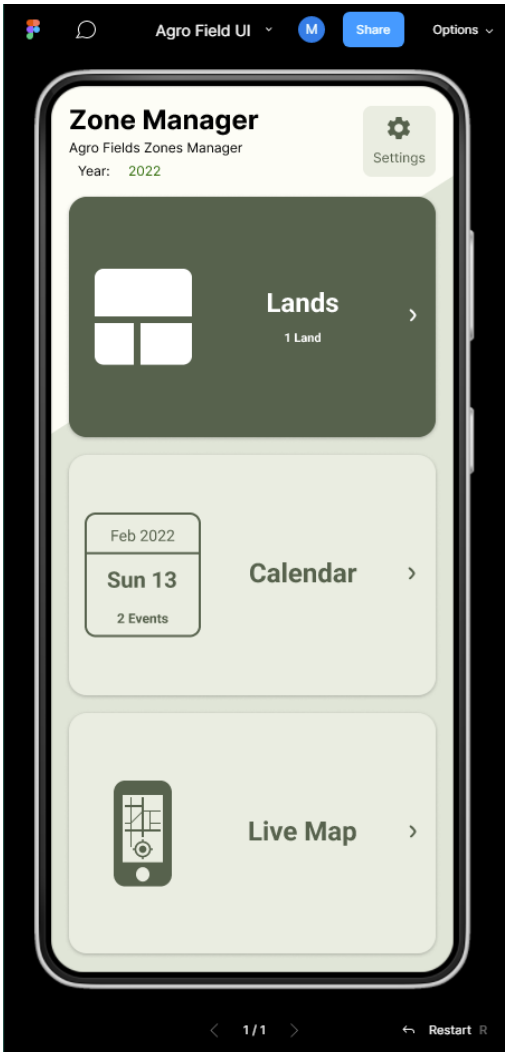
2.1.2.1. Διεπαφή Σχεδίασης UI του Figma



- Κύριο Μενού: το κύριο μενού που βρίσκεται στην πάνω πλευρά της οθόνης παρέχει όλα τα εργαλεία σχεδίασης ή επεξεργασίας διανυσματικών αντικειμένων.
- Αριστερό Μενού: στο αριστερό μέρος της οθόνης έχουμε τις καρτέλες:
  - **Layers**, που δείχνει την ιεραρχία των διανυσματικών αντικειμένων.
  - **Assets**, που παρέχει όλα τα διαδραστικά αντικείμενα του project.
  - Την επιλογή **σελίδας**, για να έχουμε τα σχέδια μας καλύτερα οργανωμένα.
- Καμβάς: εδώ έχουμε οπτική προβολή των διανυσματικών αντικειμένων που δημιουργήσαμε.
- Δεξί Μενού: στο δεξί μέρος της οθόνης έχουμε τις ακόλουθες καρτέλες:

- **Καρτέλα Design**, εδώ υπάρχουν όλες οι γραφικές ιδιότητες και λειτουργίες των διανυσματικών αντικειμένων, μερικές από αυτές είναι:
  - Αποστάσεις και διαστάσεις αντικειμένων.
  - Χρώματα και περιγράμματα αντικειμένων.
  - Εφέ και στιλ αντικειμένων.
  - Εξαγωγή αντικειμένων σε PNG, JPG, SVG, PDF.
- **Καρτέλα Prototype**, εδώ υπάρχουν όλες οι ιδιότητες και λειτουργίες των διανυσματικών αντικειμένων που σχετίζονται με την λειτουργία παρουσίασης Prototype του Figma, εδώ μπορούμε να ορίσουμε των τύπο της συσκευής παρουσίασης και τις αλληλεπιδράσεις του χρήστη με τα αντικείμενα.
- **Καρτέλα Inspect**, εδώ υπάρχουν όλες οι λεπτομέρειες εμφάνισης των διανυσματικών αντικειμένων στον καμβά και τις λεπτομέρειες για μετατροπή αντικειμένων σε κώδικα εμφάνισης, όπως: Android Layout View, iOS View, Κώδικα CSS.

2.1.2.2. Λειτουργία Prototype



Αυτή η λειτουργία μας κάνει μια παρουσίαση όλων των αντικειμένων που δημιουργήσαμε στο στάδιο σχεδίασης, με το περίγραμμα και την ανάλυση της συσκευής που επιλέξαμε στην καρτέλα Prototype.

Πλην της παρουσίασης των αντικειμένων, σε αυτή την λειτουργιά μπορούμε να αλληλεπιδράσουμε με τα αντικείμενα που έχουμε προσθέσει κάποιου τύπου αλληλεπίδρασης με των χρήστη. Αυτό είναι πολύ χρήσιμη για δοκιμαστικούς λόγους, διότι χωρίς να γράψουμε κώδικα μπορούμε να δοκιμάσουμε την εργονομικότητα της εφαρμογής μας.

Μπορούμε να μπούμε στην λειτουργία Prototype πατώντας το κουμπί present στο κύριο μενού (με εικονίδιο το play).

Link για λεπτομέρειες του Figma: https://www.figma.com

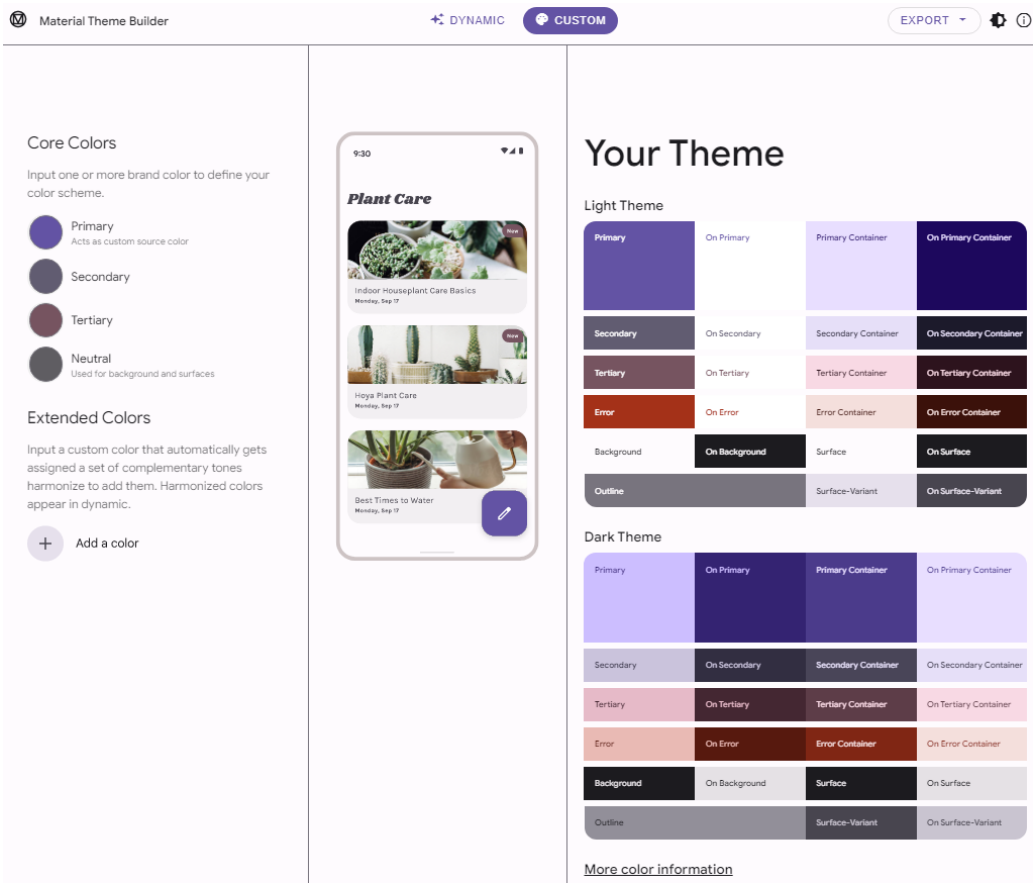
2.1.3. Material Theme Builder

Το Material Theme Builder είναι ένα εργαλείο για δημιουργία χρωματικών θεμάτων για εφαρμογές, σύμφωνα με τους κανόνες του Material IO[1]. Λόγου της ενσωμάτωσης των Material Components στο Android Studio θα χρειαστεί να δημιουργήσουμε ένα Material Theme για να προβάλετε σωστά αυτά τα components και να ακολουθούν τους κανόνες του Material IO. Το Material Theme Builder παρέχετε σε εφαρμογή ιστού και σε plugin του Figma.

2.1.3.1. Εφαρμογή ιστού

Μπαίνοντας στο “Material Theme Builder on Web” και πατώντας την επιλογή “CUSTOM” στο μενού θα μπορέσουμε να:

1. Ορίσουμε τα επιθυμητά χρώματα της εφαρμογής μας στις ανάλογες θέσεις[2].
2. Να δούμε τα επιθυμητά χρώματα μας σε “Dark Theme” πατώντας των Ήλιο στο μενού.
3. Να κάνουμε εξαγωγή το θέμα μαζί με την χρωματολογία του σε αρχεία:
   a. Jetpack Compose (Theme.kt).
   b. Android Views (XML).
   c. Flutter (Dart).
   d. Web (CSS).
   e. Material Tokens (DSP).



Εικόνα 1
Διεπαφή εφαρμογής ιστού του Material Theme Builder

[1] Δείτε την ενότητα: 2.2.1. Material IO
[2] Λεπτομέρειες των χρωμάτων είναι στην ενότητα: 3.2. Σχεδίαση Γραφικών Πόρων τής Εφαρμογής

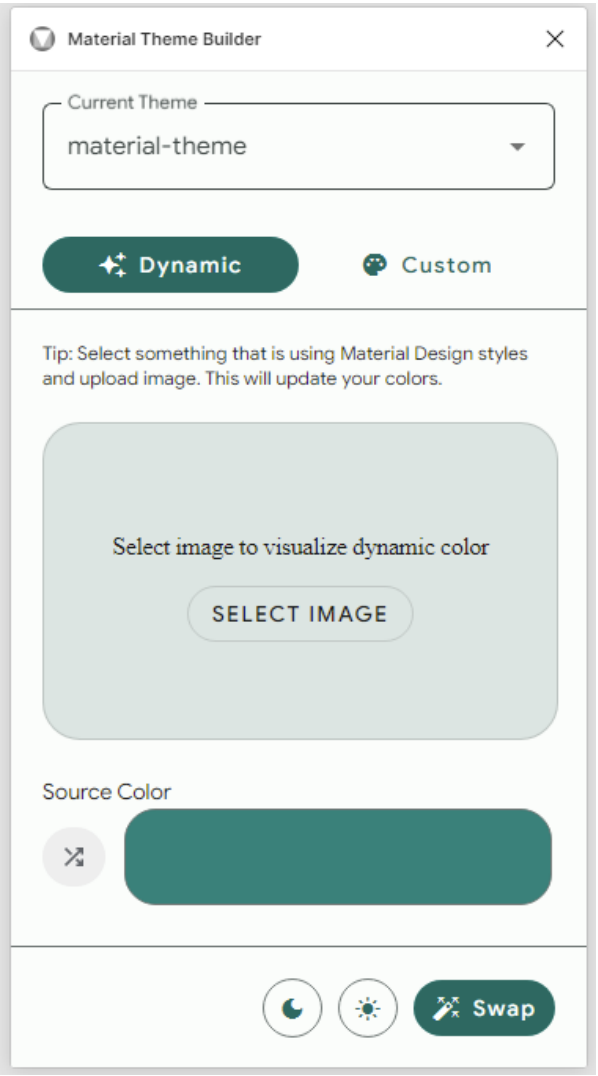
2.1.3.2. Figma Plug-In

Προσθέτοντας το Plugin στο Figma θα μπορούμε πλέον να παράγουμε Material IO Theme κατευθείαν στο Figma.

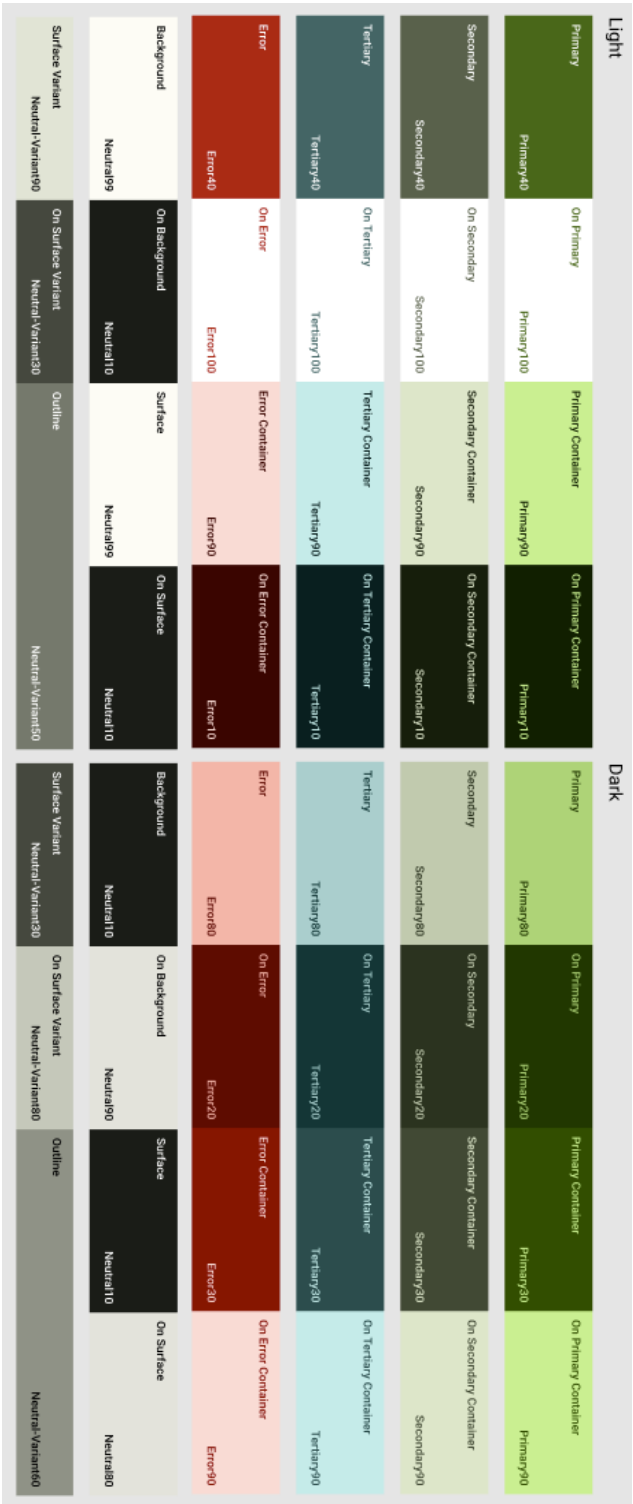
Το Plugin δημιουργεί του έξης πίνακες:

1. Τονική παλέτα χρωμάτων.
2. Παλέτα χρωμάτων του “Light Theme”.
3. Παλέτα χρωμάτων του “Dark Theme”.
4. Τονικές αποχρώσεις με βάση το elevation.
5. Κανόνες τυπογραφίας.

Όταν δημιουργήσουμε τους χρωματικούς πινάκες θα παρατηρήσουμε ότι πλέον όλες οι μεταβλητές περάστηκαν αυτόματα στο Color Style και στο Text Style του project μας.



Εικόνα 2
Αναδυόμενο παράθυρο του Plugin



Εικόνα 3
Μερικούς πίνακες του Plugin

Link του Material Theme Builder: https://material.io/blog/material-theme-builder

2.2. Βιβλιοθήκες Εφαρμογής

Σε αυτή την ενότητα θα δούμε τις βιβλιοθήκες πού θα χρειαστούμε στην εφαρμογή μας.

2.2.1. Material IO

Το Material IO είναι ένα σύνολο σχεδιαστικών κανόνων και components που δημιουργήθηκε από την Google για Android, iOS, Flutter και εφαρμογές ιστού. Στην εφαρμογή θα χρησιμοποιηθούν τα Material Components όπως το Floating Action Button (FAB) ή το Material Card View για των σχεδιασμών αρχείων Layout, έχοντας έτσι πιο όμορφες και προσαρμόσιμες οθόνες ανάλογα με τις απαίτησης αυτής της οθόνης.

Link για λεπτομέρειες του Material IO: https://m3.material.io

2.2.2. Room Database

Το Room Database είναι μια βιβλιοθήκη σχεσιακής χαρτογράφησης αντικειμένων ORM (Object Relational Mapping) σε τοπική βάση δεδομένων για Android που βασίζεται σε SQLite. Για λεπτομέρειες υλοποίησης τοπικής βάσης δεδομένων στην εφαρμογή δείτε την ενότητα: 4.2.1. Room Database.

Link του Room Database: https://developer.android.com/training/data-storage/room

2.2.3. Maps SDK for Android

Το Maps SDK για πλατφόρμες Android είναι γεωγραφικό πληροφοριακό σύστημα (GIS) που χρησιμοποιεί δεδομένα του Google Maps. Πάνω σε αυτούς τους χάρτες μπορούμε να δημιουργήσουμε σημεία, κύκλους, σχήματα ή εικόνες επικάλυψης αλλά και να αλλάξουμε την μορφή του χάρτη σε δορυφορικό χάρτη, χάρτη δρόμων ή σε δικό μας προσαρμοσμένο χάρτη.

Link για λεπτομέρειες σχετικά με το Maps SDK:
https://developers.google.com/maps/documentation/android-sdk

2.2.4. ArcGIS Runtime API

Το ArcGIS Runtime API είναι ένα API και ένα σύνολο βιβλιοθηκών που χρησιμοποιούνται για δημιουργία εφαρμογών χαρτογράφησης GIS. Μια λειτουργία αυτής της βιβλιοθήκης είναι η πράξεις συνόλων γεωγραφικών σημείων όπως η ένωση, η τομή και η διαφορά συνόλων. Για των τρόπο υλοποίησης αυτής της λειτουργίας στην εφαρμογή, δείτε την ενότητα: 4.1.3. Πράξεις Γεωγραφικών Συνόρων.

Link για λεπτομέρειες του ArcGIS Runtime API: https://developers.arcgis.com/documentation

2.2.5. Navigation Component

Το Navigation Component είναι μια βιβλιοθήκη που βοηθάει την πλοήγηση και τις αλληλεπιδράσεις του χρήστη με την εφαρμογή. Χρησιμοποιήθηκε για την πλοήγηση οθονών στην εφαρμογή, δείτε στις ενότητες: 4.3.9. Αρχεία Navigation και 4.4.1. Navigation Component.

Link του Navigation Component: https://developer.android.com/guide/navigation

2.2.6. Firebase Crashlytics

Το Firebase Crashlytics είναι ένα ελαφρύ πρόγραμμα αναφοράς σφαλμάτων σε πραγματικό χρόνο που βοηθάει την παρακολούθηση και επιδιόρθωση σφαλμάτων των χρηστών στις εφαρμογές μας. Χρησιμοποιήθηκε για την παρακολούθηση σφαλμάτων όταν ο χρήστης χρησιμοποιεί την εφαρμογή απομακρυσμένα.

Link για λεπτομέρειες του Firebase Crashlytics: https://firebase.google.com/docs/crashlytics

2.2.7. Java ESRI Shapefile Reader by olenus

Το Java ESRI Shape File Reader είναι μια βιβλιοθήκη εισαγωγής δεδομένων από αρχεία τύπου Shapefile για Java εφαρμογές. Χρησιμοποιήθηκε για την προσκόμιση λιστών γεωγραφικών σημείων από αρχεία τύπου .shp στην εφαρμογή, δείτε την ενότητα: 4.1.1.4. Αναλυτής Αρχείων Shapefile.

Link του Java ESRI Shapefile Reader: https://sourceforge.net/projects/javashapefilere

2.2.8. JDOM2 Library

Το JDOM2 είναι μια βιβλιοθήκη ανοιχτού κώδικα βασισμένο σε Java για ανάλυση XML αρχείων. Χρησιμοποιήθηκε για την προσκόμιση λιστών γεωγραφικών σημείων από αρχεία τύπου KML και GML στην εφαρμογή, δείτε την ενότητα: 4.2.4. Εισαγωγή και Εξαγωγή Αρχείων XML.

Link για λεπτομέρειες του JDOM2 Library: http://www.jdom.org/downloads/docs.html

2.2.9. Proj4J by OSGeo Library

Το Proj4J είναι μια βιβλιοθήκη Java για τη μετατροπή συντεταγμένων από ένα γεωγραφικό σύστημα συντεταγμένων σε άλλο. Χρησιμοποιήθηκε για την μετατροπή των συντεταγμένων που βρίσκονται στα γεωγραφικά αρχεία σε συντεταγμένες EPSG:4326 (WGS84) που υποστηρίζει το Maps SDK, δείτε την ενότητα: 4.1.2. Μετατροπή Συστημάτων Συντεταγμένων.

Link για λεπτομέρειες του Proj4J: https://trac.osgeo.org/proj4j

2.2.10. Apache POI Library

Η βιβλιοθήκη Apache POI παρέχει υποστήριξη εισαγωγής και εξαγωγής δεδομένων από έγγραφα Open XML για εφαρμογές Java. Χρησιμοποιήθηκε για την εισαγωγή και εξαγωγή δεδομένων σε αρχεία .xls και .xlsx ως backup αρχεία., δείτε την ενότητα: 4.2.6. Εισαγωγή και Εξαγωγή Αρχείων Open XML Sheet.

Link για λεπτομέρειες του Apache POI: https://poi.apache.org

2.2.11. CalendarView by kizitonwose

Το CalendarView ειναι μια βιβλιοθήκη ημερολογίου ανοιχτού κώδικα για Android Views (XML) σχεδιασμένη από των kizitonwose. Χρησιμοποιήθηκε για των σχεδιασμό της οθόνης “Ημερολογίου Συμβάντων”.

Link του CalendarView με παραδείγματα: https://github.com/kizitonwose/CalendarView

2.2.12. Android Collage Views by thuytrinh

Το Android Collage Views είναι μια βιβλιοθήκη για επεξεργασία διαστάσεων εικόνων με Gesture. Χρησιμοποιήθηκε για την επεξεργασία overlay εικόνας πάνω από των χάρτη.

Link του Android Collage Views με παράδειγμα:
https://github.com/thuytrinh/android-collage-views

3. Σχεδιασμός της Εφαρμογής

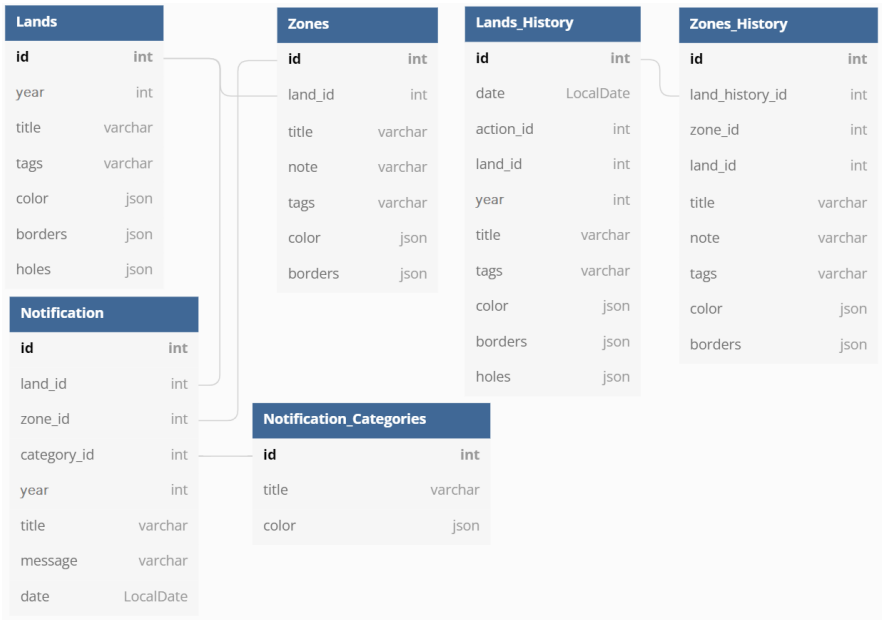
Σε αυτή την ενότητα θα μάθουμε το δεύτερο στάδιο ανάπτυξης εφαρμογών, την **σχεδίαση**. Στο στάδιο της σχεδιάσεις ξεκινάμε και σχεδιάζουμε πως επιθυμούμε την εφαρμογή μας, όπως σχέδια οθονών και σχέδια της βάσης δεδομένων. Εδώ θα μάθουμε πως να χρησιμοποιήσουμε το Figma μαζί με το plug in Material Theme Builder για να δημιουργήσουμε πρότυπα των οθονών τις εφαρμογής ,επίσης πως έγινε ο σχεδιασμός της βάσης δεδομένων της εφαρμογής.

3.1. Σχεδίαση Βάσης Δεδομένων τής Εφαρμογής

Σε αυτό το στάδιο σχεδιάζουμε την βάση δεδομένων της εφαρμογή. Για αρχή θα δούμε τι πρέπει να αποθηκεύσουμε στην εφαρμογή μας:

- ➔ Οντότητα **Χωραφιού**: Αυτή η οντότητα θα περιγράφει ένα αγρόκτημα. Τα πεδία τού θα είναι: ένα τίτλος, ένα χρώμα, τις ετικέτες του, τα σύνορα, τα κενά και ένα έτος δήλωσης.
- ➔ Οντότητα **Ζώνης**: Αυτή η οντότητα θα περιγράφει ένα τμήμα του αγροκτήματος, θα σχετίζεται με την οντότητα χωραφιού. Τα πεδία τού θα είναι: ένας τίτλος, ένα χρώμα, τις ετικέτες του, τα σύνορα.
- ➔ Οντότητα **Ιστορικό Χωραφιού**: Αυτή η οντότητα θα περιγράφει ένα στιγμιότυπο ενός χωραφιού. Τα πεδία του θα είναι ίδια με την οντότητας χωραφιού άλλα επίσης θα παρέχει την ημερομηνία της αλλαγής και το είδος της αλλαγής.
- ➔ Οντότητα **Ιστορικό Ζώνης**: Αυτή η οντότητα θα περιγράφει ένα στιγμιότυπο ενός τμήματος του αγροκτήματος, θα σχετίζεται με την οντότητα ιστορικό χωραφιού. Τα πεδία του θα είναι ίδια με την οντότητας ζώνης.
- ➔ Οντότητα **Κατηγορία Συμβάντος**: Αυτή η οντότητα θα περιγράφει τις κατηγορίες των συμβάντων. Τα πεδία του θα είναι: ο τίτλος της κατηγορίας, το χρώμα της κατηγορίας.
- ➔ Οντότητα **Συμβάντος**: Αυτή η οντότητα θα περιγράφει ένα γεγονός στο αγρόκτημα ή σε ένα τμήμα του αγροκτήματος, θα σχετίζεται με τις οντότητες χωραφιού, ζώνης και κατηγορίες συμβάντων. Τα πεδία του θα είναι: η ημερομηνία του συμβάν, ο τίτλος του συμβάν, μια περιγραφή του συμβάν και ένα πεδίο έτους δήλωσης.

Από τα παραπάνω προκύπτουν οι εξής πίνακες:

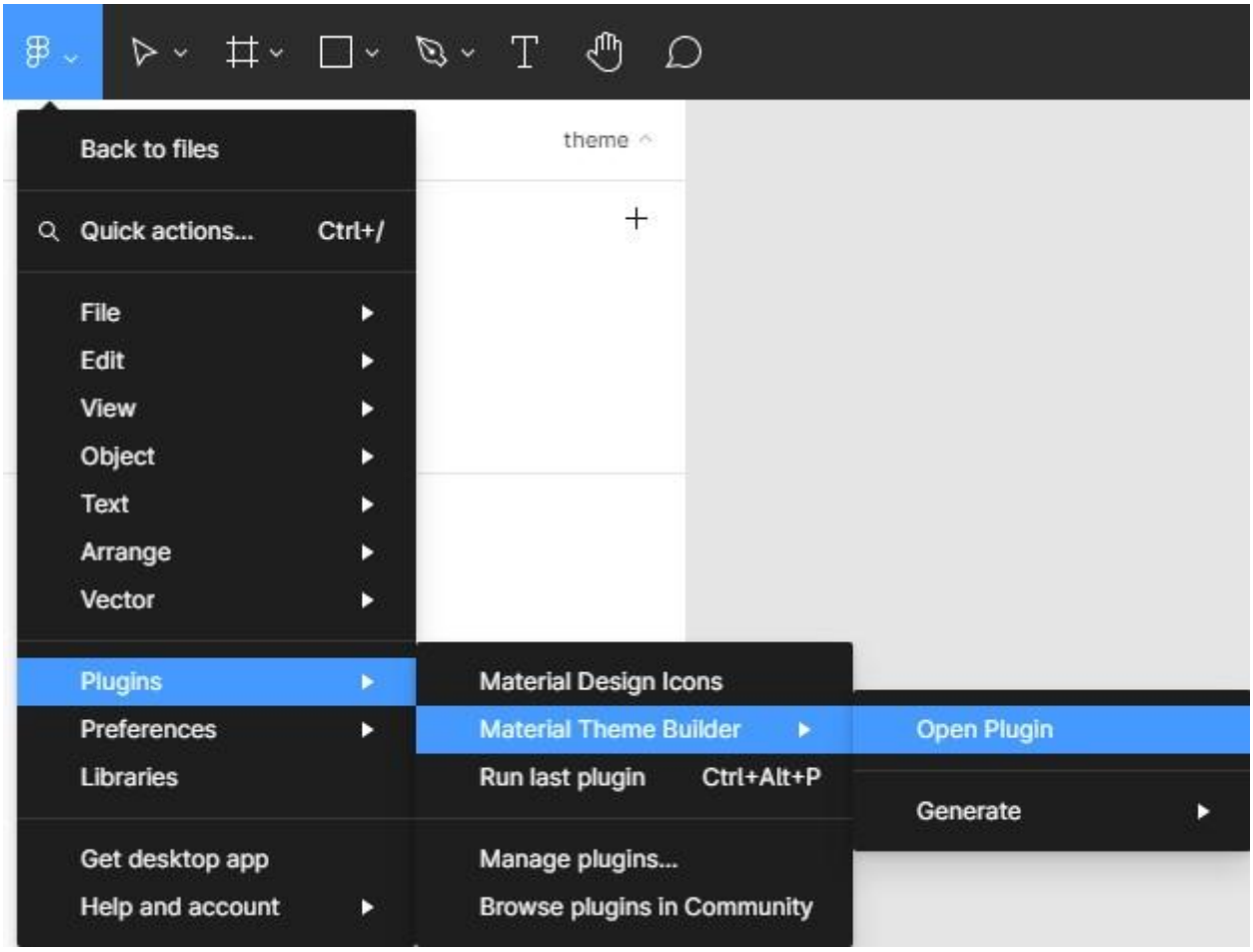


3.2. Σχεδίαση Γραφικών Πόρων τής Εφαρμογής

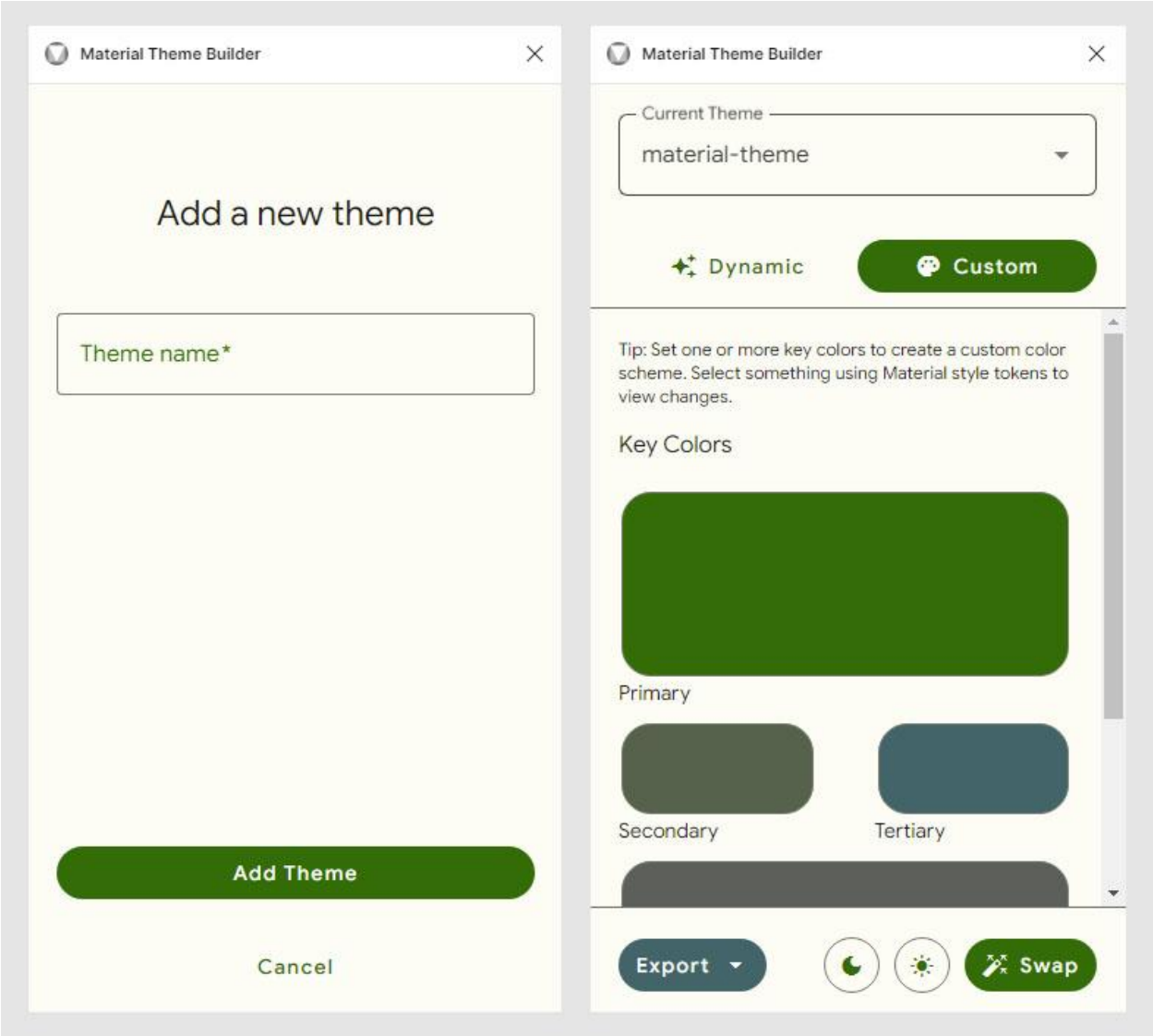
Προηγουμένως στην ενότητα 2.1.2. Figma μιλήσαμε για τις λειτουργίες του Figma και για το plugin Material Theme Builder, τώρα θα δούμε πως μπορούν σχεδιάσουμε τα πρότυπα των οθονών μας για να έχουν μια πιο καθαρή εικόνα όταν φτάσουμε στον σχεδιασμό αρχείων Layout, να μπορούμε να δημιουργήσουμε δικά μας εικονίδια και ο τρόπος δημιουργίας θέματος Material.

1. Για αρχή θα ανοίξουμε ένα νέο Project.

2. Θα τρέξουμε το plugin Material Theme Builder για να μας δημιουργήσει το χρωματικό θέμα της εφαρμογής. Πατάμε την επιλογή **Main Menu** στο κύριο μενού, πατάμε **Plugins**, επιλέγουμε το **Material Theme Builder** και τέλος πατάμε το **Open Plugin**.

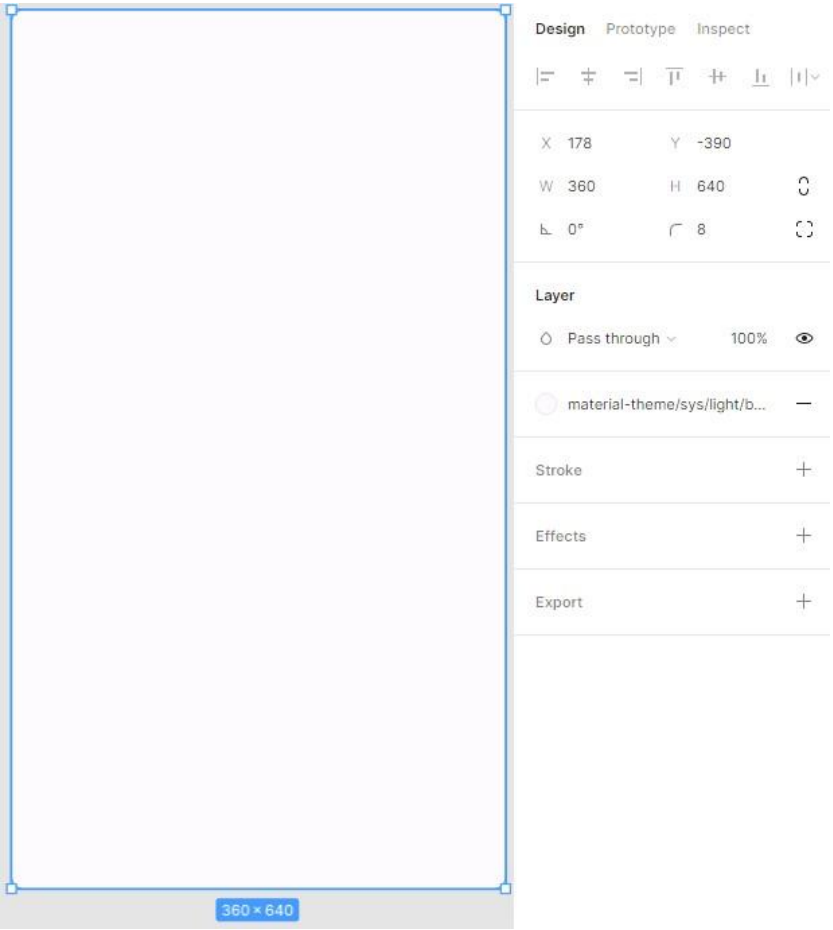


3. Πατώντας στο Current Theme μπορούμε να επιλέξουμε την επιλογή "+ Add new theme" για δημιουργία νέου θέματος, ορίζουμε των τίτλο του θέματος και έπειτα το επιλέγουμε στο Current Theme. Τώρα μπορούμε να του ορίσουμε τα χρώματα που επιθυμούμε.

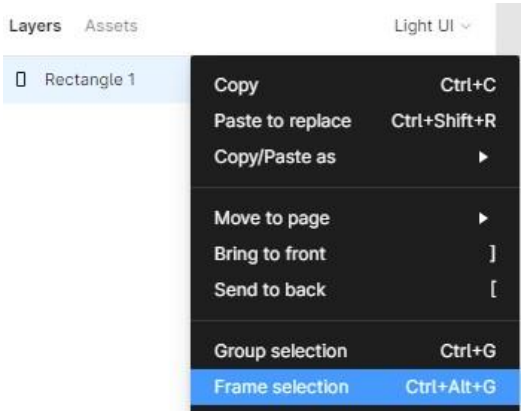


4. Θα επιλέξουμε μια νέα σελίδα από το αριστερό μενού και θα την ονομάσουμε "Light UI".

5. Τώρα θα φτιάξουμε ένα ορθογώνιο για φόντο επιλέγοντας το εργαλείο Rectangular, είτε πατώντας την επιλογή σχημάτων από το κύριο μενού είτε πατώντας το πλήκτρο R στο πληκτρολόγιο. Θα του ορίσουμε σαν μέγεθος τις διαστάσεις της συσκευής Android Small (360x640) της ενότητας Prototype και σαν Fill color θα πατήσουμε τις τέσσερις (4) τελείες, θα πάμε στην κατηγορία “material-theme/sys/light” και κάνοντας hover με το ποντίκι πάνω στα χρώματα θα ψάξουμε το χρώμα “background” και το επιλέγουμε.



6. Στο αριστερό μενού στην ενότητα Layout θα επιλέξουμε το ορθογώνιο που δημιουργήσαμε και πατώντας δεξί κλικ σε αυτό θα επιλέξουμε την ενότητα “Frame selection” για να μετατραπεί το ορθογώνιο σε ένα Frame, αυτό θα είναι μια νέα οθόνη.



7. Τώρα μέσα σε αυτό το αντικείμενο Frame θα σχεδιάσουμε την επιθυμητή οθόνη προσθέτοντας κείμενα, εικόνες ή σχήματα που προσομοιάζουν τα android components.
   Η χρηματολογία των αντικειμένων πρέπει να είναι ως εξής:

   ➢ **Primary colors**, επιλέγονται για την ικανότητά τους να εκπροσωπούν την επωνυμία της εφαρμογής μας, συνήθως βρίσκονται σε αντικείμενα όπως **app bars** και **κουμπιά**.
   Οι υποκατηγορίες του είναι:

   - **Primary** είναι το χρώμα που χρησιμοποιούμε για φόντο όταν θέλουμε να τονίσουμε το περιεχόμενο ενός αντικειμένου, όπως χρώμα σε ένα κείμενο η το fill color ενός διανυσματικού σχεδίου πάνω σε κουμπί.
   - **On Primary** είναι το χρώμα που χρησιμοποιούμε για περιεχόμενο των αντικειμένων που χρησιμοποιούν το χρώμα **Primary στο φόντο τους**.
   - **Primary Container** είναι το χρώμα που χρησιμοποιούμε για όταν θέλουμε να τονίσουμε το φόντο ενός αντικειμένου, όπως το χρώμα σε ένα κουμπί.
   - **On Primary Container** είναι το χρώμα που χρησιμοποιούμε για το περιεχόμενο των αντικειμένων που χρησιμοποιούν το χρώμα **Primary Container στο φόντο τους**.

   ➢ **Secondary colors** και **Tertiary colors**, χρησιμοποιούνται συχνά για να τονίσουμε κάποια αντικείμενα, συνήθως βρίσκονται σε αντικείμενα όπως **κουμπιά δράσης** ή **επιλογές**. Η υποκατηγορίες τους είναι ίδιες με το primary color.

   ➢ **Error colors**, υποδεικνύει σφάλματα σε στοιχεία, όπως πεδία κειμένου. Η υποκατηγορίες του είναι ίδιες με το primary color.

   ➢ **Background color**, βρίσκεται πίσω από περιεχόμενο με δυνατότητα κύλισης άλλα και σαν κύριο φόντο της εφαρμογής.

   - **Background** είναι το χρώμα που χρησιμοποιούμε στο φόντο.
   - **On Background** είναι το χρώμα που χρησιμοποιούμε στο περιεχόμενο που βρίσκεται πάνω στο φόντο, π.χ. κάποιο κείμενο.

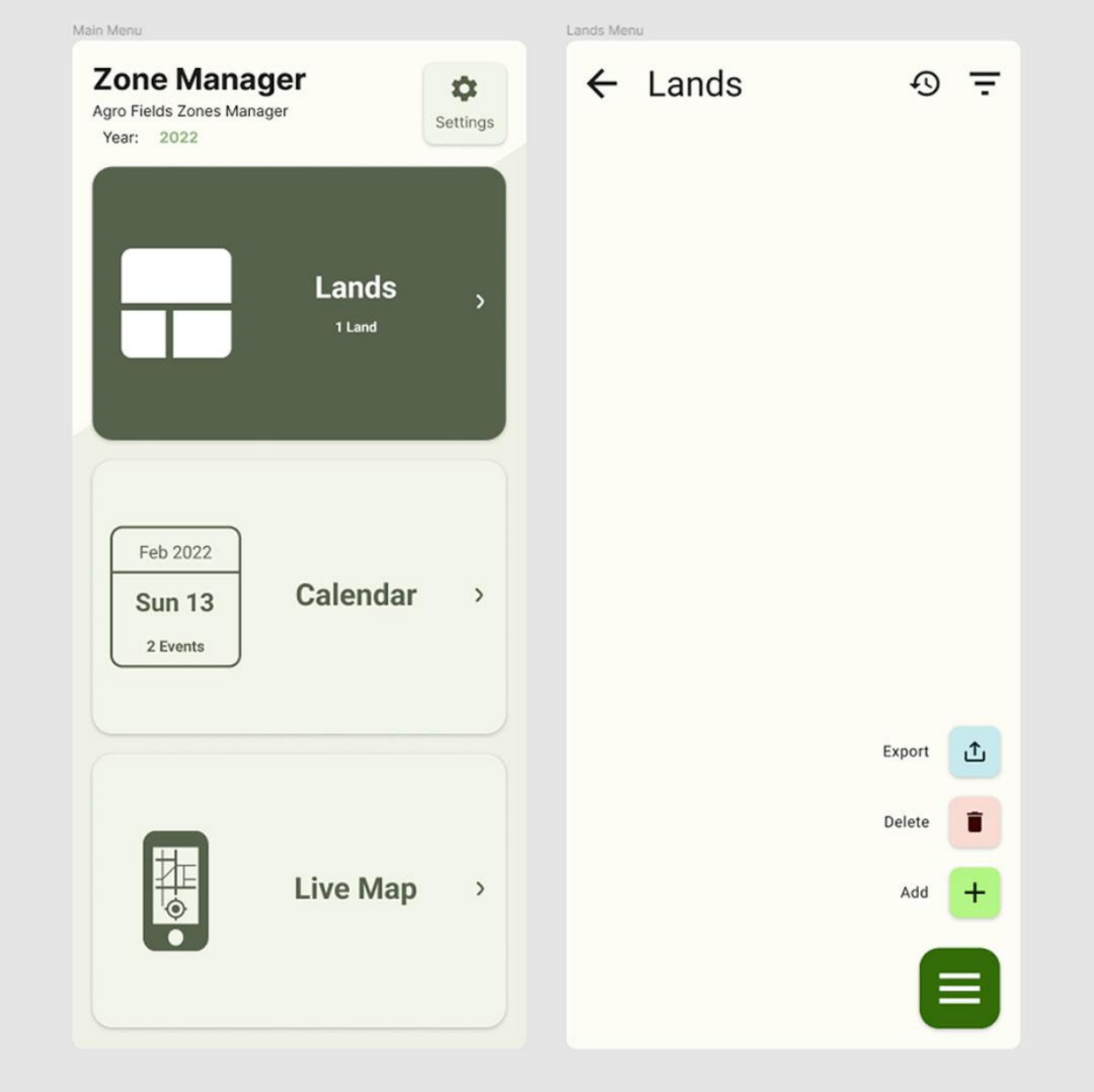
   ➢ **Surface colors**, χρησιμοποιείται σε αντικείμενα όπως cards, sheets ή μενού.

   - **Surface** είναι το χρώμα που χρησιμοποιούμε στο φόντο των αντικειμένων.
   - **On Surface** είναι το χρώμα που χρησιμοποιούμε στο περιεχόμενο που έχει ως φόντο το χρώμα Surface.
   - **Surface Variant** είναι το εναλλακτικό χρώμα που χρησιμοποιούμε στο φόντο.
   - **On Surface Variant** είναι το εναλλακτικό χρώμα που χρησιμοποιούμε στο περιεχόμενο που έχει ως φόντο το χρώμα Surface Variant.

   ★ Για περαιτέρω πληροφορίες σχετικά με τους χρωματισμούς και τις τοποθεσίες αυτών των χρωμάτων στα android components, δείτε το link στην ενότητα: 2.2.1. Material IO.

8. Μόλις θα έχουμε δημιουργήσει μια οθόνη Frame με όλα τα components του και είμαστε ευχαριστημένοι, θα επαναλαμβάνουμε το βήμα 5, βήμα 6 και βήμα 7 για να φτιάξουμε και τα άλλα Frames που αντιστοιχούν στις άλλες οθόνες της εφαρμογής μας.

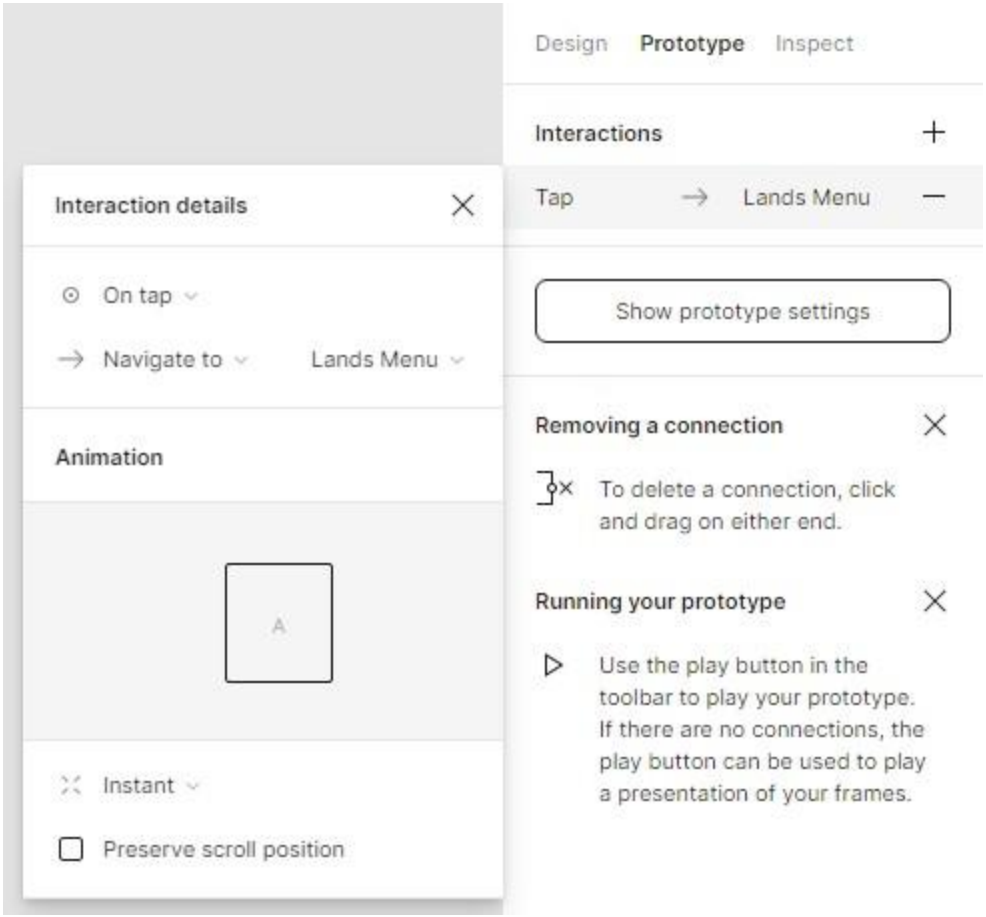
Ένα παράδειγμα δύο Frame “Light UI”:



9. Μόλις έχουμε τελειώσει με των σχεδιασμό των οθονών της εφαρμογής μας, θα ξεκινήσουμε να δοκιμάζουμε την λειτουργικότητα του UI. Πατώντας πάνω στα αντικείμενα που σχεδιάσαμε, μπορούμε να ορίσουμε αν αυτά θα έχουν κάποια αλληλεπίδραση με των χρήστη, αυτό το καταφέρνουμε ως εξής:

   - ➢ Επιλέγουμε το αντικείμενο που θέλουμε να είναι διαδραστικό είτε από το αριστερό μενού στην καρτέλα Layers είτε από των καμβά.
   - ➢ Στο δεξί μενού πάμε στην καρτέλα Prototype και στην ενότητα Interactions πατάμε το κουμπί προσθήκη (+).
   - ➢ Πατώντας πάνω στο καινούργιο αντικείμενο θα μας εμφανίσει ένα παραθυράκι με δύο drop-down box.
   - ➢ Στο πρώτο drop-down box επιλέγουμε τι τύπου αλληλεπίδρασης επιθυμούμε να έχουμε, π.χ. on tap είναι όταν ο χρήστης θα πατάει πάνω σε αυτό, on drag είναι όταν ο χρήστης σύρει το ποντίκι ή το δάχτυλό του πάνω στο αντικείμενο, και πάει λέγοντας.
   - ➢ Στο δεύτερο drop-down box επιλέγουμε τι ενέργεια θα κάνει η εφαρμογή μας με αυτή την αλληλεπίδραση, π.χ.: navigate to είναι για να πάμε σε άλλη οθόνη, change to είναι για να αλλάξει η οθόνη χωρίς να επηρεάσει το back stuck, open overlay είναι για την εμφάνιση κάποιου αναδυόμενου παραθύρου, και πάει λέγοντας.

Ένα παράδειγμα αλληλεπίδρασης:

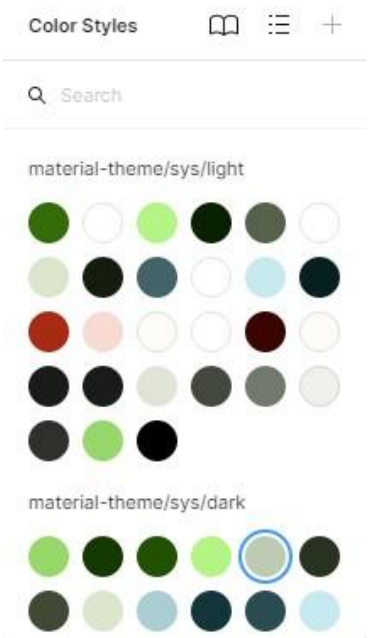


10. Μόλις τελειώσουμε να ορίζουμε τις αλληλεπιδράσεις του χρήστη, μπορούμε να τις δοκιμάσουμε πατώντας το κουμπί Present (με το εικονίδιο play) στο κύριο μενού. Μπαίνοντας στην λειτουργία “Prototype”, όπως αναφέραμε νωρίτερα στην ενότητα 2.1.2.2. Λειτουργία Prototype, εδώ μπορούμε να δοκιμάσουμε την εφαρμογή μας σαν να είμαστε ένας χρήστης, πατώντας στα σημεία που ορίσαμε στο βήμα 9 θα διαπιστώσουμε ότι τα σχέδια είναι διαδραστικά, έτσι λοιπόν μπορούμε να διακρίνουμε ποια είναι τα αδύναμα σημεία στο UI μας και να τα διορθώσουμε πριν ακόμη ξεκινήσουμε να τα γράψουμε κώδικα.

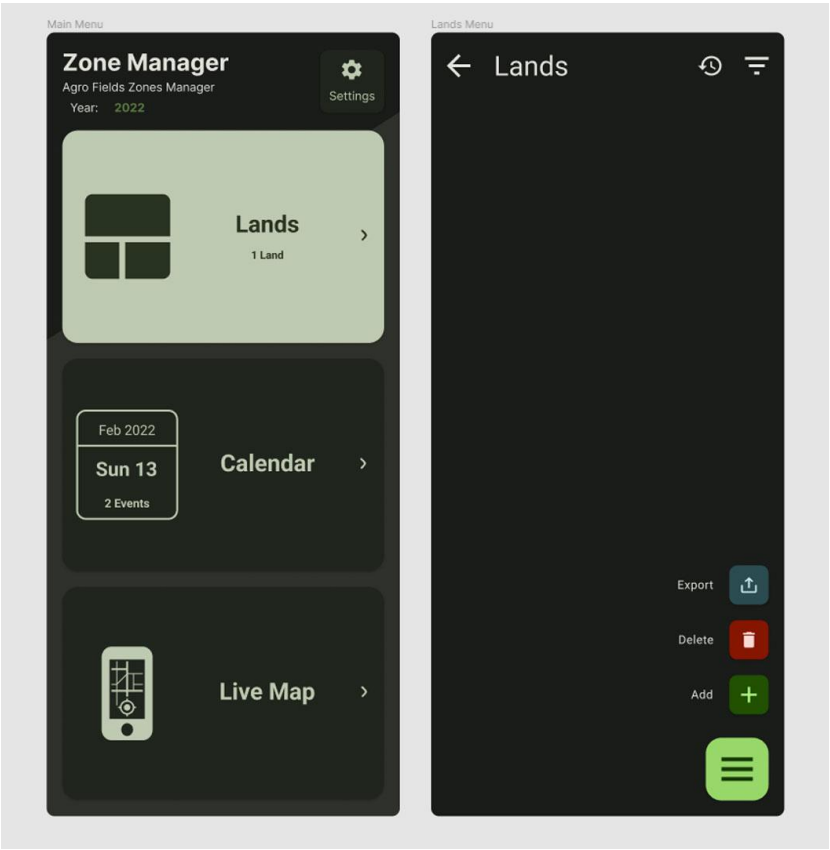
Ένα παράδειγμα της λειτουργίας “Prototype”:

Στην παραπάνω παρουσίαση η οθόνη άλλαξε πατώντας πάνω στο κουμπί “Lands”.

11. Σε περίπτωση που επιθυμήσουμε να προσθέσουμε σκοτεινό θέμα στην εφαρμογή μας, δημιουργούμε μια νέα σελίδα “Dark UI”, αντιγράφουμε το περιεχόμενο της σελίδα “Light UI” στην σελίδα “Dark UI” και αντικαθιστούμε τα χρώματα από “material-theme/sys/light” σε “material-theme/sys/dark”. Επαναλαμβάνουμε το βήμα 10 για να δούμε ότι δεν χρειαζόμαστε αλλαγές.

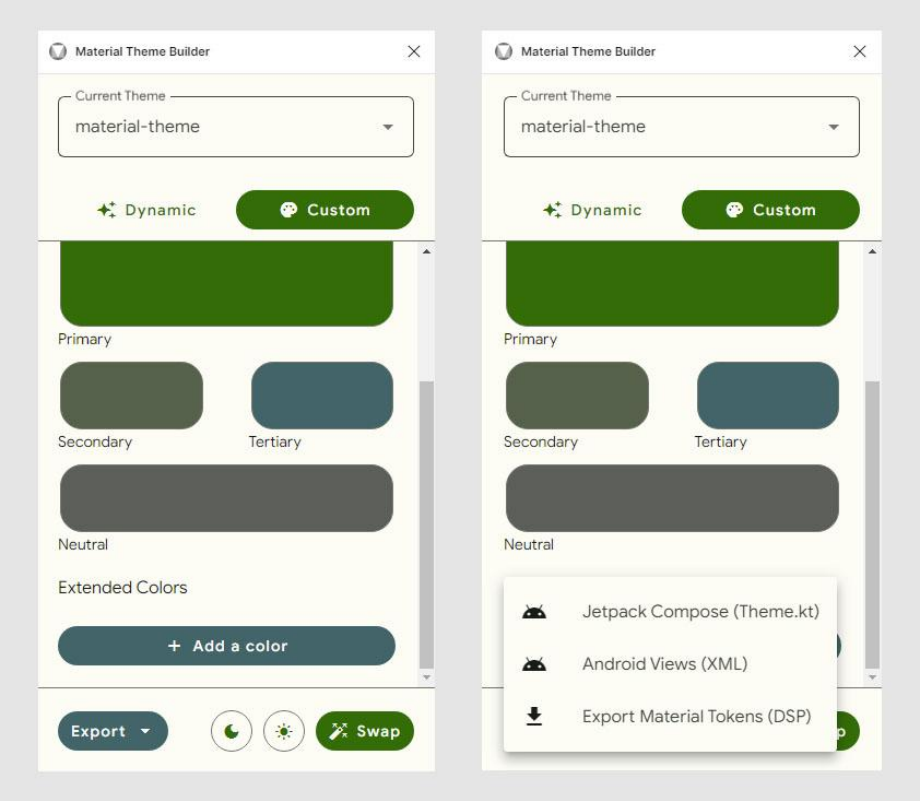


Ένα παράδειγμα δύο Frame “Dark UI”:



12. Όταν θα έχουμε τελειώσει με όλα τα Frames και είμαστε ευχαριστημένη με την διεπαφή μας, θα κάνουμε εξαγωγή το Material Theme και όλα τα εικονίδια που δημιουργήσαμε.

➢ Για εξαγωγή του Material Theme: Στο Current Theme επιλέγουμε το θέμα μας, πατάμε το κουμπί **export**, και τέλος για τύπο αρχείου επιλέγουμε το Android Views (XML).



➢ Για εξαγωγή των εικονιδίων: Πατάμε στα αντικείμενα που θέλουμε να εξάγουμε ως εικονίδιο, πάμε στην καρτέλα **Design** στην επιλογή **Export** πατάμε σε μορφή **SVG**.

4. Υλοποίηση της Εφαρμογής

Σε αυτή την ενότητα θα δούμε το τρίτο στάδιο σχεδίασης εφαρμογών, την **υλοποίηση**. Σε αυτό το στάδιο δημιουργούμε των κώδικα της εφαρμογής μας.

4.1. Λειτουργίες Εφαρμογής

Σε αυτή την ενότητα θα βασιστούμε στην υλοποίηση των βασικών λειτουργιών της εφαρμογής, όπως: την μελέτη των γεωγραφικών αρχείων, πράξεων γεωγραφικών συνόλων και την μετατροπή συντεταγμένων σε άλλα συστήματα συντεταγμένων.

4.1.1. Δομές Γεωγραφικών Αρχείων

Μια λειτουργιά της εφαρμογής μας είναι η εισαγωγή και εξαγωγή γεωγραφικών αρχείων, τα πιο δημοφιλές γεωγραφικών αρχεία σήμερα, είναι τα:

- KML (XML): .kml, .kmz
- GML (XML): .gml, .xml
- GEOJSON (JSON): .geojson, .json
- ShapeFile (Binary): .shp, .shx, .dbf

Σε αυτή την ενότητα θα δούμε την δομή αυτών των αρχείων.

4.1.1.1. Δομές KML Αρχείων

Το Keyhole Markup Language (KML) είναι αρχεία τύπου XML της Google για την έκφραση γεωγραφικών σχολίων και οπτικοποίησης σε δισδιάστατους χάρτες αλλά και τρισδιάστατα προγράμματα περιήγησης Earth. Μπορεί να γίνει εισαγωγή και εξαγωγή δεδομένων όπως σε ένα απλό αρχείο XML, δείτε την ενότητα: 4.2.4. Εισαγωγή και Εξαγωγή Αρχείων XML. Ας δούμε ένα παράδειγμα δομής αρχείου KML:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
	<Placemark>
		<name>Simple placemark</name>
		<Polygon> <!-- other shapes: Point, GroundOverlay, LineString -->
			<extrude>1</extrude>
			<altitudeMode>relativeToGround</altitudeMode>
			<outerBoundaryIs> <!-- borders -->
				<LinearRing>
<coordinates>lng1,lat1,alt1 lng2,lat2,alt2 lng3,lat3,alt3 ... lng1,lat1,alt1</coordinates>
				</LinearRing>
			</outerBoundaryIs>
			<innerBoundaryIs> <!-- holes -->
				<LinearRing>
<coordinates>lng1,lat1,alt1 lng2,lat2,alt2 lng3,lat3,alt3 ... lng1,lat1,alt1</coordinates>
				</LinearRing>
			</innerBoundaryIs>
		</Polygon>
	</Placemark>
</kml>
```

Για περισσότερες λεπτομέρειες σχετικά με τα KML αρχεία και την δομή τους, μπείτε στο link: https://developers.google.com/kml/documentation

4.1.1.2. Δομές GML Αρχείων

Το Geography Markup Language (GML) είναι αρχεία τύπου XML της Open Geospatial Consortium (OGC) που εκφράζουν γεωγραφικά χαρακτηριστικά. Μπορεί να γίνει εισαγωγή και εξαγωγή δεδομένων όπως σε ένα απλό αρχείο XML, δείτε την ενότητα: 4.2.4. Εισαγωγή και Εξαγωγή Αρχείων XML. Ας δούμε δύο παράδειγμα δομών αρχείων GML:

Δομή GML 2.0:

```
<wfs:FeatureCollection>
        <gml:boundedBy>
                <!-- <gml:Envelope srsName="..."> ... </gml:Envelope> -->
        </gml:boundedBy>
        <wfs:member> <!-- or <gml:featureMember> -->
                <myns:geomProperty>
                        <myns:name></myns:name>
                        <gml:Polygon>
                                <gml:outerBoundaryIs> <!-- borders -->
                                        <gml:LinearRing>
<gml:coordinates decimal="." cs="," ts=" ">lng1,lat1 lng2,lat2 lng3,lat3 ... lng1,lat1</gml:coordinates>
                                        </gml:LinearRing>
                                </gml:outerBoundaryIs>
                                <gml:innerBoundaryIs> <!-- holes -->
                                        <gml:LinearRing>
<gml:coordinates decimal="." cs="," ts=" ">lng1,lat1 lng2,lat2 lng3,lat3 ... lng1,lat1</gml:coordinates>
                                        </gml:LinearRing>
                                </gml:innerBoundaryIs>
                        </gml:Polygon>
                </myns:geomProperty>
        </wfs:member>
        <!-- <wfs:member> ... </wfs:member> -->
</wfs:FeatureCollection>
```

Δομή GML 3.1:

```
<wfs:FeatureCollection>
        <wfs:member> <!-- or <gml:featureMember> -->
                <myns:geomProperty>
                        <gml:Polygon srsDimension="2">
                                <gml:exterior> <!-- borders -->
                                        <gml:LinearRing srsDimension="2">
<gml:posList>lng1 lat1 lng2 lat2 lng3 lat3 lng4 lat4 ... lng1 lat1</gml:posList>
                                        </gml:LinearRing>
                                </gml:exterior>
                                <gml:interior> <!-- holes -->
                                        <gml:LinearRing srsDimension="2">
<gml:posList>lng1 lat1 lng2 lat2 lng3 lat3 lng4 lat4 ... lng1 lat1</gml:posList>
                                        </gml:LinearRing>
                                </gml:interior>
                        </gml:Polygon>
                </myns:geomProperty>
        </wfs:member>
        <!-- <wfs:member> ... </wfs:member> -->
</wfs:FeatureCollection>
```

Για περισσότερες λεπτομέρειες σχετικά με τα GML αρχεία και την δομή τους,
μπείτε στο: www.ogc.org/standards/gml

4.1.1.3. Δομές GeoJSON Αρχείων

Το GeoJSON είναι μία μορφή open standard που έχει σχεδιαστεί για την αναπαράσταση απλών γεωγραφικών χαρακτηριστικών, μαζί με τα μη χωρικά χαρακτηριστικά τους και βασίζεται στη μορφή JSON. Μπορεί να γίνει εισαγωγή και εξαγωγή δεδομένων όπως σε ένα απλό αρχείο JSON, δείτε την ενότητα: 4.2.5. Εισαγωγή και Εξαγωγή Αρχείων JSON. Ας δούμε ένα παράδειγμα δομής αρχείου GeoJSON:

```
{
	"type": "FeatureCollection",
	"features": [
		{
			"type": "Feature",
			"geometry": {
				"type": "Polygon",
				"coordinates": [
					//the first list is the border
					[
						[lng1,lat1],
						[lng2,lat2],
						[lng3,lat3],
						[lng4,lat4],
						//[...], other coordinates
						[lng1,lat1]
					]//other lists are holes
					//,[...]
				]
			}// others custom feature properties: ,"properties": { "name": "Dinagat Islands" }
		},{
			"type": "Feature",
			"geometry": {
				"type": "MultiPolygon",
				"coordinates":[ //list of Polygon coordinates
					[
						//the first list is the border
						[
							[lng1,lat1],
							[lng2,lat2],
							[lng3,lat3],
							[lng4,lat4],
							//[...], other coordinates
							[lng1,lat1]
						]//other lists are holes
						//,[...]
					]//other lists are other polygons
					//,[...]
				]
			}
		}//,{...} other features
	]
}
```

Για περισσότερες λεπτομέρειες σχετικά με τα GeoJSON αρχεία και την δομή τους, μπείτε στο link: https://geojson.org

4.1.1.4. Αναλυτής Αρχείων Shapefile

Τα αρχεία Shapefile είναι μια διανυσματική μορφή αποθήκευσης δεδομένων της ESRI για την αποθήκευση της τοποθεσίας, του σχήματος και των δεδομένων των γεωγραφικών συνόλων, η αποθήκευση αυτών των δεδομένων γίνεται σε στρώματα (Layers) μιας περιοχής.

Ως binary αρχείο κλειστού τύπο το καθιστά δύσκολο στην μελέτη μας χωρίς την βοήθεια κάποιας βιβλιοθήκης. Για των λόγω έλλειψης βιβλιοθήκης εξαγωγής αρχείων .shp σε Android, η λειτουργιά εξαγωγής γεωγραφικών δεδομένων δεν θα περιλαμβάνει αυτών των τύπο αρχείου.

Στην ενότητα 2.2.7. Java ESRI Shapefile Reader by olenus, είδαμε για μια βιβλιοθήκη ανάγνωσης δεδομένων από ShapeFile αρχεία που ονομάζετε “**Java ESRI Shapefile Reader**” τού olenus. Η ανάγνωση των Shapefile με την βοήθεια αυτής της βιβλιοθήκης γίνεται με των εξής τρόπο:

```
public static List<List<LatLng>> readShapeFile(InputStream is) throws IOException,
InvalidShapeFileException {
	List<List<LatLng>> result = new ArrayList<>();
	ShapeFileReader r = new ShapeFileReader(is);
	PolygonShape tempPolygon;
	PolygonMShape tempPolygonM;
	PolygonZShape tempPolygonZ;
	AbstractShape shape;
	while ((shape = r.next()) != null) {
		switch (shape.getShapeType()){
			case POLYGON:
				tempPolygon = (PolygonShape) shape;
				for (int i = 0; i < tempPolygon.getNumberOfParts(); i++) {
					result.add(readPoints(tempPolygon.getPointsOfPart(i)));
				}
				break;
			case POLYGON_M:
				tempPolygonM = (PolygonMShape) shape;
				for (int i = 0; i < tempPolygonM.getNumberOfParts(); i++) {
					result.add(readPoints(tempPolygonM.getPointsOfPart(i)));
				}
				break;
			case POLYGON_Z:
				tempPolygonZ = (PolygonZShape) shape;
				for (int i = 0; i < tempPolygonZ.getNumberOfParts(); i++) {
					result.add(readPoints(tempPolygonZ.getPointsOfPart(i)));
				}
				break;
			//Other ShapeType: Point, MultiPoint, MultiPatch, Polyline
		}
	}
	return result;
}
private static List<LatLng> readPoints(PointData[] points){
	List<LatLng> result = new ArrayList<>();
	for (PointData point : points) { result.add(new LatLng(point.getY(), point.getX())); }
	return result;
}
```

Προσοχή το point.getX() είναι το γεωγραφικό μήκος και το point.getY() είναι το γεωγραφικό πλάτος.

4.1.2. Μετατροπή Συστημάτων Συντεταγμένων

Ένα πρόβλημα που προκύπτει από τα γεωγραφικά αρχεία είναι ότι μπορούν να αποθηκεύουν συντεταγμένες σε άλλα γεωγραφικά συστήματα συντεταγμένων από το σύστημα συντεταγμένων που χρησιμοποιεί το google maps, που είναι το EPSG:4326 (WGS84). Για να δούμε τώρα των κώδικα πως να μετατρέψουμε τις γεωγραφικές συντεταγμένες από ένα σύστημα συντεταγμένων σε άλλο με την βοήθεια του Proj4J:

```
public class CoordinatesHelper {
	private BasicCoordinateTransform transform;
	public CoordinatesHelper(){
		transform = null;
	}
	private boolean checkIfValid(String crs){
		try{
			CoordinateReferenceSystem test = new CRSFactory()
				.createFromName(crs);
			return test != null;
		}catch (Exception e){
			return false;
		}
	}
	public boolean isTransformSet(){
		return transform != null;
	}
	public void setCrs(String crs){
		if(checkIfValid(crs)){
			try{
				CoordinateReferenceSystem srcCrs = new CRSFactory()
					.createFromName(crs);
				CoordinateReferenceSystem dstCrs = new CRSFactory()
					.createFromName("EPSG:4326");
				transform = new BasicCoordinateTransform(srcCrs, dstCrs);
			}catch (Exception e){
				transform = null;
			}
		}else{
			transform = null;
		}
	}
	public LatLng convertEPSG(LatLng point) {
		if(transform == null) return null;
		ProjCoordinate srcCoord = new ProjCoordinate(point.longitude, point.latitude);
		ProjCoordinate dstCoord = new ProjCoordinate();
		transform.transform(srcCoord, dstCoord);
		return new LatLng(dstCoord.y,dstCoord.x);
	}
}
```

Το String crs υποδηλώνει των κωδικό του συστήματος συντεταγμένων που θέλουμε να μετατρέψουμε σε EPSG:4326 (WGS84) και το LatLng point υποδηλώνει το σημείο που θέλουμε να μετατρέψουμε.

4.1.3. Πράξεις Γεωγραφικών Συνόρων

Η εφαρμογή μας πρέπει να υποστηρίζει τουλάχιστον τις βασικές πράξεις γεωγραφικών συνόρων η οποίες είναι η ένωση, η τομή και η διαφορά. Αυτή η λειτουργιά θα δημιουργηθεί με την βοήθεια της βιβλιοθήκης **ArcGIS Runtime**, ο κώδικας είναι ο εξής:

Ένωση γεωγραφικών συνόρων:

```
public static List<LatLng> union(List<LatLng> p1, List<LatLng> p2) {
        Polygon polygon1 = convert(p1); Polygon polygon2 = convert(p2);
        if(polygon1 == null || polygon2 == null) {
                return new ArrayList<>();
        }
        try{
                return extractList(GeometryEngine.union(polygon1,polygon2));
        }catch (Exception e){
                e.printStackTrace();
        }
        return new ArrayList<>();
}
```

Τομή γεωγραφικών συνόρων:

```
public static List<LatLng> intersection(List<LatLng> p1, List<LatLng> p2) {
        Polygon polygon1 = convert(p1); Polygon polygon2 = convert(p2);
        if(polygon1 == null || polygon2 == null) {
                return new ArrayList<>();
        }
        try{
                return extractList(GeometryEngine.intersection(polygon1,polygon2));
        }catch (Exception e){
                e.printStackTrace();
        }
        return new ArrayList<>();
}
```

Διαφορά γεωγραφικών συνόρων:

```
public static List<List<LatLng>> differences(List<LatLng> p1, List<LatLng> p2) {
        Polygon polygon1 = convert(p1); Polygon polygon2 = convert(p2);
        if(polygon1 == null || polygon2 == null) {
                return new ArrayList<>();
        }
        try{
                return extractLists(GeometryEngine.difference(polygon1,polygon2));
        }catch (Exception e){
                e.printStackTrace();
        }
        return new ArrayList<>();
}
```

Βοηθητικές μεθόδους πράξεων γεωγραφικών συνόρων:

```
public static Polygon convert(List<LatLng> points) {
	if(points == null) {
		return null;
	}
	PointCollection polygonPoints = new PointCollection(SpatialReferences.getWgs84());
	for(LatLng point : points){
		polygonPoints.add(point.latitude,point.longitude);
	}
	return new Polygon(polygonPoints);
}

public static List<LatLng> extractList(Geometry geometry) {
	List<LatLng> result = new ArrayList<>();
	JSONObject jsonObj = new JSONObject(geometry.toJson());
	if(!jsonObj.has("rings")) return result;
	JSONArray points = jsonObj.getJSONArray("rings").getJSONArray(0);
	for (int i = 0; i < points.length() - 1; i++) {
		result.add(new LatLng(
			points.getJSONArray(i).getDouble(0),
			points.getJSONArray(i).getDouble(1)
		));
	}
	return result;
}

public static List<List<LatLng>> extractLists(Geometry geometry) {
	List<List<LatLng>> result = new ArrayList<>();
	JSONObject jsonObj = new JSONObject(geometry.toJson());
	if(!jsonObj.has("rings")) return result;
	JSONArray points;
	List<LatLng> item;
	for(int itemIndex = 0; itemIndex < jsonObj.getJSONArray("rings").length(); itemIndex++){
		points = jsonObj.getJSONArray("rings").getJSONArray(itemIndex);
		item = new ArrayList<>();
		for(int pointIndex  = 0; pointIndex < points.length()-1; pointIndex++){
			item.add(new LatLng(
				points.getJSONArray(pointIndex).getDouble(0),
				points.getJSONArray(pointIndex).getDouble(1)
			));
		}
		result.add(item);
	}
	return result;
}
```

Η πράξεις όπως είδαμε γίνονται εύκολα με τις στατικές μεθόδους της κλάσης **GeometryEngine**, το πρόβλημα είναι όμως στην εξαγωγή των σημείων από το αντικείμενο **Geometry** διότι το αντικείμενο είναι ένα Layer για χάρτη και δεν παρέχει μέθοδο εξαγωγής σημείων. Περνώντας το αντικείμενο Geometry σε ένα αντικείμενο JSONObject με την μέθοδο toJson() θα μπορέσουμε να διαβάσουμε τα πεδία rings τα οποία παρέχουν τα σημεία που ψάχναμε.

4.2. Αρχιτεκτονική Back End Εφαρμογής

Σε αυτή την ενότητα θα δούμε πως υλοποιήθηκε η αποθήκευση και προσκόμισης δεδομένων τής εφαρμογής.

4.2.1. Room Database

Η εφαρμογή μας χρησιμοποιεί για αποθήκευση δεδομένων την τοπική βάση δεδομένων Room. Το Room, όπως αναφέρθηκε νωρίτερα στην ενότητα: 2.2.2. Room Database, είναι μια βάση δεδομένων ORM, δηλαδή οι πίνακες δημιουργούνται αυτόματα από την δομή των κλάσεων.

Ας δούμε ένα πολύ απλό παράδειγμα βάσης δεδομένων Room για αποθήκευση και ανάγνωση κατηγοριών συμβάντων:

- Για αρχή θα φτιάξουμε την κλάση CalendarCategory:

```
@Entity(tableName = "CalendarCategory")
public class CalendarCategory {
        @PrimaryKey(autoGenerate = true) @ColumnInfo(name = "ID") private long id;
        @ColumnInfo(name = "NAME") private String name;
        @ColumnInfo(name = "COLOR") private ColorData colorData;

        public CalendarCategory(long id, String name, ColorData colorData) {
                this.id = id;
                this.name = name;
                this.colorData = colorData;
        }

        public long getId() { return id; }
        public String getName() { return name; }
        public ColorData getColorData() { return colorData; }
        public void setId(long id) { this.id = id; }
        public void setName(String name) { this.name = name; }
        public void setColorData(ColorData colorData) { this.colorData = colorData; }
}
```

- Ας δούμε τώρα την διεπαφή Dao αυτού του αντικειμένου:

```
@Dao
public interface CalendarCategoriesDao {
        @Query("SELECT * FROM CalendarCategory")
        List<CalendarCategory> getCalendarCategories();

        @Query("SELECT * FROM CalendarCategory WHERE ID = :id LIMIT 1")
        CalendarCategory getCalendarCategory(long id);

        @Insert(onConflict = OnConflictStrategy.REPLACE)
        long insert(CalendarCategory category);

        @Delete
        void delete(CalendarCategory category);
}
```

- Αν η κλάση μας περιέχει συνθέτες ιδιότητες θα δημιουργήσουμε ένα TypeConverter:

```
public class MyTypesConverter {
	@TypeConverter
	public static String colorToString(ColorData color) {
		return color.toString();
	}

	@TypeConverter
	public static ColorData colorFromString(String color) {
		return new ColorData(color);
	}
}
```

- Τέλος θα δημιουργήσουμε την κλάση της βάσης δεδομένων:

```
@Database(
	version = 1,
	entities = {
		CalendarCategory.class,
		//other entities
	}
)
@TypeConverters({
	MyTypesConverter.class
})
public abstract class MyRoomDatabase extends RoomDatabase {
	public abstract CalendarCategoriesDao calendarCategoriesDao();
	//other dao
}
```

- Για την δημιουργία αντικειμένου της βάσης δεδομένων εκτελούμε των κώδικα:

```
public static MyRoomDatabase getLocalDbFromApplication(Application application){
	return Room.databaseBuilder(
			application,
			MyRoomDatabase.class,
			“database-name-string”		//database name string
		).build();
}
```

Για περισσότερες λεπτομέρειες σχετικά το Room δείτε το Link στην ενότητα: 2.2.2. Room Database

4.2.2. Διεπαφή Repository

Η διεπαφή **Repository** είναι μια διεπαφή που περιέχει όλες τις μεθόδους ανάγνωσης και εγγραφής δεδομένων από κάποιο back-end. Σκοπός αυτής της διεπαφής είναι η δυνατότητα δημιουργίας πολλαπλών implementation των οποίων ο τρόπος ανάγνωσης και εγγραφής δεδομένων διαφοροποιούνται μεταξύ τους, π.χ.: αλλιώς θα είναι το repository ενός API και αλλιώς ενός repository τοπικής βάσης δεδομένων αλλά οι μέθοδοι τους θα είναι ίδιες. Στην εφαρμογή χρησιμοποιούμε μια διεπαφή repository διότι όλα τα δεδομένα είναι συσχετισμένα μεταξύ τους, αλλά αυτό δεν σημαίνει ότι μπορούμε να έχουμε μόνο μια διεπαφή Repository.

Ας δούμε σε κώδικα ένα απόσπασμα του repository μαζί με το implementation του, στο παράδειγμα θα δούμε τις μεθόδους που σχετίζονται με τις κατηγορίες συμβάντων:

```
public interface AppRepository {
        // code..
        List<CalendarCategoryModel> getCalendarCategories();
        CalendarCategoryModel getCalendarCategory(lond id);
        void saveCalendarCategory(CalendarCategoryModel category);
        void deleteCalendarCategory(CalendarCategoryModel category);
}

public class AppRepositoryLocalImpl implements AppRepository {
        private final MyRoomDatabase db;
        public AppRepositoryLocalImpl(MyRoomDatabase db){
                this.db = db;
        }
        // code..
        @Override
        List<CalendarCategoryModel> getCalendarCategories(){
                List<CalendarCategory> categories =
                        db.calendarCategoriesDao().getCalendarCategories();
                return CalendarCategoryMapper.toModelList(categories);
        }
        @Override
        CalendarCategoryModel getCalendarCategory(lond id){
                CalendarCategory category =
                        db.calendarCategoriesDao().getCalendarCategory(id);
                return CalendarCategoryMapper.toModel(category);
        }
        @Override
        void saveCalendarCategory(CalendarCategoryModel category){
                CalendarCategory categoryEntity = CalendarCategoryMapper.toEntity(category);
                db.calendarCategoriesDao().insert(categoryEntity);
        }
        @Override
        void deleteCalendarCategory(CalendarCategoryModel category){
                CalendarCategory categoryEntity = CalendarCategoryMapper.toEntity(category);
                db.calendarCategoriesDao().delete(categoryEntity);
        }
}
```

4.2.3. MVVM Αρχιτεκτονική

Η αρχιτεκτονική Model-View-ViewModel (MVVM) είναι μια αρχιτεκτονική που διαχωρίζει την γραφική διεπαφή της εφαρμογής μας απο το back end.

- Το model είναι τα αντικείμενα που θα πρέπει να παρέχει η εφαρμογή μας για να λειτουργήσει, π.χ.: το αντικείμενο χωράφι, το αντικείμενο συμβάν, το αντικείμενο ζώνη χωραφιού, κ.α.
- Το κομμάτι του view περιγραφή το layout του αντικειμένου, δείτε για τα ViewHolder στην ενότητα: 4.3.8. ViewHolders Αντικειμένων.
- Το κομμάτι του ViewModel περιγραφή το παροχέα δεδομένων στην εφαρμογή μας χωρίς να ασχοληθεί με την γραφική διεπαφή της εφαρμογής. Μέσα σε αυτά συνηθίζεται να έχουμε αντικείμενα LiveData τα οποία μπορούμε να παρακολουθήσουμε τις αλλαγές τους από των κώδικα της οθόνης μας.

Η αρχιτεκτονική MVVM είναι μια πολύ καλή αρχιτεκτονική για των διαχωρισμό πόρων από το κύριο thread της εφαρμογής, για να δούμε ένα παράδειγμα χρήσης αρχιτεκτονικής MVVM.

- Δημιουργούμε τα Viewmodel μας πού μπορούν να έχουν ως ιδιότητες μερικά repository και μερικά LiveData που περιγράφουν μερικά αντικείμενα, μαζί με συνάρτησης πού σχετίζονται με τις ιδιότητες του, για παράδειγμα: ένα Viewmodel χωραφιών που περιέχει ένα LiveData λίστας χωραφιών και το repository τής τοπικής βάσης δεδομένων.
- Ορίζουμε στα αρχεία κώδικα των οθονών να διαβάσει και να παρακολουθεί τις αλλαγές των μεταβλητών LiveData που τους ενδιαφέρει με την μέθοδο: observe(LifecycleOwner,Observer<Object>), όπου το LifecycleOwner είναι ο κύκλος ζωής της οθόνης πού το παρακολουθεί το LiveData και το Observer<Object> είναι μια μέθοδος πού θα τρέχει όταν γίνει αλλαγή τιμής στο LiveData, προσοχή αυτή η μέθοδος πρέπει να περιέχει ως παράμετρο εισόδου των τύπο τής μεταβλητής τού LiveData.
- Καλούμε συνάρτησης από την οθόνη στο Viewmodel, πχ αρχικοποίηση λιστών, κατά σειρά το Viewmodel θα τρέξει σε παράλληλο Thread μια αργή ή βαριά συνάρτηση, π.χ.: μια μέθοδος του Repository. Όταν ολοκληρωθεί η εκτέλεση αυτής της συνάρτησης το αποτέλεσμα θα περαστεί σαν τιμή σε μία μεταβλητή LiveData.
- Όταν η τιμή τής μεταβλητής LiveData αλλάξει, ο κώδικας της οθόνης θα εκτελέσει την μέθοδο που δεσμεύτηκε στο observe(LifecycleOwner,Observer<Object>) ως Observer<Object>.

Στην εφαρμογή μας χρησιμοποιούμε ένα Viewmodel για των λόγο συσχέτισης δεδομένων, αυτό όμως δεν σημαίνει ότι μπορούμε να έχουμε μόνο ένα Viewmodel στην εφαρμογή μας, μπορούμε να φτιάξουμε όσα Viewmodel επιθυμούμε και το καθένα να είναι υπεύθυνο για τις δικές του λειτουργίες.

Τα αντικείμενα Viewmodel δεσμεύονται στα Activity κάνοντας έτσι εύκολο την μεταφοράς δεδομένων μεταξύ Fragments, αλλά δύσκολη την μεταφοράς δεδομένων μεταξύ Activity, για αυτό των λόγο επιλέχθηκε η αρχιτεκτονική One Activity Many Fragments στο front-end, δηλαδή ένα κύριο παράθυρο με πολλά υπό παράθυρα.

Για περισσότερες λεπτομέρειες σχετικά τα ViewModel δείτε το link:
https://developer.android.com/topic/libraries/architecture/viewmodel

Ας δούμε ένα παράδειγμα σχετικά με τα Viewmodel, στο παράδειγμα θα δούμε μια υλοποίηση σε κώδικα του κομματιού που σχετίζεται με τις κατηγορίες συμβάντων:

```
public class AppViewModel extends AndroidViewModel {
        private final AppRepository appRepository;
        public AppViewModel(@NonNull Application application) {
                super(application);
                appRepository = new AppRepositoryLocalImpl(
                        DatabaseHelper.getLocalDbFromApplication(application)
                );
        }

        //code...

        private final long defaultCalendarCategoryID = -1;
        private final ColorData defaultCalendarCategoryColor= new ColorData("#788DCD");
        public CalendarCategoryModel getDefaultCategory(){
                return new CalendarCategoryModel(
                        defaultCalendarCategoryID,
                        getApplication().getResources().getString(R.string.calendar_default_category),
                        defaultCalendarCategoryColor
                );
        }

        private final MutableLiveData<List<CalendarCategoryModel>> calendarCategories =
                new MutableLiveData<>();
        public LiveData<List<CalendarCategoryModel>> getCalendarCategories(){
                return calendarCategories;
        }
        public void populateCalendarCategories() {
                AsyncTask.execute(() -> {
                        List<CalendarCategoryModel> categories =
                                appRepository.getCalendarCategories();
                        if(categories == null) categories = new ArrayList<>();
                        categories.add(0,getDefaultCategory());
                        calendarCategories.postValue(categories);
                });
        }
        public void saveCalendarCategory(CalendarCategoryModel category){
                if(category == null || category.getId() == getDefaultCategory().getId()) return;
                AsyncTask.execute(() -> {
                        appRepository.saveCalendarCategory(category);
                        populateCalendarCategories();
                });
        }
        public void removeCalendarCategory(CalendarCategoryModel category){
                if(category == null || category.getId() == defaultCalendarCategoryID) return;
                AsyncTask.execute(() -> {
                        if(appRepository.getCalendarCategory(category.getId()) != null){
                                appRepository.deleteCalendarCategory(category);
                                populateCalendarCategories();
                        }
                });
        }
}
```

Ο κώδικας για να μπορούμε να κάνουμε observe το **MutableLiveData calendarCategories** μέσα σε ένα Activity:

```
private void initViewModel(){
        AppViewModel viewModel = new ViewModelProvider(this)
                .get(AppViewModel.class);
        viewModel.getCalendarCategories().observe(
                this,
                this::updateUiCategories
        );
}
```

Ο κώδικας για να μπορούμε να κάνουμε observe το **MutableLiveData calendarCategories** μέσα σε ένα Fragment:

```
private void initViewModel(){
        AppViewModel viewModel = new ViewModelProvider(requireActivity())
                .get(AppViewModel.class);
        viewModel.getCalendarCategories().observe(
                getViewLifecycleOwner(),
                this::updateUiCategories
        );
}
```

Τέλος, όταν καλέσουμε την μέθοδο **observe** για πρώτη φορά στην οθόνη ή γίνει ενημέρωση της μεταβλητής **MutableLiveData calendarCategories** από το ViewModel όσο το κάνει observe μια οθόνη, η μέθοδος updateUiCategories(List<CalendarCategoryModel>) θα καλείτε αυτόματα:

```
private void updateUiCategories(
        List<CalendarCategoryModel> calendarCategories
){
        // update UI
}
```

4.2.4. Εισαγωγή και Εξαγωγή Αρχείων XML

Για να δούμε πως μπορούμε να χρησιμοποιήσουμε ένα αντικείμενο **Document** για ανάλυση δεδομένων από ένα αρχείο XML:

```
public static List<CalendarCategoryModel> importXml(
        InputStream is
) throws ParserConfigurationException, IOException, SAXException {
        List<CalendarCategoryModel> result = new ArrayList<>();
        DocumentBuilder docBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
        Document document = docBuilder.parse(is);
        readCategoriesXml(document, result);
}
public static void readCategoriesXml(
        Document document,
        List<CalendarCategoryModel> result
){
        long id;
        String name;
        ColorData colorData;

        NodeList categoryTags = document.getElementsByTagName("category");
        Element category;
        NodeList categoryIdTags, categoryNameTags, categoryColorTags;
        for(int i = 0; i < categoryTags.getLength(); i++){
                if (categoryTags.item(i).getNodeType() != Node.ELEMENT_NODE) continue;
                category = (Element) categoryTags.item(i);

                categoryIdTags = categoryTags.getElementsByTagName("id");
                if(categoryIdTags.getLength() != 1) continue;
                id = Double.valueOf(
                        categoryIdTags.item(0).getNodeValue()
                ).longValue();

                categoryNameTags = categoryTags.getElementsByTagName("name");
                if(categoryNameTags.getLength() != 1) continue;
                name = categoryNameTags.item(0).getNodeValue();

                categoryColorTags = categoryTags.getElementsByTagName("color");
                if(categoryColorTags.getLength() != 1) continue;
                color = new ColorData(
                        categoryColorTags.item(0).getNodeValue()
                );

                result.add(new CalendarCategoryModel(id, name, colorData));
        }
}
```

Σε περίπτωση που το XML αρχείο είναι περίπλοκο τότε χρησιμοποιούμε **XmlPullParser**.

Δείτε το παράδειγμα χρήσης XmlPullParser στο link:

https://developer.android.com/reference/org/xmlpull/v1/XmlPullParser

Τώρα για να δούμε πως μπορούμε να δημιουργήσουμε ένα αντικείμενο τύπου **Document** και να το κάνουμε εξαγωγή δεδομένων σε ένα String με μορφή XML:

```
public static String exportXml(List<CalendarCategoryModel> dataList){
	Document doc = new Document();

	writeCategoriesXml(doc, dataList);

	XMLOutputProcessor xmlOutput = new AbstractXMLOutputProcessor() {
		@Override protected void printDeclaration(
			final Writer out,
			final FormatStack fStack
		) throws IOException {
			write(out, "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>");
			write(out, fStack.getLineSeparator());
		}
	};
	XMLOutputter xmOut = new XMLOutputter(Format.getPrettyFormat(), xmlOutput);

	return xmOut.outputString(doc);
}

private static String writeCategoriesXml(
	Document doc,
	List<CalendarCategoryModel> dataList
){

	Element root = new Element(“categories”);

	Element elementRoot, elementId, elementName, elementColor;
	for(CalendarCategoryModel data : dataList){
		elementRoot = new Element(“category”);

		elementId = new Element(“id”);
		elementId.addContent(data.getId());
		elementRoot.addContent(elementId);

		elementName = new Element(“name”);
		elementName.addContent(data.getName());
		elementRoot.addContent(elementName);

		elementColor = new Element(“color”);
		elementColor.addContent(data.getColorData().toString());
		elementRoot.addContent(elementColor);

		root.addContent(elementRoot);
	}

	doc.addContent(root);
}
```

4.2.5. Εισαγωγή και Εξαγωγή Αρχείων JSON

Για να δούμε πως μπορούμε να χρησιμοποιήσουμε ένα αντικείμενο **JSONObject** για ανάλυση δεδομένων από ένα αρχείο JSON:

```
public static List<CalendarCategoryModel> readCategoriesJson(
        String jsonStr
) throws JSONException{
        List<CalendarCategoryModel> result = new ArrayList<>();
        JSONObject json = new JSONObject(jsonStr);

        JSONArray categories = json.getJSONArray("categories");
        for (int i = 0; i < categories.length(); i++) {
                JSONObject category = categories.getJSONObject(i);

                result.add(new CalendarCategoryModel(
                        category.getInt("id"),
                        category.getString("name"),
                        new ColorData(category.getString("color")),
                ));
        }

        return result;
}
```

Τώρα για να δούμε πως μπορούμε να δημιουργήσουμε ένα αντικείμενο τύπου **JSONObject** και να το κάνουμε εξαγωγή δεδομένων σε ένα String με μορφή JSON:

```
public static String writeCategoriesJson(
        List<CalendarCategoryModel> dataList
) {
        JSONArray categories = new JSONArray();

        JSONObject category;
        for (CalendarCategoryModel data: dataList) {
                category = new JSONObject();

                category.put("id",data.getId());
                category.put("name",data.getName());
                category.put("color",data.getColorData().toString());

                categories.put(category);
        }

        JSONObject output = new JSONObject();
        output.put("categories", categories);

        return output.toString();
}
```

4.2.6. Εισαγωγή και Εξαγωγή Αρχείων Open XML Sheet

Για να δούμε πως μπορούμε να χρησιμοποιήσουμε ένα αντικείμενο **OPCPackage** της βιβλιοθήκης Apache POI για ανάλυση δεδομένων από ένα αρχείο Open XML Sheet:

```
public static void importXLSX(InputStream is, List<CalendarCategoryModel> result)
throws IOException, SAXException, OpenXML4JException, ParserConfigurationException {
	OPCPackage container = OPCPackage.open(is);
	OpenXmlState state = new OpenXmlState();
	ReadOnlySharedStringsTable strings = new ReadOnlySharedStringsTable(container);
	XSSFReader xssfReader = new XSSFReader(container);
	XSSFReader.SheetIterator iter = (XSSFReader.SheetIterator) xssfReader.getSheetsData();
	InputStream stream;
	String sheetName;
	while (iter.hasNext()) {
		stream = iter.next();
		sheetName = iter.getSheetName();
		if(sheetName.equal(“Calendar Categories”)){
			readCalendarCategoryFromXLSX(strings, stream, result);
		}
	}
}
private static void readCalendarCategoryFromXLSX(
	ReadOnlySharedStringsTable strings,
	InputStream sheetInputStream,
	List<CalendarCategoryModel> result
) throws IOException, SAXException, ParserConfigurationException {
	InputSource sheetSource = new InputSource(sheetInputStream);
	SAXParserFactory saxFactory = SAXParserFactory.newInstance();
	SAXParser saxParser = saxFactory.newSAXParser();
	XMLReader sheetParser = saxParser.getXMLReader();
	ContentHandler handler = new XSSFSheetXMLHandler(new StylesTable(), strings,
		new XSSFSheetXMLHandler.SheetContentsHandler() {
			private long id;
			private String name;
			private ColorData colorData;
			@Override public void startRow(int rowNum) {
				id = 0; name = “”; color = new ColorData(“#808080”);
			}
			@Override public void endRow(int rowNum) {
				result.add(new CalendarCategoryModel(id, name, colorData));
			}
			@Override public void cell(String key, String value, XSSFComment comment){
				String cellLetter = key.replaceAll("[0-9]","").toUpperCase().trim();
				switch (cellLetter){
					case "A": id = Double.valueOf(value).longValue(); break;
					case "B": name = value; break;
					case "C": colorData = new ColorData(value); break;
				}
			}
		}, false );
	sheetParser.setContentHandler(handler);
	sheetParser.parse(sheetSource);
}
```

Τώρα για να δούμε πως μπορούμε να δημιουργήσουμε ένα αντικείμενο τύπου **XSSFWorkbook** της βιβλιοθήκης Apache POI και να το κάνουμε μετατροπή δεδομένων σε ένα αρχείο Open XML Sheet:

```
public static void exportXLSX(
        FileOutputStream outputStream,
        List<CalendarCategoryModel> dataList
) throws IOException {

        XSSFWorkbook workbook = new XSSFWorkbook();

        writeCalendarCategoryToXLSX(workbook, dataList);

        workbook.write(outputStream);
        workbook.close();

}
private static void writeCalendarCategoryToXLSX(
        XSSFWorkbook workbook,
        List<CalendarCategoryModel> dataList
){
        XSSFSheet sheetLand = workbook.createSheet("Calendar Categories");

        Row row;
        Cell cell;
        for(int i = 0; i < dataList.size(); i++){
                row = sheetLand.createRow(i);

                cell = row.createCell(0);
                cell.setCellValue(data.getId());

                cell = row.createCell(1);
                cell.setCellValue(data.getName());

                cell = row.createCell(2);
                cell.setCellValue(data.getColorData().toString());
        }

}
```

4.3. Δημιουργία Οθονών

Έχοντας τα σχέδια των οθονών τις εφαρμογής μας, τα αρχεία Material Theme και τα εικονίδια μας σε αρχεία SVG, ήρθε η ώρα να τα σχεδιάσουμε στο Android Studio. Σε αυτήν την εργασία θα χρησιμοποιήσουμε Android Views για των λόγω ευκολίας δημιουργίας Layout αρχείων που να μοιάζουν με τα σχέδια των οθονών του Figma.

4.3.1. Αρχεία Drawable, Theme, Color, Strings Εφαρμογής

Για αρχή ξεκινάμε με το εύκολο κομμάτι, θα δημιουργήσουμε όλα τα επαναλαμβανόμενα res αρχεία πού θα χρειαστούμε στην εφαρμογή, όπως drawable εικονίδια από .svg, χρώματα και θέματα, αλφαριθμητικά όπως: τίτλους οθονών, περιγραφές αναδυόμενων παραθύρων, κτλ.

- Θέματα και χρώματα:
  - Κάνουμε copy paste τα αρχεία πού κατεβάζουμε του Material Theme που δημιουργίσαμε στη ενοτητα 3.2. Σχεδίαση Γραφικών Πόρων τής Εφαρμογής στο φάκελο res.
  - Κάνουμε εισαγωγή τα dependencies στο Gradle module.app του material theme: implementation 'com.google.android.material:material:<version>'
  - Ελέγχουμε για τυχόν προβλήματα στο θέμα, π.χ.: μπορεί να χρειαστεί η διαγραφή μερικών έξτρα πεδίων μέσα στα αρχεία themes.xml
  - Στο αρχείο manifest ορίζουμε το νέο θέμα ως προεπιλεγμένο αλλάζοντας το πεδίο android:theme του στοιχείου application και τελειώσαμε με την εισαγωγή των χρωμάτων και θεμάτων Material
- Αλφαριθμητικά:
  - Τα αλφαριθμητικά τίτλων, περιγραφών, κτλ. θα πάνε στο αρχείο strings.xml με μορφή: **<string name="string_name">string_value</string>**
  - Αν επιθυμούμε να δημιουργήσουμε κάποια μεταφρασμένη έκδοση της εφαρμογής μας:
    - Κάνουμε δεξί κλικ στο αρχείο strings.xml και επιλέγουμε το Open Translations Editor.
    - Πατώντας το κουμπί προσθήκη (ο πράσινος σταυρός) μπορούμε να προσθέσουμε ποιες γλώσσες επιθυμούμε.
    - Αυτομάτως το Android studio θα μας φτιάξει και άλλα αρχεία strings.xml ανάλογα τις επιλεγμένες γλώσσες.
    - Προσθέτουμε τα μεταφρασμένα string στα νέα αρχεία.
- Drawable εικονίδια:
  - Πατάμε δεξί κλικ στον φάκελο res και επιλέγουμε New > Vector Asset.
  - Επιλέγουμε Asset Type σε Local File (SVG, PSD).
  - Επιλέγουμε ένα όνομα εικόνας, για παράδειγμα ic_save ή ic_menu_share.
  - Επιλέγουμε την τοποθεσία του αρχείου svg που θέλουμε να προσθέσουμε.
  - Επιλέγουμε διαστάσεις εικόνας σε Device-independent pixel (dp), η πιο συνηθισμένη του τιμή είναι 48dp x 48dp.

Έχοντας όλα τα εικονίδια, αλφαριθμητικά, θέματα και χρώματα του project μας από τα σχέδια μας, ήρθε η ώρα να φτιάξουμε την κυρία οθόνη τής εφαρμογής.

4.3.2. Κύρια Οθόνη Activity

Για λόγους ευκολίας διαχείρισης κώδικα οθονών και χρήσης αρχιτεκτονικής MVVM, η κυρία οθόνη μας (Activity) θα λειτουργήσει ως container για υπό οθόνες (fragments). Η διαχείριση οθονών θα γίνεται με την βοήθεια της βιβλιοθήκη Navigation Component, άρα το αρχείο layout του Activity μας θα περιέχει ένα root element και μέσα σε αυτό θα έχουμε ένα στοιχείο FragmentContainerView για των έλεγχο και την προβολή των fragments στην οθόνη μας.

4.3.3. Οθόνη Fragment Αρχικό Μενού

Αναλύοντας τις κυρίες κατηγορίες των λειτουργιών της εφαρμογής μας προκύπτουν οι εξής μεγάλες κατηγορίες οθονών που θα πρέπει να έχει πρόσβαση το αρχικό μενού:

- Οθόνες Χωραφιών: εδώ είναι όλες οι λειτουργίες που σχετίζονται απευθείας στα χωράφια.
- Οθόνες Συμβάντων: εδώ είναι όλες οι λειτουργίες ημερολογίου.
- Οθόνες Προβολής: εδώ είναι όλες οι λειτουργίες προεπισκόπησης δεδομένων εφαρμογής.
- Οθόνες Ρυθμίσεων: εδώ είναι όλες οι λειτουργίες που σχετίζονται με τις ρυθμίσεις εφαρμογής.

Από διάφορα σχέδια που δημιουργήθηκαν του κυρίου μενού, αυτό πού ήταν πιο εύκολο στους χρήστες ήταν μια διεπαφή όπου το κύριο μενού αποτελείται από: μια επιλογή για τις ρύθμισης της εφαρμογής και οι υπόλοιπες κατηγορίες να αντιστοιχούν σε μεγάλα κουμπιά μέσα σε μια οριζόντια λίστα πού βρίσκεται κάτω από το κύριο μενού και επικαλύπτει το υπόλοιπο της οθόνης.

4.3.4. Οθόνες Fragments Χωραφιών

Η οθόνες χωραφιών θα αντιστοιχούν στις εξής:

- Οθόνη Μενού Χωραφιών: εδώ εμφανίζεται σε πλέγμα ή λίστα τα χωράφια μας. Εδώ έχουμε μερικά FAB για την δημιουργία, εξαγωγή ή διαγραφής χωραφιών. Στο μενού έχουμε δύο επιλογές: μία για να φιλτράρει με βάση τις ετικέτες των χωραφιών, μία για προβολή του ιστορικού όλων των χωραφιών.
- Οθόνη Προβολής Χωραφιού: εδώ απεικονίζεται σε χαρτί το επιλεγμένο χωράφι από το μενού χωραφιών. Περιέχει ένα FAB για την προβολή της λίστας τμημάτων του επιλεγμένου χωραφιού. Στο μενού έχει μια επιλογή προβολή ιστορικού του χωραφιού. Αν η οθόνη προβολής χωραφιού καλέστηκε από μια οθόνη ιστορικού τότε θα εμφανιστεί και μια επιλογή επαναφοράς του χωραφιού με βάση το στιγμιότυπο του ιστορικού.
- Οθόνη Πληροφοριών Χωραφιού: εδώ εμφανίζονται οι παραμέτρους των χωραφιών όπως τίτλος και ετικέτες. Χρησιμοποιούμε TextInputLayout για να μπορούμε να εμφανίσουμε τυχόν σφάλμα σαν λεζάντα από κάτω από τα πεδία.
- Οθόνη Επεξεργασίας Χωραφιού: αυτή η οθόνη είναι υπεύθυνη για την δημιουργία συνόρων τού χωραφιού, επίσης εδώ μπορούμε να πατήσουμε για εισαγωγή εικόνας για προβολή ως overlay πάνω στον χάρτη και την δυνατότητα εισαγωγής, προσθήκης ή αφαίρεσης γεωγραφικών συνόρων από ένα γεωγραφικό αρχείο.

- Οθόνη Εισαγωγής Χωραφιού: αυτή η οθόνη καλείται από την οθόνη επεξεργασίας χωραφιού και είναι υπεύθυνη για την επιλογή των γεωγραφικών συνόρων από τα γεωγραφικά σύνορα που διαβάστηκαν από ένα γεωγραφικό αρχείο.
- Οθόνη Μενού Τμημάτων: αυτή η οθόνη παρέχει τις ίδιες λειτουργίες της οθόνης μενού χωραφιών απλά εμφανίζει τα επιμέρους τμήματα του χωραφιού που επιλέξαμε προηγούμενος στο μενού χωραφιών.
- Οθόνη Επεξεργασίας Τμήματος: αυτή η οθόνη είναι υπεύθυνη για την δημιουργία και επεξεργασία τμημάτων χωραφιού.
- Οθόνη Ιστορικού Χωραφιών: σε αυτή την οθόνη προβάλλονται σε λίστα οι τίτλοι όλων των χωραφιών μαζί με την ενέργειες που έγιναν στο χωράφι.
- Οθόνη Ιστορικού Χωραφιού: σε αυτή την οθόνη προβάλετε μια λίστα ενεργειών που έγιναν σε ένα συγκεκριμένο χωράφι.

4.3.5. Οθόνες Fragments Συμβάντων

Η οθόνες Συμβάντων θα αντιστοιχούν στις εξής:

- Οθόνη Ημερολογίου Συμβάντων: σε αυτή την οθόνη προβάλλονται τα συμβάντα σε ημερολόγιο ή λίστα. Έχει δύο επιλογές στο μενού: μια για αλλαγή από λίστα σε ημερολόγιο και αντίστροφα, μια για φιλτράρισμα συμβάντων με βάση κατηγορίες και ημερομηνία. Έχουμε και ένα FAB για την δημιουργία νέου συμβάντος.
- Οθόνη Ημερομηνίας Συμβάντων: σε αυτή την οθόνη προβάλλονται τα συμβάντα της επιλεγμένης μέρας σε λίστα. Στο μενού έχει την επιλογή φιλτραρίσματος συμβάντων με βάση την κατηγορία τους. Περιέχει και ένα FAB για δημιουργία συμβάντος στην επιλεγμένη ημερομηνία.
- Οθόνη Επεξεργασίας Συμβάντος: αυτή η οθόνη είναι υπεύθυνη για την δημιουργία και επεξεργασίας των συμβάντων. Χρησιμοποιεί TextInputLayout για την εμφάνιση τυχόν σφαλμάτων.

4.3.6. Οθόνες Fragments Προβολής

Η οθόνες που θα σχετίζονται στην προβολή δεδομένων είναι:

- Οθόνη Προβολής Χάρτη: σε αυτή την οθόνη προβάλλονται όλα τα χωράφια με marker clustering μαζί με τις ζώνες τους. Στο μενού έχει την επιλογή να επαναφέρει την κάμερα στην αρχική θέση. Στον χάρτη έχει την επιλογή να ανοίξει την λειτουργία gps για εμφάνιση την τοποθεσίας του χρήστη και να ειδοποιείτε όταν μπει σε χωράφι ή τμήμα χωραφιού. Αν ο χρήστης κάνει κλικ πάνω στο Πολύγωνο του χωραφιού ή τμήματος, θα πλοηγηθεί στην οθόνη συμβάντων χωραφιού ή τμήματος.
- Οθόνη Συμβάντων Χωραφιού ή Τμήματος: εδώ εμφανίζονται όλα τα συμβάντα που έχουν σχέση με το επιλεγμένο χωράφι ή τμήμα χωραφιού. Στο μενού έχει μια επιλογή για φιλτράρισμα συμβάντων με βάση την κατηγορία τους και την ημερομηνία.
- Οθόνη Προεπιλεγμένης Εφαρμογής Γεωγραφικών Αρχείων: αυτή η οθόνη είναι υπεύθυνη για την προβολή γεωγραφικών συνόρων πού προέρχεται από γεωγραφικό αρχείο που ανοίχτηκε από το σύστημα, χρήστη ή browser.

4.3.7. Οθόνες Fragments Ρυθμίσεων Εφαρμογής

Αυτές οι οθόνες υπάρχουν για την ρύθμιση της εφαρμογής.

- Οθόνη Ρυθμίσεων Εφαρμογή: εδώ υπάρχουν όλες οι βασικές ρυθμίσεις της εφαρμογής. Στο μενού έχει δύο επιλογές: μια για επαναφορά ρυθμίσεων και μια για τις πληροφορίες της εφαρμογής. Εδώ επίσης μπορούμε να κάνουμε import / export την βάση δεδομένων της εφαρμογής σε αρχεία Excel, xls και xlsx.
- Οθόνη Κατηγοριών Συμβάντων Εφαρμογής: αυτή η οθόνη είναι υπεύθυνη για την απεικόνιση των κατηγοριών των συμβάντων σε πλέγμα ή λίστα. Έχει δύο FAB: ένα για δημιουργία νέας κατηγορίας και ένα για την διαγραφή των επιλεγμένων κατηγοριών.
- Οθόνη Επεξεργασίας Κατηγορίας Συμβάντος: αυτή η οθόνη είναι υπεύθυνη για την δημιουργία και επεξεργασίας των κατηγοριών συμβάντων. Χρησιμοποιεί TextInputLayout για την εμφάνιση τυχόν σφαλμάτων.
- Οθόνη Μαζικής Επεξεργασίας Χωραφιών και Τμημάτων: αυτή η οθόνη μπορεί να ενημερώσει μαζικά ιδιότητες των χωραφιών ή τμημάτων χωραφιών με βάση μια επιλεγμένη ετικέτα.
- Οθόνη Εμφάνισης Διαστάσεων Χωραφιών: αυτή η οθόνη εμφανίζει το συνολικό μέγεθος των χωραφιών και χρησιμοποιεί την αντίστοιχη μονάδα μέτρησης της χώρας που είναι επιλεγμένη στην συσκευή.

4.3.8. ViewHolders Αντικειμένων

Εφόσον τελειώσαμε με των σχεδιασμό των οθονών, πρέπει να δημιουργήσουμε μικρότερα layout αρχεία πού θα χρησιμοποιήσουμε για την απεικόνιση αντικειμένων σε λίστες. Αυτά τα layout ονομάζονται ViewHolders και χρησιμοποιούνται μέσα σε RecyclerView.

Ένα παράδειγμα ViewHolder layout:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
        <com.google.android.material.card.MaterialCardView android:id="@+id/cvCalendarCategory"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginHorizontal="16dp"
                android:layout_marginVertical="8dp">
                <LinearLayout
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:orientation="vertical"
                        android:padding="16dp" >
                        <TextView android:id="@+id/tvCalendarCategoryTitle"
                                android:layout_width="match_parent"
                                android:layout_height="wrap_content" />
                        <!-- other elements -->
                </LinearLayout>
        </com.google.android.material.card.MaterialCardView>
</androidx.constraintlayout.widget.ConstraintLayout>
```

4.3.9. Αρχεία Navigation

Εφόσον τελειώσαμε με την δημιουργία των αρχείων οθονών, ήρθε η ώρα να ορίσουμε τους δρόμους που μπορεί να κάνει η εφαρμογή μας. Με την βοήθεια του Navigation Component μπορούμε να φτιάξουμε αρχεία navigation και να τα ορίζουμε σε ένα FragmentContainerView. Ανοίγοντας ένα αρχείο navigation θα δούμε ότι έχουμε την δυνατότητα να προσθέσουμε όλα τα Fragments που υλοποιήσαμε, εφόσον περάσουν μπορούμε να ορίσουμε ποιο είναι το αρχικό fragment αλλά και τις διαδρομές των fragments, δηλαδή ποιό fragment μπορεί να οδηγήσει σε άλλο fragment και ποιό είναι αυτό, ένα fragment μπορεί να έχει όσες διαδρομές από και προς αυτό επιθυμούμε.

Ένα παράδειγμα αρχείου navigation nav-graph.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/nav_land_graph"
        app:startDestination="@id/LoadingFragment">

        <fragment android:id="@+id/MenuMainFragment"
                android:name="com.mosc.simo.ptuxiaki3741.ui.fragments.AppMainFragment"
                android:label="@string/fragment_menu"
                tools:layout="@layout/fragment_menu_main" >

                <!-- Menu Main fragment paths code -->

                <action android:id="@+id/toAppSettings"
                        app:destination="@id/AppSettingsFragment" />
        </fragment>
        <fragment android:id="@+id/AppSettingsFragment"
                android:name=
                "com.mosc.simo.ptuxiaki3741.ui.fragments.setting.AppSettingsFragment"
                android:label="@string/fragment_app_settings"
                tools:layout="@layout/fragment_app_settings" >

                <!-- Settings fragment paths code -->

        </fragment>

        <!-- other fragment code -->

</navigation>
```

4.4. Λειτουργίες Front End Εφαρμογής

Σε αυτή την ενότητα θα περιγράψουμε τι λειτουργίες χρειάστηκαν να υλοποιηθούν στις οθόνες.

4.4.1. Navigation Component

Νωρίτερα στην ενότητα: 4.3.2. Κύρια Οθόνη Activity αναφέραμε πως η κύρια οθόνη μας θα χρησιμοποιήσει την βιβλιοθήκη Navigation Component ως καμβά των fragments, ας δούμε τώρα πως μπορούμε να προβάλουμε fragments και να πλοηγηθούμε σε αυτά με την βοήθεια αυτής της βιβλιοθήκης:

1. Πρέπει να έχουμε δημιουργήσει τα αρχεία nav-graph.xml και activity_main.xml από τις ενότητες: 4.3.2. Κύρια Οθόνη Activity και 4.3.9. Αρχεία Navigation.
2. Αποθηκεύουμε ένα αντικείμενο τύπου NavHostFragment στο MainActivity.java που μπορούμε να χρησιμοποιούμε για ελεχνο ροής προγράμματος, ο κώδικας είναι ο εξής:

```
public class MainActivity extends AppCompatActivity {
        private NavHostFragment navHostFragment;
        private boolean doubleBackToExit = false;

        //code...

        @Override protected void onCreate(Bundle savedInstanceState) {
                //code...
                navHostFragment = (NavHostFragment) getSupportFragmentManager()
                        .findFragmentById(R.id.nhfMainNav);
        }

        @Override public void onBackPressed() {
                if(doubleBackToExit){
                        super.onBackPressed();
                }else{
                        FragmentManager fm = navHostFragment.getChildFragmentManager();
                        if(fm.getBackStackEntryCount() > 0){
                                navHostFragment.getNavController().popBackStack();
                        }else{
                                doubleBackToExit = true;
                                new Handler().postDelayed(
                                        () -> doubleBackToExit=false,
                                        AppValues.doubleTapBack
                                );
                                showSnackBar(getText(R.string.double_tap_exit_msg));
                        }
                }
        }
}
```

3. Για την περιήγηση μεταξύ fragments γίνεται ως εξής:

```
public class AppMainFragment extends Fragment{
        //code...
        public void toSettings(){
                NavController nav = NavHostFragment.findNavController(this);
                nav.navigate(R.id.toAppSettings);
        }
}
```

4.4.2. Recycler View Adapter με Diff Util

Για να προβάλουμε μια λίστα αντικειμένων στην οθόνη γνωρίζουμε ότι το Android χρησιμοποιεί τα αντικείμενα τύπου RecyclerView τα οποία περνώντας ένα αντικείμενο τύπου Adapter μπορούμε να περιγράψουμε των τρόπο απεικόνισης μιας λίστας αντικειμένων, ας δούμε ένα παράδειγμα ενός Adapter:

```
public class CalendarCategoriesAdapter
        extends RecyclerView.Adapter<CalendarCategoriesAdapter.CalendarCategoryViewHolder>
{
        private final List<CalendarCategoryModel> data;
        private final CalendarCategoriesListener listener;
        public CalendarCategoriesAdapter(CalendarCategoriesListener listener){
                data = new ArrayList<>();
                this.listener = listener;
        }
        @Override
        public CalendarCategoryViewHolder onCreateViewHolder(ViewGroup parent, int viewType){
                View view = LayoutInflater.from(parent.getContext())
                        .inflate(R.layout.calendar_categories_item, parent, false);
                return new CalendarCategoryViewHolder(view);
        }
        @Override
        public void onBindViewHolder(CalendarCategoryViewHolder viewHolder, final int position){
                CalendarCategoryModel element = data.get(position);
                viewHolder.title.setText(element.getName());
                viewHolder.cardView.setCardBackgroundColor(element.getColorData().getColor());
                if(UIUtil.showBlackText(element.getColorData())){
                        viewHolder.title.setTextColor(Color.BLACK);
                }else{
                        viewHolder.title.setTextColor(Color.WHITE);
                }
                viewHolder.cardView.setOnClickListener(v -> listener.onClick(element));
        }
        @Override public int getItemCount(){ return data.size(); }
        protected static class CalendarCategoryViewHolder extends RecyclerView.ViewHolder {
                public final MaterialCardView cardView;
                public final TextView title;
                public CalendarCategoryViewHolder(View view) {
                        super(view);
                        cardView = (MaterialCardView) view.findViewById(R.id.cvCalendarCategory);
                        title = (TextView) view.findViewById(R.id.tvCalendarCategoryTitle);
                }
        }
        public void saveData(List<CalendarCategoryModel> newData) {
                List<CalendarCategoryModel> oldData = new ArrayList<>(data);
                DiffUtil.DiffResult diffResult = DiffUtil.calculateDiff(
                        new CalendarCategoriesDiffUtilCallback(oldData,newData)
                );
                data.clear();
                data.addAll(newData);
                diffResult.dispatchUpdatesTo(this);
        }
}
```

Θα διαπιστώσουμε ότι το Adapter κάνει από μόνο του update στην μέθοδο saveData() με την βοήθεια του DiffUtil. Το DiffUtil είναι μια κλάση utility που υπολογίζει τη διαφορά μεταξύ δύο λιστών και εξάγει μια λίστα βημάτων ενημέρωσης που μετατρέπει την πρώτη λίστα στη δεύτερη, άρα εμείς καλώντας μονό την μέθοδο saveData() γλιτώνουμε μεθόδους notify στο Adapter.

Ας δούμε τώρα πως μπορούμε να δημιουργήσουμε ένα DiffUtilCallback:

```
public class CalendarCategoriesDiffUtilCallback extends DiffUtil.Callback {
        private final List<CalendarCategoryModel> oldData, newData;

        public DiffUtilCallback(
                List<CalendarCategoryModel> oldData,
                List<CalendarCategoryModel> newData
        ) {
                this.oldData = oldData;
                this.newData = newData;
        }

        @Override
        public int getOldListSize() {
                return oldData.size();
        }

        @Override
        public int getNewListSize() {
                return newData.size();
        }

        @Override
        public boolean areItemsTheSame(int oldItemPosition, int newItemPosition) {
                return oldData.get(oldItemPosition).getId() == newData.get(newItemPosition).getId();
        }

        @Override
        public boolean areContentsTheSame(int oldItemPosition, int newItemPosition) {
                CalendarCategoryModel oldElement = oldData.get(oldItemPosition);
                CalendarCategoryModel newElement = newData.get(newItemPosition);

                if(!oldElement.getName().equals(newElement.getName()))
                        return false;

                if(!oldElement.getColorData().equals(newElement.getColorData()))
                        return false;

                return true;
        }
}
```

4.4.3. Google Maps και Marker Clustering

Σε αυτή την ενότητα θα δούμε πως να εμφανίζουμε χωράφια στον χάρτη, πως να παίρνουμε την τοποθεσία που ο χρήστης έκανε click και τέλος πως να γίνει ομαδοποίηση χωραφιών με την χρήση Marker Clustering.

Έχοντας ένα αντικείμενο fragment SupportMapFragment ή ένα αντικείμενο MapView ο τρόπος προσκόμισης αντικειμένου GoogleMap παραμένει ίδιος, παίρνουμε το οπτικό αντικείμενο του χάρτη και εκτελούμε την μέθοδο getMapAsync(OnMapReadyCallback) όπου το OnMapReadyCallback εκτελείται όταν το αντικείμενο GoogleMap είναι έτοιμος προς χρήση.

Ας δούμε τώρα πως μπορούμε να εμφανίσουμε χωράφια στον χάρτη:

```
public class LandPreviewFragment extends Fragment {
	private GoogleMap mMap;
	private Polygon landPolygon;
	//code…
	@Override public void onViewCreated(View view, Bundle savedInstanceState) {
		super.onViewCreated(view, savedInstanceState);
		//fragment initialization code…
		MapView mapView = view.findViewById(R.id.mvMapView);
		mapView.getMapAsync(googleMaps->{
			mMap = googleMaps;
			//map initialization code…
		})
	}
	//code…
	private void drawLandOnMap(Land land) {
		if(mMap == null || land == null) return;
		clearLandOnMap();
		PolygonOptions options = new PolygonOptions();
		options.addAll(land.getBorder());
		for(List<LatLng> hole : land.getHoles()){
			options.addHole(hole);
		}
		options.strokeColor(Color.argb(
			AppValues.defaultStrokeAlpha,
			land.getColorData().getRed(),
			land.getColorData().getGreen(),
			land.getColorData().getBlue()
		));
		options.fillColor(Color.argb(
			AppValues.defaultFillAlpha,
			land.getColorData().getRed(),
			land.getColorData().getGreen(),
			land.getColorData().getBlue()
		));
		options.zIndex(AppValues.defaultLandIndex);
		landPolygon = mMap.addPolygon(options);
	}
	private void clearLandOnMap() {
		if(landPolygon == null) return;
		landPolygon.remove();
		landPolygon = null;
	}// εναλλακτικά τρέχουμε μονό το mMap.clear(); πλήρες καθαρισμό του χάρτη
}
```

Ας δούμε τώρα πως μπορούμε να παίρνουμε την τοποθεσία που ο χρήστης έκανε click στον χάρτη:

```
public class ZoneEditorFragment extends Fragment {
        //code…
        @Override public void onViewCreated(View view, Bundle savedInstanceState) {
                //code…
                MapView mapView = view.findViewById(R.id.mvMapView);
                mapView.getMapAsync(googleMaps->{
                        //map initialization code…
                        googleMaps.setOnMapClickListener(this::onMapClick);
                });
        }
        //code…
        private void onMapClick(LatLng point) {
                //onMapClick code…
        }
}
```

Ας δούμε τώρα πως μπορούμε να ομαδοποιούμε χωράφια με την χρήση Marker Clustering στον χάρτη, Για αρχή θα δημιουργήσουμε μια κλάση ClusterLand που να κάνει implement από την κλάση ClusterItem:

```
public class ClusterLand implements ClusterItem {
        private final Land land;
        private final List<LandZone> zones;

        public ClusterLand(Land land, List<LandZone> zones) {
                this.land = land;
                this.zones = zones;
        }

        @Override public LatLng getPosition() {
                if(land.getBorder().size()>0){
                        LatLngBounds.Builder builder = new LatLngBounds.Builder();
                        for(LatLng point : landData.getBorder()){
                                builder.include(point);
                        }
                        return builder.build().getCenter();
                }
                return new LatLng(0,0);
        }
        @Override public String getTitle() { return land.getTitle(); }
        @Override public String getSnippet() { return “”; }

        public Land getLand(){ return land; }
        public List<LandZone> getZones(){ return zones; }
}
```

Στην συνέχεια δημιουργούμε μια κλάση LandRendered που να κάνει extends από την κλάση DefaultClusterRenderer<ClusterLand>:

```
public class LandRendered extends DefaultClusterRenderer<ClusterLand> {
	private final Context ctx;
	private final GoogleMap map;
	private final Map<ClusterLand, List<Polygon>> polygons = new HashMap<>();
	public LandRendered(Context ctx, GoogleMap map, ClusterManager<ClusterLand> manager) {
		super(ctx, map, clusterManager);
		this.ctx = ctx;
		this.map = map;
	}
	@Override protected void onBeforeClusterItemRendered(ClusterLand i, MarkerOptions o) {
		removePolygons(i);
		addPolygons(i);
		setupMarkerOptions(i, o);
	}
	@Override
	protected void onBeforeClusterItemRendered(Cluster<ClusterLand> c, MarkerOptions o) {
		c.getItems().forEach(this::removePolygons);
		setupMarkerOptions(c, o);
	}
	@Override protected void onClusterItemUpdated(ClusterLand i, Marker m) {
		setupMarkerOptions(i, m);
	}
	@Override protected void onClusterUpdated(Cluster<ClusterLand> c, Marker m) {
		setupMarkerOptions(c, m);
	}
	private void removePolygons(ClusterLand i) {
		List<Polygon> itemPolygons = this.polygons.getOrDefault(item,null);
		if(itemPolygons == null) return;
		for(Polygon polygon : itemPolygons){ polygon.remove(); }
		this.polygons.remove(item);
	}
	private void addPolygons(ClusterLand i) {
		List<Polygon> itemPolygons = new ArrayList<>();
		itemPolygons.add(createLandPolygon(i.getLand()));
		for(LandZone zone: i.getZones()){ itemPolygons.add(createZonePolygon(zone)); }
		this.polygons.put(i, itemPolygons);
	}
	private void setupMarkerOptions(ClusterLand i, MarkerOptions o) {
		//code to change cluster item marker options…
	}
	private void setupMarkerOptions(ClusterLand i, Marker m) {
		//code to change cluster item marker …
	}
	private void setupMarkerOptions(Cluster<ClusterLand> c, MarkerOptions o) {
		//code to change cluster marker options…
	}
	private void setupMarkerOptions(Cluster<ClusterLand> c, Marker m) {
		//code to change cluster marker …
	}
	private Polygon createLandPolygon(Land land){ //code to create polygon from map… }
	private Polygon createZonePolygon(LandZone zone){ //code to create polygon from map… }
}
```

Τέλος δημιουργούμε ένα ClusterManager σε μια οθόνη με των εξής τρόπο:

```
public class ZoneEditorFragment extends Fragment {
        private static final int Min_Cluster_Size = 2;
        private static final int Distance_Between_Clustered_Items = 60;

        //code…

        private ClusterManager<ClusterLand> clusterManager;

        //code…

        @Override public void onViewCreated(View view, Bundle savedInstanceState) {
                //code…
                MapView mapView = view.findViewById(R.id.mvMapView);
                mapView.getMapAsync(googleMaps->{
                        //code…
                        clusterManager = new ClusterManager<>(getActivity(), googleMaps);
                        LandRendered renderer = new LandRendered(
                                getActivity(),
                                googleMaps,
                                clusterManager
                        );
                        renderer.setMinClusterSize(Min_Cluster_Size);
                        clusterManager.setRenderer(renderer);
                        NonHierarchicalDistanceBasedAlgorithm<ClusterLand> algorithm =
                                new NonHierarchicalDistanceBasedAlgorithm<>();
                        algorithm.setMaxDistanceBetweenClusteredItems(
                                Distance_Between_Clustered_Items
                        );
                        clusterManager.setAlgorithm(algorithm);
                        clusterManager.setOnClusterItemClickListener(this::onClusterItemClick);
                        clusterManager.setOnClusterClickListener(this::onClusterClick);
                });
        }

        //code…

        private void drawOnMap(Map<Land, List<LandZone>> data) {
                if(clusterManager == null) return;
                clusterManager.clearItems();
                data.forEach((land,zones) ->
                        clusterManager.addItem(new ClusterLand(land,zones))
                );
                clusterManager.cluster();
        }
        private boolean onClusterItemClick(ClusterLand item) {
                //onClusterItemClick code…
                return true;
        }
        private boolean onClusterClick(Cluster<ClusterLand> cluster) {
                //onClusterClick code…
                return true;
        }
}
```

5. Εκπληρωμένοι Στόχοι Και Σύνοψη Εργασίας

5.1. Εκπληρωμένοι Στόχοι

Σε αυτή την ενότητα θα δούμε το τέταρτο στάδιο σχεδίασης εφαρμογών, η **λειτουργία**. Σε αυτό το στάδιο δοκιμάζουμε και τρέχουμε όλες τις λειτουργίες της εφαρμογής μας, για να σιγουρευτούμε ότι όλα λειτουργούν κανονικά. Έχοντας επαλήθευση την λειτουργία της εφαρμογή μας, ας δούμε τι λειτουργίες καταφέραμε να φτιάξουμε στην εφαρμογή:

- ➔ Δημιουργία και διαχείριση χωραφιών.
- ➔ Δημιουργία και διαχείριση τμημάτων χωραφιών.
- ➔ Χωρισμός χωραφιών ανά έτους, με διάλογο εισαγωγής χωραφιών από το προηγούμενο έτος όταν αλλάξει η χρονιά.
- ➔ Εισαγωγή και εξαγωγή χωραφιών σε γεωγραφικά αρχεία.
- ➔ Εξαγωγή τμημάτων χωραφιού σε γεωγραφικά αρχεία.
- ➔ Πράξεις μεταξύ συνόλων γεωγραφικών σημείων, όπως αφαίρεση ή πρόσθεση σε χωράφι από γεωγραφικό αρχείο.
- ➔ Εισαγωγή και εξαγωγή χωραφιών και τμημάτων σε αρχεία Excel.
- ➔ Φίλτρα χωραφιών και ζωνών με βάση ετικετών.
- ➔ Διατήρηση ιστορικών γεγονότων χωραφιών και δυνατότητα επαναφοράς αυτού.
- ➔ Δημιουργία και διαχείριση συμβάντων σε χωράφια και ζώνες μαζί με ειδοποίηση αυτού.
- ➔ Ημερολόγιο συμβάντων με φίλτρα τις κατηγορίες συμβάντων και την ημερομηνία.
- ➔ Οθόνη προβολής χωραφιών μαζί με τις ζώνες τους, που αν πατήσει ο χρήστης πάνω σε χωράφι ή σε μια ζώνη εμφανίζει τα συμβάντα εκείνης της περιοχής. Επίσης διαβάζει την τοποθεσία του χρήστη και ειδοποιεί όταν μπει σε μια ζώνη ή ένα χωράφι.
- ➔ Οθόνη για ανάγνωση γεωγραφικών αρχείων που να ξεκινάει ως προεπιλεγμένο πρόγραμμα ανάγνωσης αυτών από το σύστημα.

5.2. Σύνοψη Εργασίας

Μετά από αυτήν την εργασία έχουμε μια πιο σαφή εικόνα για πως να υλοποιηθεί μια εφαρμογή σε Android, επίσης είδαμε την λογική πού κρύβεται πίσω από έναν κύκλο ανάπτυξης εφαρμογής, δηλαδή ανάλυση, σχεδίαση, υλοποίηση, λειτουργία και συντήρηση. Έχοντας διαβάσει την λογική πού ακολουθήθηκε για των σχεδιασμό και της υλοποίησης της εφαρμογής μαζί με των πηγαίο κώδικα του, μάθαμε μερικά βασικά στοιχεία όπως:

- ★ Πώς να προβάλουμε ένα αγρόκτημα σε χαρτί με την χρήση τού Google maps.
- ★ Την δομή των βασικών γεωγραφικών αρχείων (KML, GML, GeoJSON).
- ★ Των τρόπο εισαγωγής και εξαγωγής δεδομένων σε αρχεία XML, JSON, Open XML Sheet.
- ★ Πώς να κάνουμε πράξεις συνόλων γεωγραφικών σημείων.
- ★ Πώς να κάνουμε clustering διάφορα σημεία ενός χάρτη Google maps.
- ★ Των τρόπο αποθήκευσης και προσκόμισης δεδομένων σε τοπική βάση δεδομένων.
- ★ Των τρόπο μετατροπής μεταξύ συστημάτων συντεταγμένων.
- ★ Την μέθοδο πλοήγησης οθονών με χρήση το Navigation Component.

6. Μελλοντικές Ενημερώσεις

Σε αυτή την ενότητα θα δούμε το πέμπτο στάδιο σχεδίασης εφαρμογών, την **συντήρηση**. Σε αυτό το στάδιο θα δούμε τής μελλοντικές αναβαθμίσεις τής εφαρμογής μας.

6.1. Version 1.5: Online Features

Η πρώτη μελλοντική αναβάθμιση της εφαρμογής θα αφορά την δημιουργία πλάνων χρηστών, των συγχρονισμό δεδομένων μεταξύ συσκευών μέσου internet και προβολής μετεωρολογικής πρόβλεψης καιρού των χωραφιών.

6.1.1. Users Update

Ο χρήστης θα έχει την επιλογή να δημιουργήσει έναν δωρεάν λογαριασμό ή να χρησιμοποίηση την εφαρμογή ως επισκέπτης, στην περίπτωση που ο χρήστης φτιάξει έναν λογαριασμό θα έχει την δυνατότητα πληρωμής μίας ετήσιας συνδρομής για να αποκτήσει πρόσβαση σε ποιο προχωρημένες λειτουργίες της εφαρμογής μας.

Τα πλάνα χρηστών είναι τα εξής:

Guest User	Free Account	Vip Account
Οι επισκέπτες θα έχουν τις εξής λειτουργίες:	Το δωρεάν πλάνο θα έχει τις εξής λειτουργίες:	Το vip πλάνο θα έχει τις εξής λειτουργίες:
➢ Διαχείριση τμημάτων αγροκτήματος. ➢ Διαχείριση ημερολογίου τμημάτων. ➢ Διαχείριση καλλιεργειών. (ver:2.0) ➢ Διαχείριση εξοπλισμού και προμηθειών. (ver:3.0)	➢ Όλες οι λειτουργίες του “Guest User” ➢ Συγχρονισμό δεδομένων μεταξύ συσκευών. ➢ Τιμές σοδειών περασμένων ετών. (ver:2.0)	➢ Όλες οι λειτουργίες του “Free Plan” ➢ Προβλέψεις καιρού χωραφιών. ➢ Τιμές σοδειών τωρινού έτους. (ver:2.0) ➢ Επικοινωνία με έξυπνες συσκευές. (ver:4.0)

6.1.2. Data Sync Update

Το “Data Sync Update” θα συσχετίζετε με την σύνδεση της εφαρμογής με μια εξωτερική βάση δεδομένων. Η τοπική βάση δεδομένων θα μετατραπεί σε cache της εξωτερικής βάσης δεδομένων. Καταφέρνοντας έτσι των συγχρονισμό των δεδομένων μεταξύ των συσκευών του χρήστη και την δυνατότητα διατήρησης των δεδομένων σε τοποθεσίες όπου το δίκτυο είναι πιο αδύνατο.

6.1.3. Land Meteo Update

Το “Land Meteo Update” θα συσχετίζετε με την προβολή πρόβλεψης καιρού στα χωράφια. Στην ενότητα “Live Map” της εφαρμογής μας εκτός της προβολής συμβάντων των ζωνών και χωραφιών μας, θα προστεθεί μια νέα ενότητα προβολής μετεωρολογικών δεδομένων αυτών.

6.2. Version 2.0: Crops And Market Update

Σκοπός της έκδοσης 2.0 είναι η εισαγωγή διαχείρισης καλλιεργειών στην εφαρμογή. Σε αυτή την έκδοση θα υπάρχουν δύο (2) κύριες ενημερώσεις: το “Crop Management Update” και το “Country Market Pricing Update”.

6.2.1. Crop Management Update

Στο “Crop Management Update” ο χρήστης θα μπορεί να ορίζει τα σύνορα και πληροφορίες των καλλιεργειών του, όπως των τύπο καλλιέργειας, κόστος συντήρησης και κέρδος σοδειάς είναι μερικές βασικές πληροφορίες. Με αυτό των τρόπο ο χρήστης θα μπορεί να διατηρήσει ένα ιστορικό υπόβαθρο όλων των καλλιεργειών του.

6.2.2. Country Market Pricing Update

Με την έγκριση του χρήστη, στο “Country Market Pricing Update”, η εφαρμογή θα συλλέγει τις πληροφορίες των καλλιεργειών ή θα τις παίρνουμε από τρίτες πηγές και θα παρέχονται στην εφαρμογή μας σύμφωνα με το εκάστοτε πακέτο του χρήστη.

6.3. Version 3.0: Land Equipment Update

Σκοπός της έκδοσης 3.0 είναι η εισαγωγή διαχείρισης εξοπλισμού και προμηθειών στην εφαρμογή μας η οποία θα παρέχεται ανεξαρτήτως του πακέτο χρήστη. Σε αυτή την έκδοση θα υπάρχουν δύο (2) κύριες ενημερώσεις: το “Tools Management Update” και το “Supplies Management Update”.

6.3.1. Tools Management Update

Στο “Tools Management Update” ο χρήστης θα μπορεί πλέον να ορίζει των εξοπλισμό του μαζί με της κατηγορίες τους, π.χ.: μηχανήματα και υποκατηγορίες μηχανημάτων, εργαλεία και τύπου εργαλείου, κ.α. Ο χρήστης θα μπορεί επίσης να γράφει και σημειώσεις στων εξοπλισμό του, για την διατήρηση ιστορικών γεγονότων.

6.3.2. Supplies Management Update

Στο “Supplies Management Update” ο χρήστης θα μπορεί να ορίζει τις προμήθειες του μαζί με τις κατηγορίες τους, όπως: λιπάσματα, φυτοφάρμακα και άλλα. Ο χρήστης θα μπορεί επίσης να ορίζει λεπτομέρειες σε αυτές τις προμήθειες, όπως ποσότητες και τιμή ανά ποσότητα, μαζί με τις ημερομηνίες αγοράς προμηθειών, για να μπορεί να έχει ένα ολοκληρωμένο υπόβαθρο των αποθηκών του.

6.4. Version 4.0: Smart Tools Communication Update

Σκοπός της έκδοσης 4.0 είναι η επικοινωνία της εφαρμογής με των έξυπνο εξοπλισμό του αγρότη, εφόσον βέβαια συνεργάζεται η εφαρμογή με των κατασκευαστή. Με αυτών των τρόπο ο εξοπλισμός του θα παίρνει οδηγίες είτε εντός μιας οθόνης είτε από την τοποθεσία του αγρότη.

Παραρτήματα

Παράρτημα Αρχείων KML

Αναλυτής Αρχείων KML

```
public final class KmlParser {

private KmlParser(){}

@NonNull
public static List<MapsLand> parse(
        InputStream input
) throws ParserConfigurationException, IOException, SAXException {
        List<MapsLand> result = new ArrayList<>();
         if(input == null) return result;
         DocumentBuilder docBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
        Document document = docBuilder.parse(input);
        NodeList polygons= document.getElementsByTagName("Polygon");
        for(int i = 0; i < polygons.getLength(); i++){
                if(polygons.item(i).getNodeType() != Node.ELEMENT_NODE) continue;
                Element polygon = (Element) polygons.item(i);
                MapsLand temp = readPolygon(polygon);
                if(temp != null) result.add(temp);
        }
        return result;
}

private static MapsLand readPolygon( Element polygon ){
        return new MapsLand(
                readBorder(polygon),
                readHoles(polygon)
        );
}

private static List<LatLng> readBorder( Element polygon ){
        NodeList outerBoundaryIsList = polygon.getElementsByTagName("outerBoundaryIs");
        if(outerBoundaryIsList.getLength() == 0) return new ArrayList<>();
        if(outerBoundaryIsList.item(0).getNodeType() != Node.ELEMENT_NODE)
                return new ArrayList<>();
        Element outerBoundaryIs = (Element) outerBoundaryIsList.item(0);
        List<List<LatLng>> result = readCoordinates(outerBoundaryIs);
        if(result.size() > 0) return result.get(0);
        return new ArrayList<>();
}

private static List<List<LatLng>> readHoles( Element polygon ){
        NodeList innerBoundaryIsList = polygon.getElementsByTagName("innerBoundaryIs");
        if(innerBoundaryIsList.getLength() == 0) return new ArrayList<>();
        if(innerBoundaryIsList.item(0).getNodeType() != Node.ELEMENT_NODE)
                return new ArrayList<>();
        Element innerBoundaryIs = (Element) innerBoundaryIsList.item(0);
        return readCoordinates(innerBoundaryIs);
}
```

```
private static List<List<LatLng>> readCoordinates( Element root ){
	List<List<LatLng>> result = new ArrayList<>();
	List<LatLng> temp = new ArrayList<>();
	NodeList coordinatesList = root.getElementsByTagName("coordinates");
	for(int i = 0; i < coordinatesList.getLength(); i++){
		temp.clear();
		String[] coordinatePairsList = coordinatesList.item(i) .getFirstChild().getNodeValue()
			.trim().replaceAll("\n","").replaceAll("\t","").replaceAll(" +", " ") .split(" ");
		for(String coordinatePairs : coordinatePairsList){
			String[] coordinates = coordinatePairs.split(",");
			if(coordinates.length < 2) continue;
			temp.add(new LatLng(
				Double.parseDouble(coordinates[1]),
				Double.parseDouble(coordinates[0])
			));
		}
		if(temp.size()>0) result.add(temp);
	}
	return result;
}

}
```

Εξαγωγή Αρχείων KML

```
public final class KmlExporter {

private static final String TAG = "KmlExporter";

private KmlExporter(){}

public static boolean export(List<MapsLand> mapsEntries, OutputStream fileOutputStream){
	try{
		Document doc = KmlExporter.export(mapsEntries);
		TransformerFactory tf = TransformerFactory.newInstance();
		Transformer trans = tf.newTransformer();
		trans.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
		trans.setOutputProperty(OutputKeys.INDENT, "yes");
		trans.transform(new DOMSource(doc), new StreamResult(fileOutputStream));
		return true;
	} catch (ParserConfigurationException | TransformerException e) {
		Log.e(TAG, "exportKmlMaps: ", e);
		return false;
	}
}

private static Document export(
	List<MapsLand> landList
) throws ParserConfigurationException {
	DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
	factory.setNamespaceAware(true);
	DocumentBuilder builder = factory.newDocumentBuilder();
	Document doc = builder.newDocument();
	if(landList.size() == 0) return doc;
	Element kmlTag = doc.createElement("kml");
	kmlTag.setAttribute("xmlns","http://www.opengis.net/kml/2.2");
	kmlTag.setAttribute("xmlns:kml","http://www.opengis.net/kml/2.2");
	kmlTag.setAttribute("xmlns:gx","http://www.google.com/kml/ext/2.2");
	kmlTag.setAttribute("xmlns:atom","http://www.w3.org/2005/Atom");
	Element documentTag = doc.createElement("Document");
	Element nameTag = doc.createElement("name");
	nameTag.appendChild(doc.createTextNode("Lands"));
	documentTag.appendChild(nameTag);
	Element temp;
	for(MapsLand land:landList){
		temp = getLandPlaceMark(doc,land);
		if(temp != null) documentTag.appendChild(temp);
	}
	kmlTag.appendChild(documentTag);
	doc.appendChild(kmlTag);
	return doc;
}
```

```
private static Element getLandPlaceMark(Document doc, MapsLand land){
	if(land.getBorder().size() == 0) return null;
	Element placemarkTag = doc.createElement("Placemark");
	Element placemarkNameTag = doc.createElement("name");
	placemarkNameTag.appendChild(doc.createTextNode("Land"));
	placemarkTag.appendChild(placemarkNameTag);
	Element polygonTag = doc.createElement("Polygon");
	Element extrudeTag = doc.createElement("extrude");
	extrudeTag.appendChild(doc.createTextNode("1"));
	polygonTag.appendChild(extrudeTag);
	Element altitudeModeTag = doc.createElement("altitudeMode");
	altitudeModeTag.appendChild(doc.createTextNode("relativeToGround"));
	polygonTag.appendChild(altitudeModeTag);
	Element outerBoundaryIsTag = doc.createElement("outerBoundaryIs");
	Element LinearRingTag = doc.createElement("LinearRing");
	Element coordinatesTag = doc.createElement("coordinates");
	coordinatesTag.appendChild(
		doc.createTextNode(getCoordinatesString(land.getBorder()))
	);
	LinearRingTag.appendChild(coordinatesTag);
	outerBoundaryIsTag.appendChild(LinearRingTag);
	polygonTag.appendChild(outerBoundaryIsTag);
	if(land.getHoles().size() != 0){
		Element innerBoundaryIsTag = doc.createElement("innerBoundaryIs");
		for(List<LatLng> hole : land.getHoles()){
			LinearRingTag = doc.createElement("LinearRing");
			coordinatesTag = doc.createElement("coordinates");
			coordinatesTag.appendChild(
				doc.createTextNode(getCoordinatesString(hole))
			);
			LinearRingTag.appendChild(coordinatesTag);
			innerBoundaryIsTag.appendChild(LinearRingTag);
		}
		polygonTag.appendChild(innerBoundaryIsTag);
	}
	placemarkTag.appendChild(polygonTag);
	return placemarkTag;
}

private static String getCoordinatesString(List<LatLng> points){
	StringBuilder coordinatesValues = new StringBuilder();
	LatLng temp;
	for(int i = 0; i < points.size(); i++){
		temp = points.get(i);
		if(i != 0) coordinatesValues.append(" ");
		coordinatesValues.append(temp.longitude).append(",")
			.append(temp.latitude).append(",").append(0);
	}
	return coordinatesValues.toString();
}

}
```

Παράρτημα Αναλυτή Αρχείων GML

```
public final class GmlParser {

private GmlParser(){}

@NonNull
public static List<MapsLand> parse(
        InputStream input
) throws ParserConfigurationException, IOException, SAXException {
        List<MapsLand> result = new ArrayList<>();
        if(input == null) return result;
        DocumentBuilder docBuilder = DocumentBuilderFactory.newInstance()
                .newDocumentBuilder();
        Document document = docBuilder.parse(input);
        NodeList polygons = document.getElementsByTagName("gml:Polygon");
        for(int i = 0; i < polygons.getLength(); i++){
                if(polygons.item(i).getNodeType() != Node.ELEMENT_NODE) continue;
                Element polygon = (Element) polygons.item(i);
                MapsLand temp = readPolygon(polygon);
                if(temp != null) result.add(temp);
        }
        return result;
}

private static MapsLand readPolygon( Element polygon ) {
        MapsLand temp = new MapsLand(
                readBorder(polygon),
                readHoles(polygon)
        );
        if(temp.getBorder().size() > 0) return temp;
        return null;
}

private static List<LatLng> readBorder( Element polygon ) {
        NodeList bordersTags = polygon.getElementsByTagName("gml:outerBoundaryIs");
        if(bordersTags.getLength() == 1)
                if(bordersTags.item(0).getNodeType() == Node.ELEMENT_NODE)
                        return readGml2Border((Element) bordersTags.item(0));
        bordersTags = polygon.getElementsByTagName("gml:exterior");
        if(bordersTags.getLength() == 1)
                if(bordersTags.item(0).getNodeType() == Node.ELEMENT_NODE)
                        return readGml31Border((Element) bordersTags.item(0));
        return new ArrayList<>();
}
private static List<LatLng> readGml2Border( Element borderTag ) {
        List<List<LatLng>> pointsList = readGml2Coordinates(borderTag);
        if(pointsList.size() != 1) return new ArrayList<>();
        return pointsList.get(0);
}
private static List<LatLng> readGml31Border( Element borderTag ) {
        List<List<LatLng>> pointsList = readGml31Coordinates(borderTag);
        if(pointsList.size() != 1) return new ArrayList<>();
        return pointsList.get(0);
}
```

```
private static List<List<LatLng>> readHoles( Element polygon ) {
	NodeList holesTags = polygon.getElementsByTagName("gml:innerBoundaryIs");
	if(holesTags.getLength() == 1)
		if(holesTags.item(0).getNodeType() == Node.ELEMENT_NODE)
			return readGml2Coordinates((Element) holesTags.item(0));
	holesTags = polygon.getElementsByTagName("gml:interior");
	if(holesTags.getLength() == 1)
		if(holesTags.item(0).getNodeType() == Node.ELEMENT_NODE)
			return readGml31Coordinates((Element) holesTags.item(0));
	return new ArrayList<>();
}

private static List<List<LatLng>> readGml2Coordinates( Element pointsRoot ) {
	List<List<LatLng>> result = new ArrayList<>();
	List<LatLng> temp;
	NodeList linearRings = pointsRoot.getElementsByTagName("gml:LinearRing");
	for(int i = 0; i < linearRings.getLength(); i++){
		if(linearRings.item(i).getNodeType() != Node.ELEMENT_NODE) continue;
		Element linearRing = (Element) linearRings.item(i);
		NodeList coordinatesList = linearRing.getElementsByTagName("gml:coordinates");
		for(int j = 0; j < coordinatesList.getLength(); j++){
			if(coordinatesList.item(j).getNodeType() != Node.ELEMENT_NODE) continue;
			Element coordinates = (Element) coordinatesList.item(j);
			String decimal=".",cs=",",ts=" ";
			if(coordinates.hasAttribute("decimal"))
				decimal = coordinates.getAttribute("decimal");
			if(coordinates.hasAttribute("cs"))
				cs = coordinates.getAttribute("cs");
			if(coordinates.hasAttribute("ts"))
				ts = coordinates.getAttribute("ts");
			String[] coordinatePairs = coordinates.getFirstChild().getNodeValue().trim()
				.replace(decimal,".").replace(cs,",").replace(ts," ")
				.replaceAll("\n"," ").replaceAll("\t"," ").replaceAll(" +"," ").split(" ");
			temp = new ArrayList<>();
			for(String coordinatePair : coordinatePairs){
				String[] coordinate = coordinatePair.split(",");
				if(coordinate.length < 2) continue;
				temp.add(new LatLng(
					Double.parseDouble(coordinate[1]),
					Double.parseDouble(coordinate[0])
				));
			}
			if(temp.size()>0) result.add(temp);
		}
	}
	return result;
}
```

```
private static List<List<LatLng>> readGml31Coordinates( Element pointsRoot ) {
	List<List<LatLng>> result = new ArrayList<>();
	List<LatLng> temp;
	NodeList linearRings = pointsRoot.getElementsByTagName("gml:LinearRing");
	int srsDimension;
	for(int i = 0; i < linearRings.getLength(); i++){
		if(linearRings.item(i).getNodeType() != Node.ELEMENT_NODE) continue;
		Element linearRing = (Element) linearRings.item(i);
		srsDimension = 2;
		if(linearRing.hasAttribute("srsDimension"))
			srsDimension = Integer.parseInt(linearRing.getAttribute("srsDimension"));
		if(srsDimension < 2) continue;
		NodeList coordinatesList = linearRing.getElementsByTagName("gml:posList");
		for(int j = 0; j < coordinatesList.getLength(); j++){
			if(coordinatesList.item(j).getNodeType() != Node.ELEMENT_NODE) continue;
			Element coordinates = (Element) coordinatesList.item(j);
			String[] coordinatePoints = coordinates.getFirstChild().getNodeValue().trim()
				.replaceAll("\n"," ").replaceAll("\t"," ").replaceAll(" +"," ")
				.replace(","."").split(" ");
			if(srsDimension > coordinatePoints.length) continue;
			temp = new ArrayList<>();
			for(int z = 0; z < coordinatePoints.length; z = z + srsDimension){
				temp.add(new LatLng(
					Double.parseDouble(coordinatePoints[z+1]),
					Double.parseDouble(coordinatePoints[z])
				));
			}
			if(temp.size()>0) result.add(temp);
		}
	}
	return result;
}

}
```

Παράρτημα Αρχείων GeoJSON

Αναλυτής Αρχείων GeoJSON

```
public final class GeoJsonParser {

private GeoJsonParser(){}

@NonNull
public static List<MapsLand> parse(
        InputStream input
) throws IOException, JSONException {
        List<MapsLand> result = new ArrayList<>();
        if(input == null) return result;
        JSONObject jsonObject = new JSONObject(inputSteamToString(input));
        if(!jsonObject.has("type")) return result;
        JSONObject geometry;
        String type = jsonObject.getString("type");
        if(type.equalsIgnoreCase("featurecollection")){
                JSONArray features = jsonObject.getJSONArray("features");
                for(int i = 0; i < features.length(); i++) {
                        geometry = features.getJSONObject(i).getJSONObject("geometry");
                        readGeometry(geometry, result);
                }
        }else if(type.equalsIgnoreCase("feature")){
                geometry = jsonObject.getJSONObject("geometry");
                readGeometry(geometry, result);
        }
        return result;
}

private static String inputSteamToString(InputStream input) throws IOException {
        BufferedReader streamReader = new BufferedReader(
                new InputStreamReader(input, StandardCharsets.UTF_8)
        );
        StringBuilder responseStrBuilder = new StringBuilder();
        String inputStr;
        while ((inputStr = streamReader.readLine()) != null) {
                responseStrBuilder.append(inputStr);
        }
        streamReader.close();
        return responseStrBuilder.toString();
}

private static void readGeometry(
        JSONObject geometry,
        List<MapsLand> result
) throws JSONException {
        String geometryType = geometry.getString("type").toLowerCase();
        if(geometryType.equals("polygon")) {
                MapsLand temp = readPolygon(geometry);
                if(temp != null) result.add(temp);
        }else if(geometryType.equals("multipolygon")) {
                readMultiPolygon(geometry, result);
        }
}
```

```
private static MapsLand readPolygon(
        JSONObject polygon
) throws JSONException {
        JSONArray coordinatesArray = polygon.getJSONArray("coordinates");
        return getPolygon(coordinatesArray);
}

private static void readMultiPolygon(
        JSONObject multiPolygon,
        List<MapsLand> result
) throws JSONException {
        JSONArray coordinatesArray = multiPolygon.getJSONArray("coordinates");
        MapsLand temp;
        for(int i = 0; i < coordinatesArray.length(); i++){
                temp = getPolygon(coordinatesArray.getJSONArray(i));
                if(temp != null) result.add(temp);
        }
}

private static MapsLand getPolygon(
        JSONArray jsonArray
) throws JSONException {
        List<List<LatLng>> points  = new ArrayList<>();
        List<LatLng> temp;
        for(int i = 0; i < jsonArray.length(); i++){
                temp = new ArrayList<>();
                for(int j = 0; j < jsonArray.getJSONArray(i).length(); j++){
                        if(jsonArray.getJSONArray(i).getJSONArray(j).length() > 1){
                                temp.add(new LatLng(
                                        jsonArray.getJSONArray(i).getJSONArray(j).getDouble(1),
                                        jsonArray.getJSONArray(i).getJSONArray(j).getDouble(0)
                                ));
                        }
                }
                points.add(temp);
        }
        if(points.size() == 1)
                return new MapsLand(
                        points.get(0),
                        new ArrayList<>()
                );
        else if(points.size() > 1)
                return new MapsLand(
                        points.get(0),
                        points.subList(1,points.size())
                );
        return null;
}

}
```

Εξαγωγή Αρχείων GeoJSON

```
public final class GeoJsonExporter {

private static final String TAG = "GeoJsonExporter";

private GeoJsonExporter(){}

public static boolean export(List<MapsLand> mapsEntries, OutputStream fileOutputStream){
	try{
		JSONObject object = GeoJsonExporter.export(mapsEntries);
		PrintWriter writer = new PrintWriter(fileOutputStream);
		writer.print(object.toString(2));
		writer.close();
		return true;
	} catch (JSONException e) {
		Log.e(TAG, "exportGeoJsonMaps: ", e);
		return false;
	}
}

private static JSONObject export(
	List<MapsLand> landList
) throws JSONException {
	if(landList.size() == 0) return new JSONObject();
	List<List<List<LatLng>>> pointsToExtract = new ArrayList<>();
	List<List<LatLng>> tempPoints;
	for(MapsLand land:landList){
		if(land == null) continue;
		if(land.getBorder().size() == 0) continue;
		tempPoints = new ArrayList<>(land.getHoles());
		tempPoints.add(0,land.getBorder());
		pointsToExtract.add(tempPoints);
	}
	if(pointsToExtract.size() == 0) return new JSONObject();
	else if(pointsToExtract.size() == 1) return getFeature(pointsToExtract.get(0));
	else return getFeatureCollection(pointsToExtract);
}

private static JSONObject getFeatureCollection(
	List<List<List<LatLng>>> coordinatesList
) throws JSONException {
	JSONObject root = new JSONObject();
	root.put("type","FeatureCollection");
	JSONArray features = new JSONArray();
	for(List<List<LatLng>> coordinates : coordinatesList){
		features.put(getFeature(coordinates));
	}
	root.put("features",features);
	return root;
}
```

```
private static JSONObject getFeature(
        List<List<LatLng>> coordinates
) throws JSONException {
        JSONObject root = new JSONObject();
        root.put("type","Feature");
        root.put("geometry",getGeometry(coordinates));
        return root;
}

private static JSONObject getGeometry(
        List<List<LatLng>> coordinates
) throws JSONException {
        JSONObject root = new JSONObject();
        root.put("type","Polygon");
        root.put("coordinates",getCoordinates(coordinates));
        return root;
}

private static JSONArray getCoordinates(
        List<List<LatLng>> coordinatesList
) throws JSONException {
        JSONArray root = new JSONArray();
        JSONArray row, points;
        for(List<LatLng> coordinates:coordinatesList){
                row = new JSONArray();
                for(LatLng coordinate:coordinates){
                        points = new JSONArray();
                        points.put(0,coordinate.longitude);
                        points.put(1,coordinate.latitude);
                        row.put(points);
                }
                root.put(row);
        }
        return root;
}

}
```

Βιβλιογραφία

- https://developer.android.com/studio
- https://www.figma.com
- https://material.io/blog/material-theme-builder
- https://m3.material.io
- https://developer.android.com/training/data-storage/room
- https://developers.google.com/maps/documentation/android-sdk
- https://developers.arcgis.com/documentation
- https://developer.android.com/guide/navigation
- https://firebase.google.com/docs/crashlytics
- https://sourceforge.net/projects/javashapefilere
- http://www.jdom.org/downloads/docs.html
- https://trac.osgeo.org/proj4j
- https://poi.apache.org
- https://github.com/kizitonwose/CalendarView
- https://github.com/thuytrinh/android-collage-views
- https://developers.google.com/kml/documentation
- https://www.ogc.org/standards/gml
- https://geojson.org
- https://developer.android.com/codelabs/basic-android-kotlin-training-repository-pattern
- https://developer.android.com/topic/libraries/architecture/viewmodel
- https://developer.android.com/reference/org/xmlpull/v1/XmlPullParser
- https://developer.android.com/reference/androidx/recyclerview/widget/DiffUtil
- https://developers.google.com/maps/documentation/android-sdk/utility/marker-clustering