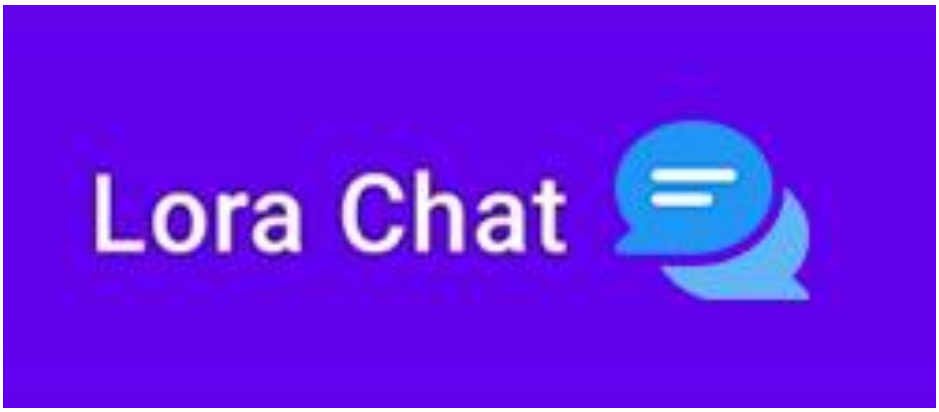
ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ANDROID ΓΙΑ ΕΠΙΚΟΙΝΩΝΙΑ P2P ΜΕΣΩ ΤΟΥ ΠΡΩΤΟΚΟΛΛΟΥ LoRa»



Του φοιτητή
Λοράν Τάσι
Αρ. Μητρώου: 154543

Επιβλέπων Καθηγητής
Χρήστος Ηλιούδης

Θεσσαλονίκη 2022

Τίτλος Δ.Ε. Ανάπτυξη εφαρμογής Android για επικοινωνία p2p μέσω του πρωτοκόλλου LoRa
Κωδικός Δ.Ε.: 21238
Ονοματεπώνυμο φοιτητή: Λοράν Τάσι
Ονοματεπώνυμο εισηγητή: Χρήστος Ηλιούδης
Ημερομηνία ανάληψης Δ.Ε. 01/04/2021
Ημερομηνία περάτωσης Δ.Ε. 01/04/2021

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

«Αφιέρωση»

Πρόλογος

Ένας από τους σημαντικότερους κλάδους της πληροφορικής αποτελεί εκείνος των τηλεπικοινωνιών. Με τη χρήση των τηλεπικοινωνιών οι άνθρωποι επικοινωνούν και ανταλλάσσουν απόψεις σε πραγματικό χρόνο. Οι τηλεπικοινωνίες όμως δεν αφορούν μόνο το ίντερνετ αλλά κάθε μορφή απομακρυσμένης επικοινωνίας. Μέσω αυτής της πτυχιακής εργασίας ήθελα να παρουσιάσω μία άλλη οπτική στο θέμα των τηλεπικοινωνιών, αφαιρώντας το ίντερνετ και κάνοντας χρήση άλλων τεχνολογιών, όπως είναι το πρωτόκολλο LoRa. Η ενασχόληση και το ενδιαφέρον μου για τις ασύρματες επικοινωνίες με ώθησε να αναλάβω αυτή την πτυχιακή εργασία ώστε να εξοικειωθώ περισσότερο και να επεκτείνω τις γνώσεις μου στον προγραμματισμό σε Arduino αλλά και σε Android. Παράλληλα κατανόησα καλύτερα το πρωτόκολλο που χρησιμοποιήθηκε και τις δυνατότητες και τους περιορισμούς του. Κατά την δημιουργία κάθε τμήματος της πτυχιακής λήφθηκαν σημαντικές γνώσεις σχετικά με τεχνολογίες και τον ορθό τρόπο δημιουργίας και οργάνωσης μίας σύνθετης εφαρμογής Android για chatting.

Περίληψη

Στόχος αυτής της πτυχιακής εργασίας είναι η υλοποίηση ενός συστήματος και μίας Android εφαρμογής η οποία θα διευκολύνει την ανταλλαγή μηνυμάτων κειμένου δύο ανθρώπων σε μακρινές αποστάσεις κάνοντας χρήση του πρωτοκόλλου επικοινωνίας μακρινών αποστάσεων LoRa.

Για την υλοποίηση του συστήματος γίνεται χρήση ενός μικροελεγκτή, του NodeMCU ESP8266 και ενός πομποδέκτη, του Reyax RYLR896. Ο μικροελεγκτής συνδέεται με τον πομποδέκτη για την αξιοποίηση των λειτουργιών του που είναι η διαχείρισή του και η αποστολή μηνυμάτων και αποτελεί συνδετικό κρίκο του hardware με το software. Η επικοινωνία μεταξύ του ESP8266 και του RYLR896 είναι σειριακή, ενώ για την επικοινωνία εφαρμογών με τον μικροελεγκτή γίνεται με τη χρήση websockets.

Ο σχεδιασμός του συστήματος έχει γίνει έτσι, ούτως ώστε κάθε μέλος να είναι ανεξάρτητο από το άλλο, με σκοπό την εύκολη επεκτασιμότητα και τροποποίησή του. Το σύστημα σε επίπεδο hardware αποτελείται από το NodeMCU ESP8266 και του RYLR896.

Σε επίπεδο software απαρτίζεται από τρία επίπεδα. Το πρώτο επίπεδο είναι εκείνο που αφορά την επικοινωνία του hardware μεταξύ του αλλά και με άλλες εφαρμογές παρέχοντας ένα API μέσω websockets.Το δεύτερο επίπεδο είναι ένα πρόγραμμα γραμμένο σε Kotlin το οποίο κάνει αξιοποίηση αυτού του API και παρέχει μεθόδους σε υψηλότερο επίπεδο για την χρήση από άλλα προγράμματα ή εφαρμογές. Το τρίτο επίπεδο είναι η Android εφαρμογή.

Η Android εφαρμογή χρησιμοποιεί της μεθόδους του προγράμματος του δευτέρου επιπέδου και μέσα από ένα ωραίο UI δίνει τη δυνατότητα ανταλλαγής μηνυμάτων. Η εφαρμογή εκτός από ανταλλαγή μηνυμάτων, παρέχει και μια σειρά από ρυθμίσεις που αφορούν την ασφάλεια των μηνυμάτων και την απόσταση που αυτά μπορούν να φτάσουν. Στις επόμενες σελίδες παρουσιάζονται όλες οι πληροφορίες που θα χρειαστεί κάποιος για την ουσιαστική κατανόηση της λειτουργίας και της δημιουργίας της εφαρμογής.

«Android application development for p2p communication through the LoRa protocol»

«Loran Tashi»

Abstract

The aim of this thesis is the implementation of a system and an Android application that will facilitate the exchange of text messages of two people at long distances using the LoRa long-distance communication protocol. For the implementation of the system, a microcontroller is used, the NodeMCU ESP8266 and a transceiver, the Reyax RYLR896. The microcontroller is connected to the transceiver to use its functions which are its management and sending messages and is a link between hardware and software. The communication between ESP8266 and RYLR896 is serial, while for the communication of applications with the microcontroller is done using websockets. The design of the system has been done in such a way that each member is independent from the other, aiming at its easy scalability and modification. The hardware-level system consists of the NodeMCU ESP8266 and the RYLR896. At the software level it consists of three levels. The first level is the one that concerns the communication of hardware with each other but also with other applications providing an API through websockets.The second level is a program written in Kotlin which makes use of this API and provides methods at a higher level for use by other programs or applications. The third level is the Android app. The Android application uses the methods of the program of the second level and through a nice and useful UI gives the possibility of messaging. In addition to messaging, the application also provides a number of settings regarding the security of messages and the distance they can reach. The following pages present all the information that someone will need for a meaningful understanding of the operation and creation of the application.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία

Κεφάλαιο 1ο: Εισαγωγή

1.1 Λίγα Λόγια για την Εφαρμογή

Το LoRa Chat είναι μία εφαρμογή γραμμένη εξ' ολοκλήρου σε Kotlin όσο αφορά τον client και Arduino στον server. Αποτελείται από τρία κύρια μέρη, τα οποία είναι ο websocket server, που αποτελεί τον συνδετικό κρίκο ανάμεσα στη συσκευή που είναι συνδεδεμένη μέσω Wi-Fi με το ESP8266, σε αυτή την περίπτωση είναι ένα κινητό Android, ένα Kotlin module το οποίο αναλαμβάνει τον ρόλο του websocket client και παρέχει την λειτουργικότητα για την επικοινωνία της εφαρμογής με τον websocket server και κατ' επέκταση με τον πομποδέκτη RYLR896 και τέλος η Android εφαρμογή.

Ο πομποδέκτης RYLR896 αναλαμβάνει την μεταβίβαση των μηνυμάτων που στέλνει η εφαρμογή μέσω websocket, σε μακρινές αποστάσεις χρησιμοποιώντας το πρωτόκολλο LoRa.

Το Kotlin module είναι ανεξάρτητο που σημαίνει ότι μπορεί να χρησιμοποιηθεί σαν συνδετικός κρίκος σε οποιοδήποτε μηχάνημα μπορεί να τρέξει την εικονική μηχανή της Java (JVM) και όχι απαραίτητα μόνο στην εφαρμογή LoRa Chat.

Η Android εφαρμογή με λίγα λόγια από την πλευρά της λειτουργικότητας περιέχει τις διαδικασίες για την αλληλεπίδραση του χρήστη με το module ώστε να στέλνει μηνύματα, να λαμβάνει και να αλλάζει ορισμένες ρυθμίσεις και ταυτόχρονα έχει ένα εύχρηστο User Interface.

Στο δεύτερο κεφάλαιο γίνεται μία προσπάθεια να παρουσιαστούν οι βασικές τεχνολογίες που χρησιμοποιήθηκαν, όπως η γλώσσα προγραμματισμού Kotlin και οι διαφορές της με την Java. Ακόμα, αναφέρεται γιατί προτιμήθηκε ο μικροελεγκτής ESP8266 και εξετάζονται τα τεχνικά χαρακτηριστικά του. Στη συνέχεια γίνεται μία περιγραφή του πρωτοκόλλου LoRa και της τεχνικής διαμόρφωσης σήματος που χρησιμοποιεί για να επιτυγχάνει την επικοινωνία σε μακρινές αποστάσεις. Κατόπιν γίνεται αναφορά στα τεχνικά χαρακτηριστικά του RYLR896 και στα AT commands που χρησιμοποιούνται για την επικοινωνία του με άλλες συσκευές. Τέλος γίνεται μία αναφορά στο Android και το Android stack τα οποία ύστερα παρουσιάζονται αναλυτικά.

Στο τρίτο κεφάλαιο θα αναφερθούν οι τεχνολογίες και οι βιβλιοθήκες που χρησιμοποιήθηκαν για την υλοποίηση της εργασίας.

Στο τέταρτο κεφάλαιο θα γίνει εκτενής περιγραφή του τρόπου υλοποίησης του κώδικα. Αρχικά περιγράφεται ο κώδικας Arduino και τις βιβλιοθήκες που συμπεριλήφθηκαν. Έπειτα γίνεται αναφορά για τον τρόπο αναπαράστασης των μηνυμάτων που αφορούν την βιβλιοθήκη σε κλάσεις και τον τύπο των μηνυμάτων. Ακολουθεί ο τρόπος διαχείρισης των εισερχόμενων μηνυμάτων σε επίπεδο AT commands και μετατροπής τους στους κατάλληλους τύπους κλάσεων. Τέλος ακολουθεί η διαδικασία της υλοποίησης της εφαρμογής Android όσον αφορά τα Activities της εφαρμογής και τις λειτουργίες τους.

Στο πέμπτο κεφάλαιο δίνονται οδηγίες βήμα προς βήμα για την χρήση της εφαρμογής με μορφή screenshots, τα λάθη που μπορεί να προκύψουν και αναλύονται οι ρυθμίσεις που μπορεί να αλλάξει ο χρήστης από την οθόνη ρυθμίσεων.

Στο έκτο και τελευταίο κεφάλαιο υπάρχουν τα συμπεράσματα που προέκυψαν κατά τη δημιουργία της εργασίας και προτάσεις βελτίωσης σε κάποια επόμενη έκδοση.

Κεφάλαιο 2ο: Kotlin, Arduino, ESP8266, Πρωτόκολλο LoRa, RYLR896, AT Commands, Android

2.1 Kotlin

2.1.1 Εισαγωγή

Η Kotlin δημιουργήθηκε από την ομάδα JetBrains από την Αγία Πετρούπολη της Ρωσίας, η οποία έχει δημιουργήσει και το IntelliJ IDEA. Το IntelliJ IDEA αποτελεί τη βάση για το Android Studio. Ο κύριος σχεδιαστής γλώσσας είναι ο Roman Elizarov. Το έργο ξεκίνησε το 2010 και ήταν ανοιχτού κώδικα από πολύ νωρίς. Η πρώτη επίσημη κυκλοφορία 1.0 ήταν τον Φεβρουάριο του 2016. Η Kotlin χρησιμοποιείται πλέον σε μεγάλη ποικιλία περιβαλλόντων σε πολλαπλά λειτουργικά συστήματα. Έχουν περάσει σχεδόν πέντε χρόνια από τότε που η Google ανακοίνωσε την υποστήριξη του Kotlin στο Android. Σύμφωνα με το Android Developers Blog, από το 2021, πάνω από 1,2 εκατομμύρια εφαρμογές στο Google Play store χρησιμοποιούν το Kotlin, συμπεριλαμβανομένου του 80% από τις χίλιες κορυφαίες εφαρμογές.

2.1.2 Κύριε Εκδόσεις της Kotlin

Οι κύριες εκδόσεις της Kotlin παρουσιάζονται στον παρακάτω πίνακα:

Πίνακας 2.1 Κύριες Εκδόσεις Kotlin

Αριθμός έκδοσης	**Ημερομηνία**
1.1	15 February 2016
1.2	28 November 2017
1.3	29 October 2018
1.4	17 August 2020
1.5	5 May 2021

2.1.3 Kotlin και Java

Η Kotlin είναι 100% διαλειτουργική με τη Java και έχει δοθεί μεγάλη έμφαση στη διασφάλιση ότι η υπάρχουσα βάση κωδίκων μπορεί να αλληλεπιδράσει σωστά με την Kotlin.

Δίνεται η δυνατότητα:

- Αντικειμενοστρεφούς προγραμματισμού
- Functional προγραμματισμού
- Συνδυασμού και των δύο.

Μπορούμε με ευκολία να καλέσουμε κώδικα γραμμένο σε Kotlin από Java και κώδικα Java από Kotlin. Υπάρχει επίσης ένας αυτοματοποιημένος μετατροπέας Java-to-Kotlin ενσωματωμένος στο IDE που απλοποιεί τη μετατροπή του υπάρχοντος κώδικα.

Το όνομα της γλώσσας προκύπτει από ένα νησί κοντά στην Αγία Πετρούπολη, ενώ το όνομα Java προκύπτει από νησί στην Ινδονησία .

Αν και η Kotlin υποστηρίζει άλλες πλατφόρμες (iOS, WebAssembly, Kotlin/JS, κ.λπ.), ένα κλειδί για την ευρεία χρήση της Kotlin είναι η υποστήριξή της για την εικονική μηχανή Java (JVM).

Εφόσον η Kotlin μπορεί να μεταγλωττιστεί σε Java bytecode, μπορεί να τρέξει οπουδήποτε εκτελείται ένα JVM.

Η Kotlin είναι μια νέα και διαφορετική γλώσσα με συνδέσεις με Scala, Swift και C# που είναι σχεδόν τόσο ισχυρές όσο η σύνδεσή της με την Java. Έχει τα δικά της χαρακτηριστικά και τους δικούς της ιδιωματισμούς.

2.1.4 Διαφορές της Kotlin με τη Java

Τι περιλαμβάνει η Java που δεν έχει η Kotlin

- Checked exceptions
- Οι πρωταρχικοί τύποι δεν είναι κλάσεις. Το byte-code χρησιμοποιεί πρωταρχικούς τύπους όπου είναι δυνατόν.
- Τα στατικά μέλη αντικαθιστώνται από τα companion objects, τις top-level functions, τις extension functions, ή το annotation @JvmStatic.
- Ο τριαδικός τελεστής a ? b : c αντικαθιστάται με if expression στην Kotlin.

Τι περιλαμβάνει η Kotlin που δεν έχει η Java

- Lambda expressions
- Inline functions
- Extension functions
- Το null-safety σύστημα της Kotlin στοχεύει στην εξαφάνιση του κινδύνου των null references, γνωστών ως The Billion Dollar Mistake
- Smart casts
- String templates
- Operator overloading
- Companion objects
- Data classes
- Ξεχωριστά interfaces για read-only και mutable collections
- Coroutines

2.2 Arduino

2.2.1 Εισαγωγή

Το Arduino είναι μία open source ηλεκτρονική πλατφόρμα βασισμένη σε εύχρηστο υλικό και λογισμικό. Οι πλακέτες Arduino μπορούν να διαβάζουν εισόδους, όπως το πάτημα ενός κουμπιού, το φως σε έναν αισθητήρα και να το μετατρέψουν σε έξοδο, ενεργοποιώντας έναν κινητήρα, να ανάψουν ένα LED και πολλά άλλα. Μπορούμε να πούμε στην πλακέτα τι να κάνει στέλνοντας μια σειρά οδηγιών στον μικροελεγκτή στην πλακέτα.

Για να το καταφέρουμε αυτό, χρησιμοποιούμε την γλώσσα προγραμματισμού Arduino και το λογισμικό Arduino IDE.

2.2.2 Λόγοι Προτίμησης Arduino ή Arduino Compatible Boards

Χάρη στην απλή και προσιτή εμπειρία χρήσης, το Arduino έχει χρησιμοποιηθεί από χιλιάδες διαφορετικά πρότζεκτ και εφαρμογές.

Το λογισμικό πέρα από το γεγονός ότι είναι εύκολο στη χρήση για αρχάριους, είναι αρκετά ευέλικτο για έμπειρους χρήστες. Μπορεί να τρέξει σε Mac, Windows και Linux.

Χρησιμοποιείται για να δημιουργηθούν πρότζεκτ χαμηλού κόστους από παιδιά, χομπίστες, προγραμματιστές και έχει μεγάλη κοινότητα και μπορείς να βρεις λύση σε κάθε πρόβλημα ρωτώντας άλλους χρήστες.

Τα κύρια χαρακτηριστικά γιατί να προτιμήσεις ένα Arduino ή Arduino compatible board αναφέρονται στη λίστα παρακάτω:

- **Φθηνό** – Τα Arduino boards είναι χαμηλού κόστους
- **Cross-Platform** – Το Arduino IDE τρέχει σε Windows, Macintosh OSX και Linux λειτουργικά συστήματα.
- **Απλή και κατανοητή εμπειρία προγραμματισμού** – Το Arduino IDE είναι εύχρηστο και αρχάριους αλλά και ευέλικτο για πιο έμπειρους.
- **Ανοιχτού κώδικα και επεκτάσιμο λογισμικό** – Ο κώδικας για το Arduino είναι open source και διαθέσιμο για επέκταση από έμπειρους προγραμματιστές. Η γλώσσα μπορεί να επεκταθεί μέσω βιβλιοθηκών C++.Παρουσίαση Δ.Ε.

2.3 ESP8266

2.3.1 Εισαγωγή

Το ESP8266 είναι ένα System on a Chip (SoC), που κατασκευάζεται από την κινεζική εταιρεία Espressif.

Αποτελείται από μια μονάδα μικροελεγκτή 32-bit Tensilica L106 (MCU) και έναν πομποδέκτη Wi-Fi. Διαθέτει 11 GPIO pins (General Purpose Input/Output pins), καθώς και μια αναλογική είσοδο.

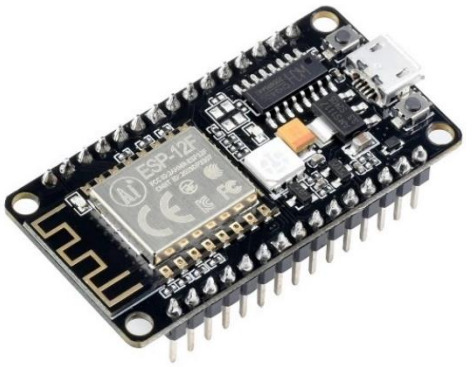
Μπορεί να λειτουργήσει σε θερμοκρασίες της τάξεως των -40°C έως τους 125°C.

Το ESP8266 μπορεί να προγραμματιστεί μεταξύ άλλων και με τη χρήση του Arduino IDE και να χρησιμοποιηθεί σαν σημείο πρόσβασης (Access Point) και να συνδέονται συσκευές ασύρματα σε αυτό ή σαν πελάτης (Station) και να συνδέεται εκείνο σε κάποιο Access Point το οποίο ενδεχομένως μπορεί να είναι και συνδεδεμένο και στο ίντερνετ. Επομένως μπορεί να συνδεθεί στο ίντερνετ, να φιλοξενήσει έναν απλό web server, να συνδεθεί το κινητό ή ο υπολογιστής σε αυτό και άλλα.

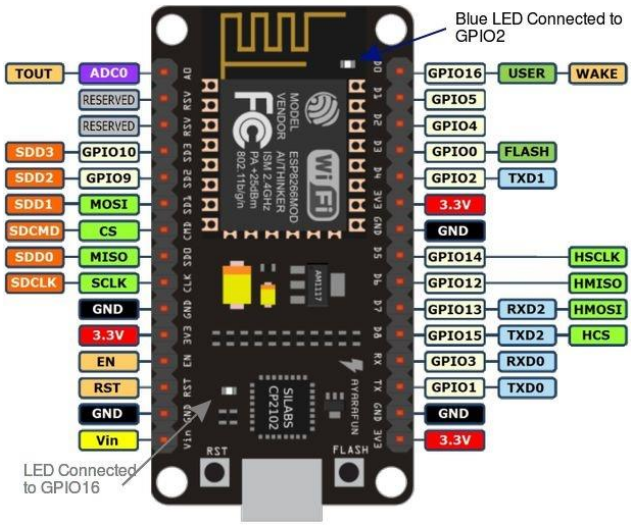
Το ESP8266 είναι σχετικά μικρό για αρχάριους να ξεκινήσουν να πειραματίζονται με αυτό. Για τον λόγο αυτό έχουν δημιουργηθεί development boards ώστε να μπορεί να συνδεθούν εύκολα σε breadboard και να καθίσταται δυνατός ο προγραμματισμός και η τροφοδοσία μέσω θύρας USB.

Το πιο γνωστό και αυτό που χρησιμοποιείται ευρέως είναι το NodeMCU.

Πρόσφατα η εταιρεία Espressif κυκλοφόρησε το ESP32, έναν περισσότερο ισχυρό χαμηλής κατανάλωσης μικροελεγκτή με διπύρηνο επεξεργαστή με περισσότερες δυνατότητες καθώς και υποστήριξη Bluetooth εκτός από Wi-Fi.



Εικόνα 2.1 NodeMCU ESP8266



Εικόνα 2.2 NodeMCU GPIOs – Πηγή randomnerdtutorials

2.3.2 Προδιαγραφές ESP8266

Πίνακας 2.2 Προδιαγραφές ESP8266

Τάση	3.3V
Κατανάλωση	10uA – 170mA
Μνήμη Flash	16Mb μέγιστη (512K σύνηθες)
Επεξεργαστής	Tensilica L106 32-bit
Ταχύτητα επεξεργαστή	80-160MHz
Μνήμη RAM	32K + 80K

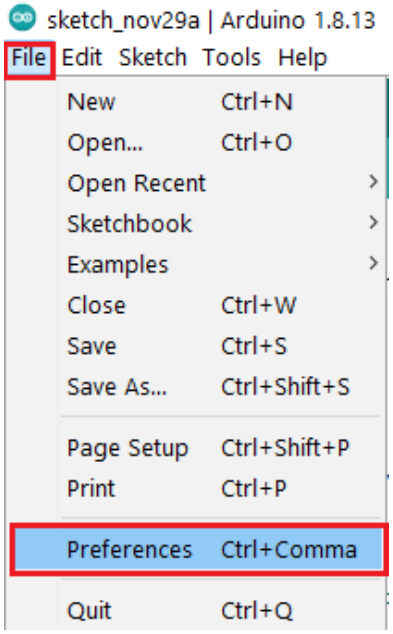
GPIO pins	17
802.11	b/g/n/d/e/i/k/r
Μέγιστος αριθμός TCP συνδέσεων	5

2.3.3 Οδηγίες Εγκατάστασης του ESP8266 στο Arduino IDE

Το Arduino IDE υποστηρίζει το ESP8266 αλλά πρώτα πρέπει να γίνει μία απλή διαδικασία εγκατάστασής του.

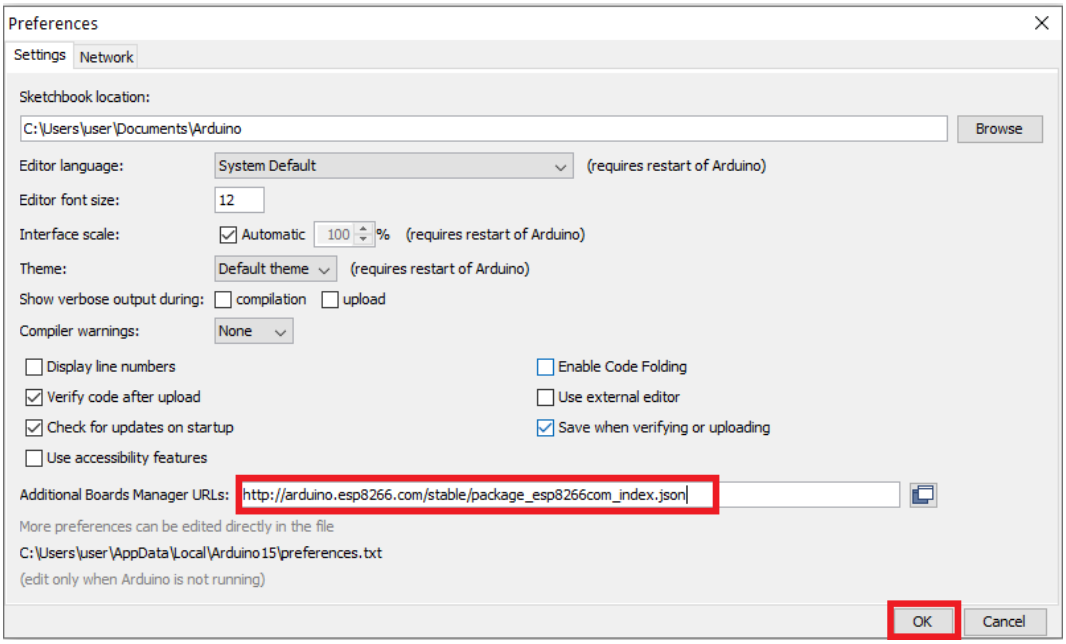
Παρακάτω ακολουθούν αναλυτικές οδηγίες εγκατάστασης και εικόνες για το κάθε βήμα που πρέπει να ακολουθήσουμε:

1. Στο πρώτο βήμα πρέπει να μεταβούμε στη μπάρα μενού, να κάνουμε κλικ στο File και έπειτα να πατήσουμε στο Preferences.



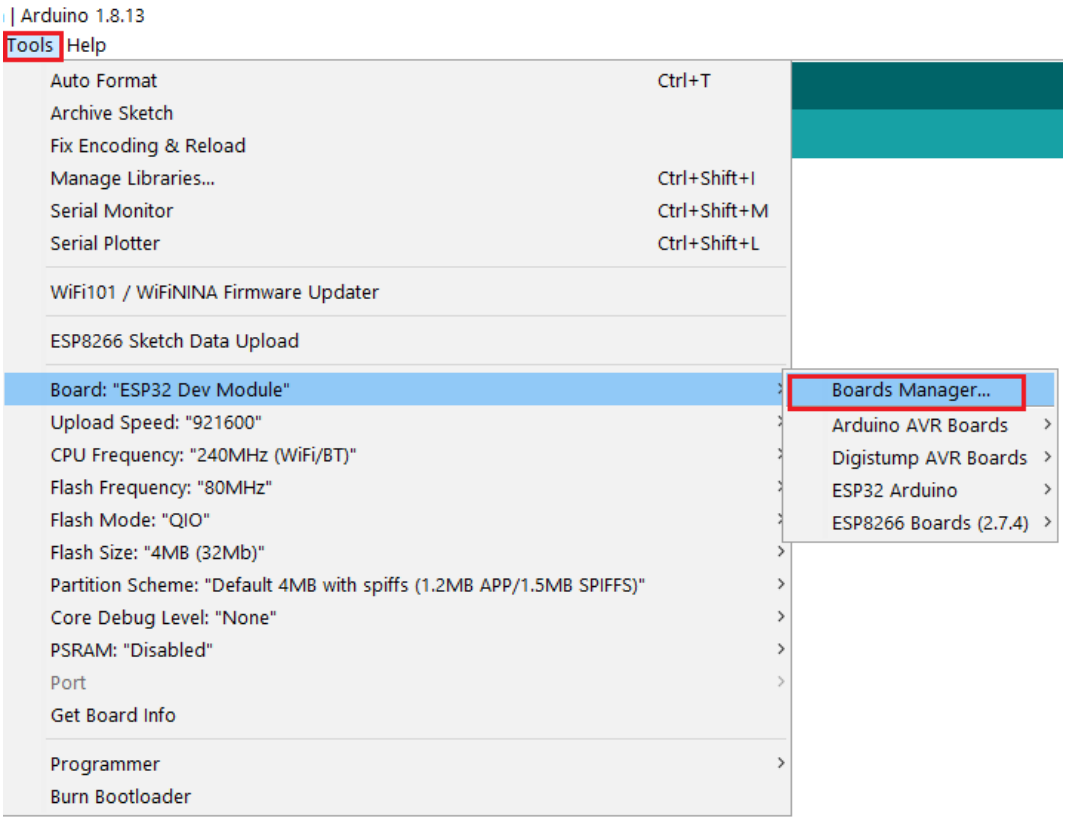
Εικόνα 2.3 Οδηγίες Εγκατάστασης του ESP8266 στο Arduino IDE 1

2 Στο δεύτερο βήμα πρέπει να βάλουμε τον σύνδεσμο «http://arduino.esp8266.com/stable/package_esp8266com_index.json» στο σημείο που γράφει Additional Board Manager URLs. Για τυχόν προσθήκη επιπλέον board, οι σύνδεσμοι χωρίζονται με κόμμα



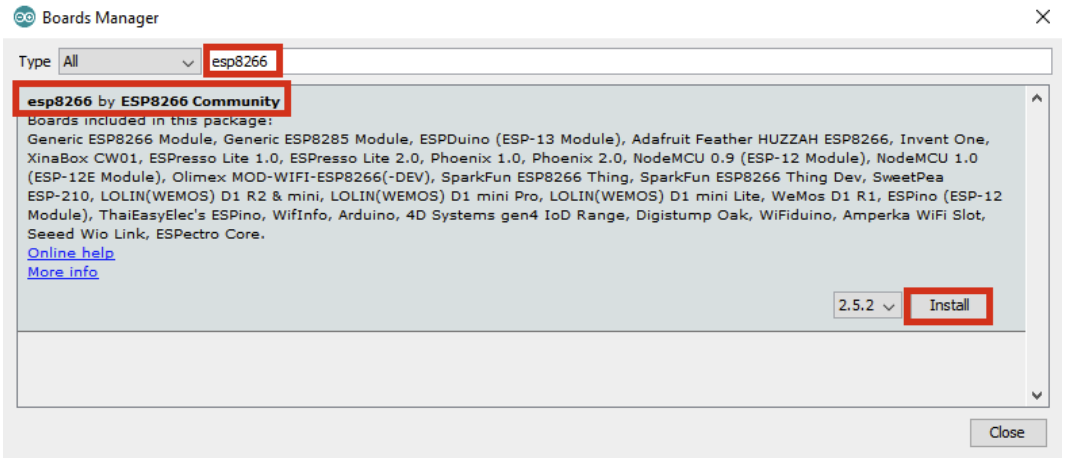
Εικόνα 2.4 Οδηγίες Εγκατάστασης του ESP8266 στο Arduino IDE 2

3 Στο τρίτο βήμα πρέπει να μεταβούμε στο μενού Tools και να επιλέξουμε το Boards Manager.



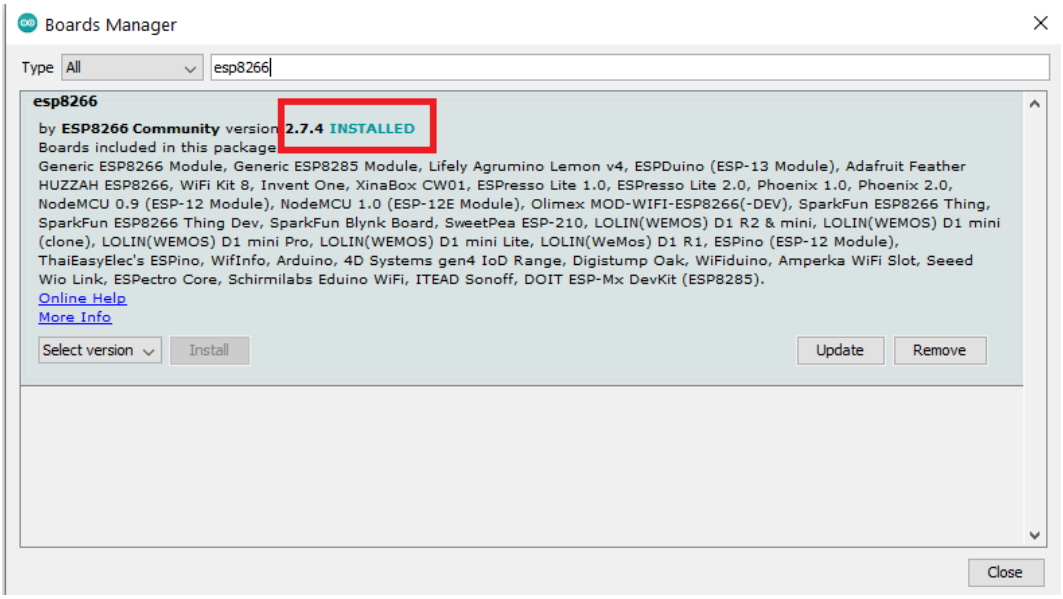
Εικόνα 2.5 Οδηγίες Εγκατάστασης του ESP8266 στο Arduino IDE 3

4 Στο τέταρτο βήμα αναζητούμε «esp8266» στην μπάρα αναζήτησης και πατάμε Install.



Εικόνα 2.6 Οδηγίες Εγκατάστασης του ESP8266 στο Arduino IDE 4

5 Στο τελευταίο βήμα ελέγχουμε ότι έγινε η εγκατάσταση.

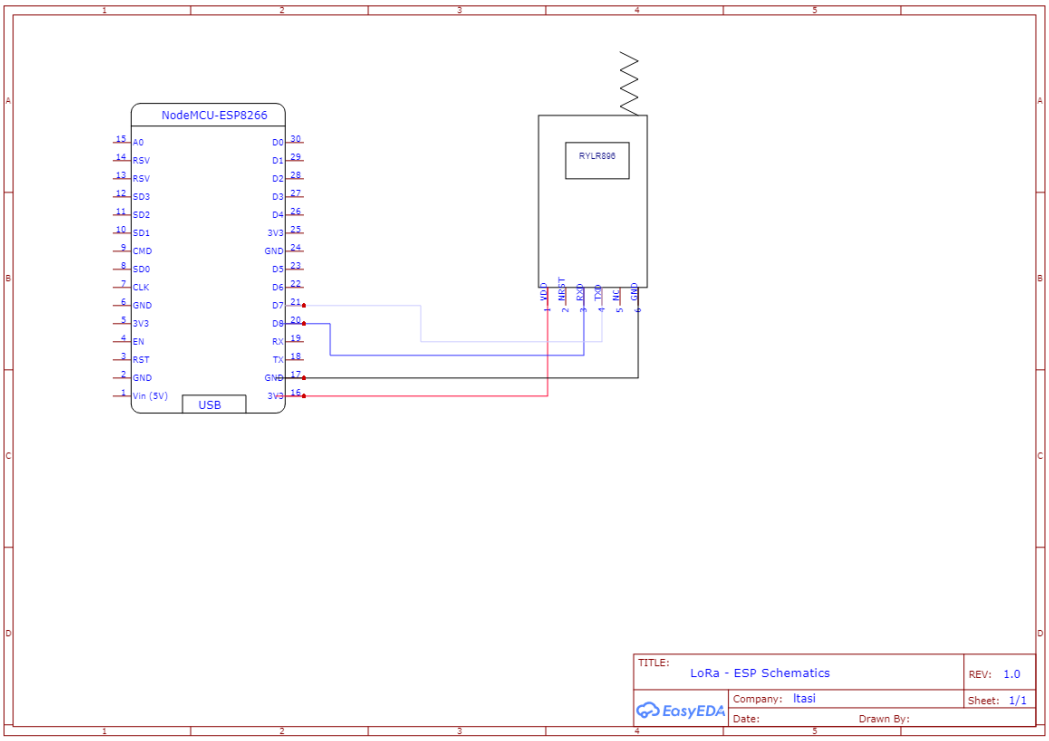


Εικόνα 2.7 Οδηγίες Εγκατάστασης του ESP8266 στο Arduino IDE 5

2.3.4 Λόγοι Επιλογής NodeMCU ESP8266 Developmend Board

Η επιλογή του NodeMCU ESP8266 δεν ήταν τυχαία.

Ο πιο εύκολος τρόπος για να επικοινωνήσεις με ένα κινητό Android για τέτοιες εφαρμογές είναι το Wi-Fi. Το NodeMCU ESP8266 έχει το κατάλληλο μέγεθος καθώς και η τροφοδοσία του γίνεται και μέσω micro USB καθιστώντας την εύκολη, αφού μπορεί να χρησιμοποιηθεί ακόμα και ένα powerbank.



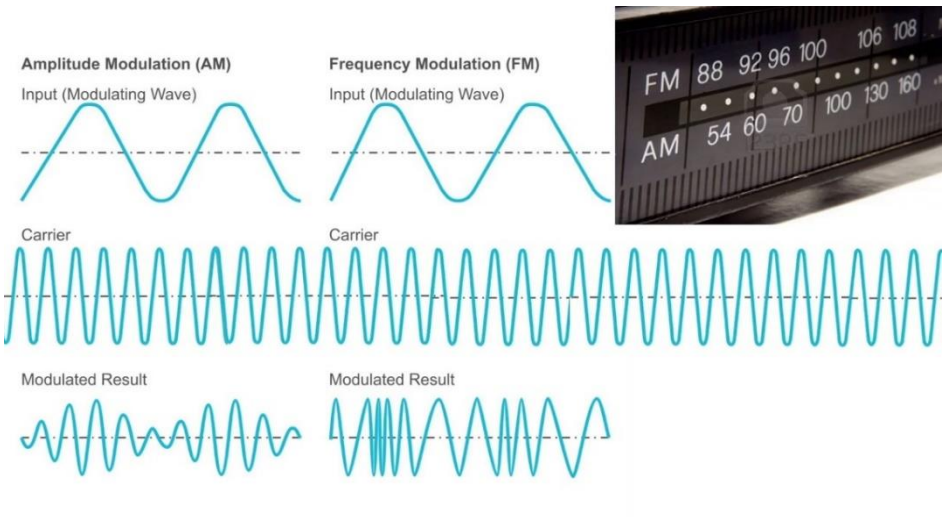
Εικόνα 2.8 Schematics

2.4 Πρωτόκολλο LoRa

2.4.1 Εισαγωγή

Ένα σκέτο ηλεκτρομαγνητικό σήμα δεν περιέχει κάποια πληροφορία και για αυτό θα πρέπει να μεταβληθεί για να μεταφέρει πληροφορίες.

Δύο τρόποι με τους οποίους είμαστε εξοικειωμένοι και χρησιμοποιούνται για τα ραδιόφωνα είναι το FM που σημαίνει διαμόρφωση συχνότητας και ο άλλος το AM που σημαίνει διαμόρφωση πλάτους.



Εικόνα 2.9 Επεξήγηση του thethingsnetwork – Πηγή thethingsnetwork

Στην ψηφιακή μετάδοση σήματος, στο FM η μία συχνότητα μπορεί να χρησιμοποιηθεί για να συμβολίζει το 0 και η άλλη το 1. Αυτή η μέθοδος είναι συνηθισμένη και ονομάζεται Frequency Shift Keying.

2.4.2 Παραμετροποίηση της Διαμόρφωσης LoRa

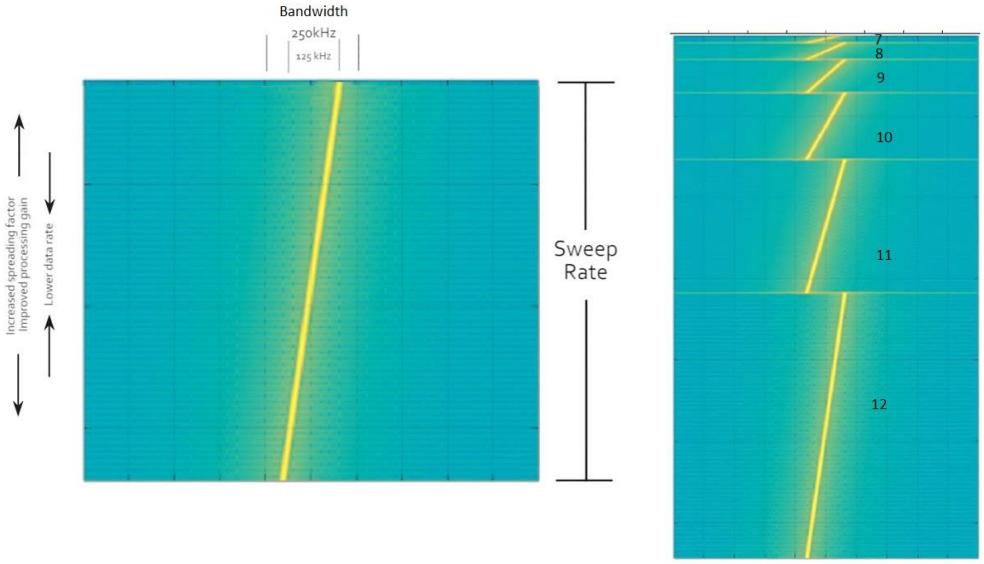
Υπάρχουν διάφορες παράμετροι για την παραμετροποίηση της διαμόρφωσης LoRa.

Δύο από αυτές είναι το Bandwidth και το Spreading Factor (SF).

Η αύξηση του Bandwidth δίνει σήμα καλύτερης ποιότητας, όμως το Bandwidth δεν είναι απεριόριστο. Το Spreading Factor, το οποίο παίρνει τιμές από 7 - 12 ελέγχει τον ρυθμό των chirp, επομένως ελέγχει και την ταχύτητα της μετάδοσης δεδομένων. Μικρό spreading factor σημαίνει πιο γρήγορα chirps με αποτέλεσμα μια πιο γρήγορη μετάδοση δεδομένων. Για κάθε αύξηση του spreading factor, το chirp sweep rate (ρυθμός σάρωσης) μειώνεται στο μισό και έτσι μειώνεται στο μισό και ο ρυθμός μετάδοσης.

Ένας αργός ρυθμός σάρωσης (sweep rate) είναι εύκολος να αποκωδικοποιηθεί και όσο πιο γρήγορος γίνεται, μπορούμε να μεταφέρουμε περισσότερη πληροφορία αλλά δυσκολεύει η αποκωδικοποίηση του σήματος.

Παρακάτω μπορεί να γίνει πιο κατανοητό πως μοιάζει η μετάδοση.



Εικόνα 2.10 LoRa Sweep Rate και Spreading Factors – Πηγή thethingsnetwork

Πίνακας 2.3 Ευαισθησία Λήψης του Semtech SX1276 μετρημένη σε dBm

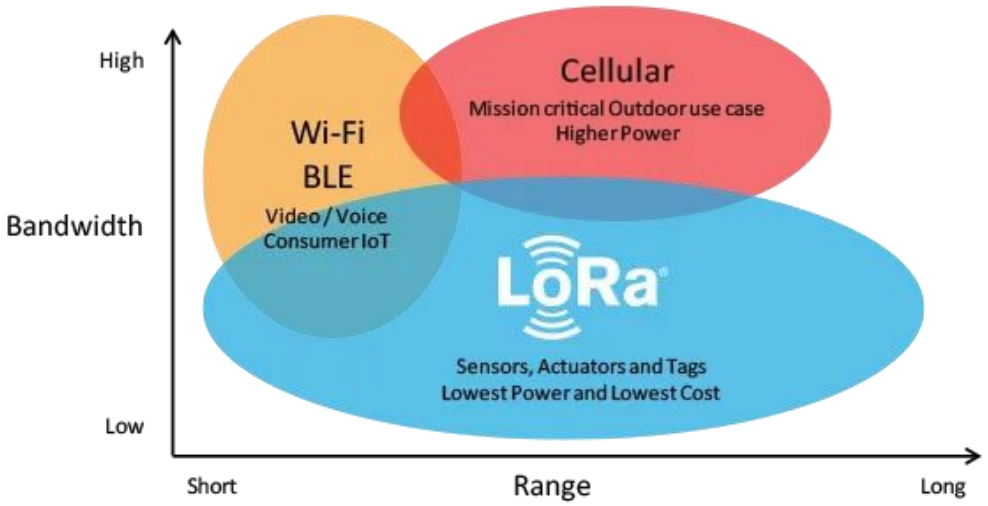
SF / BW	7	8	9	10	11	12
125 KHz	-123	-126	-129	-132	-133	-136
250 KHz	-120	-123	-125	-128	-130	-133

500KHz	-116	-119	-122	-125	-128	-130

Το LoRa είναι μια τεχνική ασύρματης διαμόρφωσης που προέρχεται από την τεχνολογία Chirp Spread Spectrum (CSS) όπου τα chirp (γνωστά και ως σύμβολα) είναι ο φορέας της πληροφορίας.

Κωδικοποιεί πληροφορίες για ραδιοκύματα χρησιμοποιώντας παλμούς chirp - παρόμοιο με τον τρόπο που επικοινωνούν τα δελφίνια και οι νυχτερίδες!

Η διαμορφωμένη μετάδοση LoRa είναι ανθεκτική έναντι σε παρεμβολές και μπορεί να ληφθεί σε μεγάλες αποστάσεις. Η διαμόρφωση LoRa είναι μια ιδιόκτητη τεχνολογία της εταιρείας Semtech και ως εκ τούτου δεν είναι πλήρως ανοιχτή. Η διαμόρφωση LoRa χρησιμοποιεί μια τροποποιημένη διαμόρφωση συχνότητας (Frequency Shift Keying) που επιτρέπει τη μεταφορά δεδομένων σε εμβέλεια δεκάδων χιλιομέτρων.



Εικόνα 2.11 Συγκριση πρωτοκόλλου LoRa με άλλες τεχνολογίες – Πηγή TheThingsNetwork

Το LoRa είναι ιδανικό για εφαρμογές που μεταδίδουν μικρά κομμάτια δεδομένων με χαμηλούς ρυθμούς bit. Τα δεδομένα μπορούν να μεταδοθούν σε μεγαλύτερη εμβέλεια σε σύγκριση με τεχνολογίες όπως WiFi, Bluetooth ή ZigBee.

Αυτά τα χαρακτηριστικά καθιστούν το LoRa κατάλληλο για αισθητήρες που λειτουργούν σε λειτουργία χαμηλής ισχύος. Το LoRa μπορεί να λειτουργήσει σε ζώνες χωρίς άδεια sub-gigahertz, για παράδειγμα, 915 MHz, 868 MHz και 433 MHz. Στην Ευρώπη επιτρέπεται η χρήση του EU-863-870 ενώ σε μερικές χώρες και η EU-433. Αυτές οι συχνότητες εμπίπτουν σε ζώνες ISM (Industrial, Scientific and Medical) που είναι δεσμευμένες διεθνώς για βιομηχανικούς, επιστημονικούς και ιατρικούς σκοπούς και δεν υπάρχουν επιπρόσθετες χρεώσεις για απόκτηση άδειας.

Μπορεί επίσης να λειτουργήσει στα 2,4 GHz για να επιτύχει υψηλότερους ρυθμούς μετάδοσης δεδομένων σε σύγκριση με τις ζώνες sub-gigahertz, με αντάλλαγμα όμως τη μείωση της εμβέλειας.

2.5 RYLR896

2.5.1 Τι Είναι το RYLR896

Η μονάδα πομποδέκτη RYLR896 διαθέτει το μόντεμ μεγάλης εμβέλειας LoRa που παρέχει επικοινωνία ευρέος φάσματος εξαιρετικά μεγάλης εμβέλειας και υψηλή θωράκιση ενάντια στις παρεμβολές, ενώ ελαχιστοποιεί την κατανάλωση ενέργειας. Το RYLR896 είναι πιστοποιημένο από NCC και FCC.

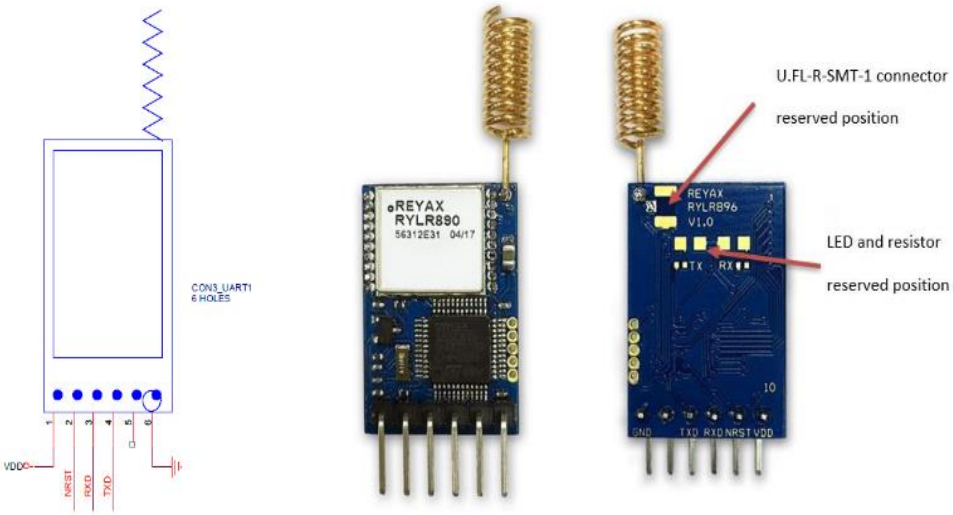
2.5.2 Χαρακτηριστικά του RYLR896

• Semtech SX1276 Engine
• Εξαιρετική θωράκιση ενάντια στις παρεμβολές
• Χαμηλή κατανάλωση
• Υψηλή ευαισθησία
• Εύκολη επικοινωνία με AT commands
• 127 dB Dynamic Range RSSI
• Σχεδιασμένο με ενσωματωμένη κεραία
• AES128 κρυπτογράφηση δεδομένων

2.5.3 Χρησιμότητα του RYLR896

• IoT εφαρμογές
• Φορητός εξοπλισμός
• Ασφάλεια σπιτιού
• Βιομηχανικός εξοπλισμός
• Συναγερμός αυτοκινήτου

2.5.4 Περιγραφή PIN του RYLR896



Εικόνα 2.12 PIN του RYLR896 – Πηγή Reyax Documentation

Πίνακας 2.4 Reyax RYLR896 pins

Pin	Name	I/O	Condition
1	VDD	I	Power Supply
2	NRST	I	RESET(Active Low)100KΩ Internal pull up, Pull down at least 100ms
3	RXD	I	UART Data Input
4	TXD	O	UART Data Output
5	NC	-	
6	GND		Ground (γείωση)

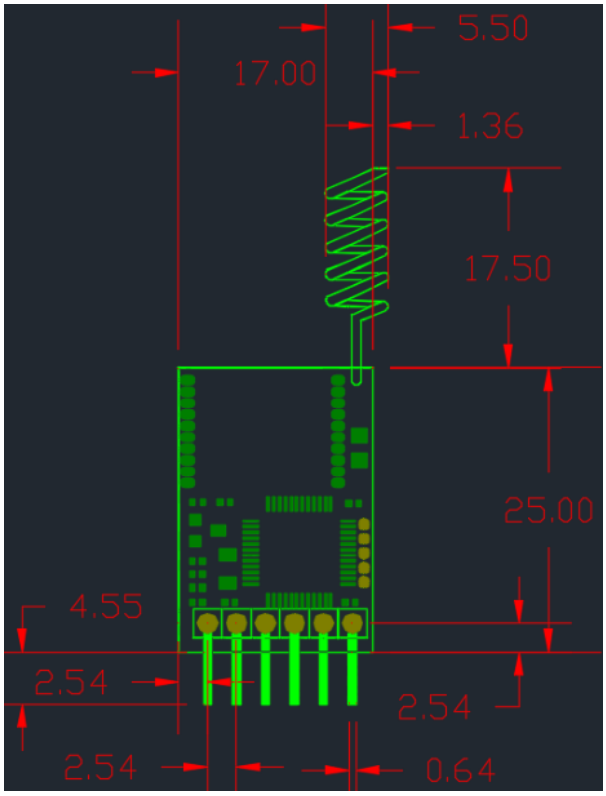
Προδιαγραφές:

Πίνακας 2.5 Reyax RYLR896 Προδιαγραφές

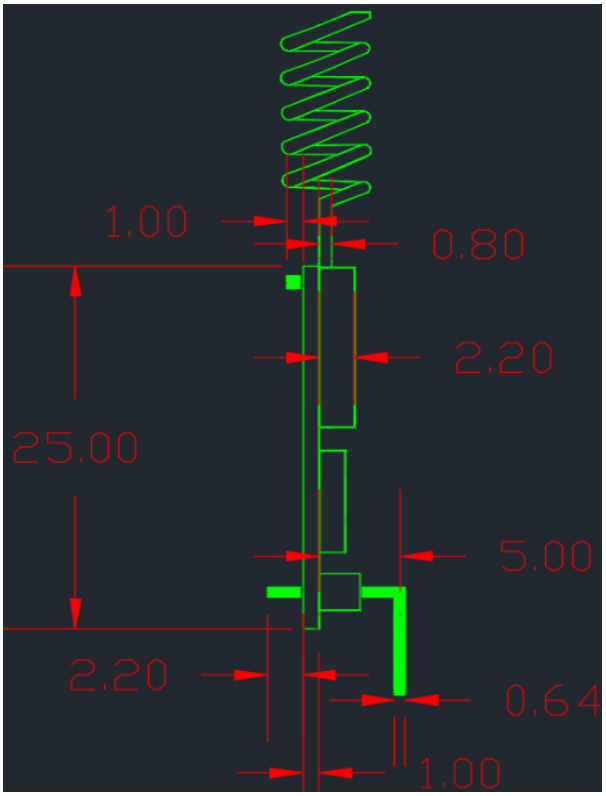
Item	Min.	Typical	Max.	Unit	Condition
VDD Power Supply	2	3.3	3.6	V	VDD
RF Output Power Range	-4		15	dBm	
Filter insertion loss	1	2	3	dB	
RF Sensitivity	-148			dBm	
RF Input Level			10	dBm	
Frequency Range	862	868/915	1020	MHz	
Frequency Accuracy		± 2		ppm	
Communication Range		4.5	15	km	Εξαρτάται από τις παραμέτρους
Transmit current		43		mA	RFOP = +15dBm
Receive current		16.5		mA	AT+MODE=0
Sleep Current		0.5		uA	AT+MODE=1
Baud rate	300	115200	115200	Bps	8, N, 1
Data Input Level High	0.7*VDD		VDD	V	VIH
Data Input Level Low	0		0.3*VDD	V	VIL
Digital Output Level High	0.9		VDD	V	VOH

Data Output Level Low			0.1	V	VOL
Cycling (erase/write) EEPROM data memory		300		K	Cycles
Weight		7		G	
Operating Temperature	-40	25	+85	℃	

Διαστάσεις:



Εικόνα 2.14 Εμπρόσθια Όψη του RYLR896, σε mm



Εικόνα 2.13 Πλάγια Όψη του RYLR896, διαστάσεις σε mm Πηγή Reyax Docs

2.6 AT Commands

2.6.1 Εισαγωγή

Τα AT commands είναι ένα σύνολο από εντολές που χρησιμοποιούνται για την διαχείριση ενός modem, ενώ το AT είναι συντομογραφία για τη λέξη “attention” – προσοχή.

Πήραν το όνομά τους από το γεγονός ότι η σύνταξη κάθε εντολής ξεκινάει με AT.

2.6.2 Χρησιμότητα των AT Commands

Τα AT commands χρησιμοποιούνται για να διαχειριζόμαστε modems.

Χρησιμοποιούνται από την εποχή των dial-up modems ενώ σήμερα ακόμα χρησιμοποιούνται σε 5G modems για την εδραίωση επικοινωνίας, καθώς και στα όλο ένα και πιο ανερχόμενα και δημοφιλή LoRa modules. Βεβαίως, τα AT commands χρησιμεύουν και στο debugging καθώς με αυτές τις εντολές μπορούμε να λάβουμε σημαντικές πληροφορίες.

2.6.3 Σύνταξη των AT Commands στο Reyax RYLR896

Το LoRa module RYLR896 παρέχει μια πληθώρα από AT commands.

Αυτές οι εντολές χωρίζονται σε δύο κατηγορίες:

- Εκείνες που θέτουν μια τιμή
- και εκείνες που διαβάζουν την τρέχουσα τιμή μίας ρύθμισης.

Η μορφή των εντολών για να θέσουμε μία τιμή είναι της μορφής AT+ΠΑΡΑΜΕΤΡΟΣ=ΤΙΜΗ, ενώ για να διαβάσουμε μία τιμή κάποιας ρύθμισης είναι της μορφής AT+ΠΑΡΑΜΕΤΡΟΣ?

Όλες οι εντολές πρέπει να έχουν στο τέλος “enter” ή “\r\n”, όπου \r σημαίνει carriage return ενώ το \n σημαίνει new line. Για να στείλουμε καινούργια εντολή θα πρέπει να περιμένουμε την απάντηση +OK και μετά να συνεχίσουμε.

Στους παρακάτω πίνακες παρουσιάζονται μερικές βασικές εντολές με παραδείγματα για το τι μπορούμε να λάβουμε ως απάντηση:

1. AT – δοκιμή για το αν το module μπορεί να απαντήσει στις εντολές

Πίνακας 2.6 Περιγραφή Χρήσης Εντολής AT

Σύνταξη εντολής	Απάντηση
AT	+OK

2. Software RESET

Πίνακας 2.7 Περιγραφή Χρήσης Εντολής AT + RESET

Σύνταξη εντολής	Απάντηση
AT+RESET	+RESET +READY

3. AT+PARAMETER – Θέτω τις παραμέτρους RF (Radio Frequency)

Πίνακας 2.8 Περιγραφή Χρήσης Εντολής AT + PARAMETER

Σύνταξη εντολής	Απάντηση
AT+PARAMETER=<Spreading Factor>,	+OK

<Bandwidth>,<Coding Rate>, <Programmed Preamble> <Spreading Factor> τιμές 7-12 (προκαθορισμένο είναι το 12) <Bandwidth> 1-9 όπως φαίνεται παρακάτω 0 : 7.8KHz (δεν συνιστάται, εκτός προδιαγραφών.) 1 : 10.4KHz (δεν συνιστάται, εκτός προδιαγραφών.) 2 : 15.6KHz 3 : 20.8 KHz 4 : 31.25 KHz 5 : 41.7 KHz 6 : 62.5 KHz 7 : 125 KHz (προκαθορισμένο). 8 : 250 KHz 9 : 500 KHz <Coding Rate> 1-4 (προκαθορισμένο το 1) Χρησιμοποιείται για error correction. Χρησιμοποιούνται επιπλέον bits για να επιτευχθεί αυτό. Η τιμή 1 σημαίνει 25% πλεονασμός, 2 50% κοκ. Στην τιμή 4 το μεταφερόμενο πακέτο θα είναι 2 φορές το μέγεθος των αποστελλόμενων δεδομένων για καλύτερη διόρθωση λαθών. <Programmed Preamble> 4-7 (προκαθορισμένο το 4) Παράδειγμα: Θέτω τις τιμές των παραμέτρων ως εξής: <Spreading Factor> 7,<Bandwidth> 20.8KHz, <Coding Rate> 4,<Programmed Preamble> 5 Εντολή που χρησιμοποιώ: AT+PARAMETER=7,3,4,5	
AT+PARAMETER?	+PARAMETER=7,3,4,5

4. AT+BAND – Θέτω την συχνότητα RF (Radio Frequency)

Πίνακας 2.9 Περιγραφή Χρήσης Εντολής AT + BAND

• Σύνταξη εντολής	• Απάντηση
• AT+BAND=<parameter> • <parameter> είναι η συχνότητα RF και μετριέται σε Hz • Η προκαθορισμένη τιμή είναι 915000000Hz • • Παράδειγμα: Θέτω την συχνότητα στα 868500000Hz Εντολή που χρησιμοποιώ: • AT+BAND=868500000	• +OK
• AT+BAND?	• +BAND=868500000

5. AT+CPIN – Θέτω έναν AES128 κωδικό

Πίνακας 2.10 Περιγραφή Χρήσης Εντολής AT + CPIN

• Σύνταξη εντολής	• Απάντηση
• AT+CPIN=<Password> • <Password>: Ένας κωδικός AES 32 χαρακτήρων από 00000000000000000000000000000001 • έως FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF • • Μόνο με την χρήση του ίδιου κωδικού μπορούν να αναγνωριστούν τα δεδομένα. • Μετά την επαναφορά ρυθμίσεων ο κωδικός χάνεται. • • Παράδειγμα: Θέτω τον κωδικό ως εξής: • 71BC3FCBD373D90588E2D678B72DB90E • • Η εντολή που χρησιμοποιώ: • AT+CPIN= 71BC3FCBD373D90588E2D678B72DB90E •	• +OK

• AT+CPIN?	• +CPIN=71BC3FCBD373D90588E2D678B72DB90E +CPIN=No Password! (προκαθορισμένη τιμή)

6. AT+SEND Αποστολή δεδομένων (μηνύματος)

Πίνακας 2.11 Περιγραφή Χρήσης Εντολής AT + SEND

• Σύνταξη εντολής	Απάντηση
AT+SEND=<ADDRESS>,<Payload Length>,<Data> <Address> 0-65535 μία διεύθυνση που βρίσκονται οι συσκευές που θέλουν να επικοινωνήσουν. Όταν το Address είναι 0, τότε τα δεδομένα στέλνονται σε όλες τις διευθύνσεις (0-65535) <Payload Length> Το μέγεθος του μηνύματος, μέγιστη τιμή 240bytes <Data> Το μήνυμα σε ASCII μορφή Παράδειγμα: Αποστολή του κειμένου HELLO στην διεύθυνση 50 Η εντολή που χρησιμοποιώ: AT+SEND=50,5,HELLO	+OK
AT+SEND?	+SEND=50,5,HELLO

7. +RCV – Προβολή του εισερχόμενου μηνύματος

Πίνακας 2.12 Περιγραφή Χρήσης Εντολής RCV

• Σύνταξη εντολής	Απάντηση
• +RCV=<Address>,<Length>,<Data>,<RSSI>,<SNR> • <Address>Η διεύθυνση του αποστολέα <Length> Μέγεθος δεδομένων <Data> Τα δεδομένα <RSSI> Received Signal Strength Indicator - Ισχύς του σήματος <SNR> Signal-to-noise ratio – Αναλογία σήματος – θορύβου	•

Παράδειγμα: Λάβαμε ένα μήνυμα με διεύθυνση 50 και δεδομένα μεγέθους 5 bytes, το περιεχόμενο είναι HELLO, το RSSI είναι -99dBm και το SNR 40

+RCV=50,5,HELLO,=99,40

-

2.7 Android

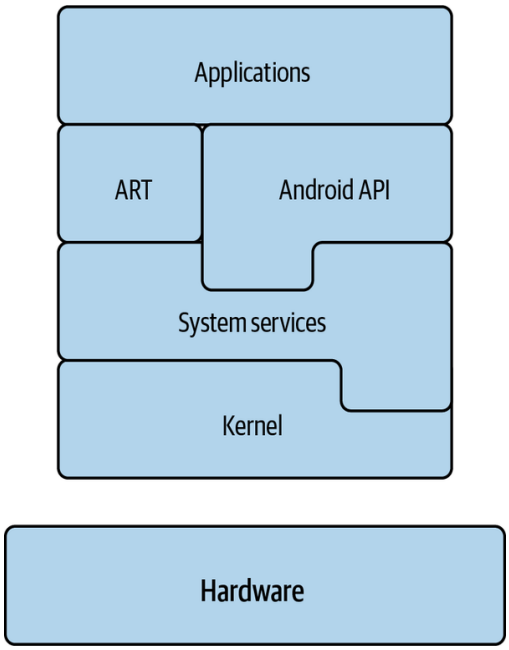
2.7.1 Εισαγωγή

Το Android δημιουργήθηκε τις αρχές του 2000, μία εποχή που η μνήμη RAM, η επεξεργαστική ισχύς και ο αποθηκευτικός χώρος ήταν περιορισμένα. Αρχικά ξεκίνησε σαν ένα λειτουργικό σύστημα για την βελτιστοποίηση των λειτουργικών συστημάτων των ψηφιακών καμερών, ενώ το 2005 εξαγοράστηκε από την Google. Πλέον το λειτουργικό σύστημα Android ταυτίζεται με τα κινητά τηλέφωνα.

Υπάρχουν πολλοί λόγοι για να επιλέξει κάποιος το Android.

- Ευρεία αποδοχή
- Όμορφο UI
- Βασισμένο σε Linux
- Open Source

2.7.2 Android Stack



Εικόνα 2.15 Android Stack – Πηγή
Programming Android With Kotlin

- Hardware
  Ενώ το Hardware δεν είναι κομμάτι του Android Stack, αξίζει να αναφερθεί καθώς το λειτουργικό σύστημα Android σχεδιάστηκε με μεγάλους περιορισμούς, με τον πιο βασικό απ'

όλους την ισχύ. Τα περισσότερα λειτουργικά συστήματα θεωρητικά έχουν μεγάλη ισχύ, αλλά το Android όχι.

- Kernel
  Ένας kernel είναι υπεύθυνος για τις βασικές λειτουργίες της συσκευής, όπως ένα filesystem, τα threads, τη διαχείριση του hardware κλπ. Το Android βασίζεται στο Linux Kernel. Ο Linux Kernel είναι open source και αυτό τον καθιστά δημοφιλή στους κατασκευαστές συσκευών και hardware.

- System Services
  Το επίπεδο του System Services είναι περίπλοκο και μεγάλο. Περιλαμβάνει μια πληθώρα βοηθητικών προγραμμάτων που εκτελείται ως μέρος του kernel, όπως τα προγράμματα οδήγησης (drivers) και βιβλιοθήκες για την κρυπτογραφία και την παρουσίαση μέσων.

- Android Runtime Environment
  Το επίπεδο πάνω από το system services είναι η υλοποίηση του Android Runtime Environment. Το Android Runtime Environment είναι μια συλλογή βιβλιοθηκών που μπορούμε να τις χρησιμοποιήσουμε στην εφαρμογή μας. Είναι υλοποιημένες κυρίως σε Java, C ή C++. Το μέρος που χρησιμοποιούμε κατά τον προγραμματισμό είναι υλοποιημένο σε Java ενώ από κάτω καλεί μεθόδους που είναι γραμμένες σε C/C++.

- Applications
  Στην κορυφή της στοίβας βρίσκονται οι εφαρμογές Android.

2.7.3 Android Studio

Το Android Studio είναι το επίσημο Integrated Development Environment (IDE) για την ανάπτυξη εφαρμογών Android, που βασίζεται στο IntelliJ IDEA. Διατίθεται για λειτουργικά συστήματα Windows, macOS και συστήματα που βασίζονται στο Linux. Στο Google I/O 2019, ανακοινώθηκε ότι η ανάπτυξη Android θα είναι ολοένα και περισσότερο Kotlin-first. Με τον όρο kotlin-first νοείται πως μεταξύ άλλων, δίνεται βάση στην online εκπαίδευση και παραδείγματα γραμμένα στην Kotlin αντί για Java. Η Java υποστηρίζεται ακόμα, όπως και η C++.

Στις 16 Μαΐου 2013 ανακοινώθηκε η έκδοση 0.1 ενώ πλέον βρίσκεται στην έκδοση Arctic Fox (2020.3.1) που βγήκε τον Ιούλιο του 2021

2.7.4 Design Patterns

Το Jetpack είναι μία σουίτα από βιβλιοθήκες με σκοπό να βοηθήσει τους developers να ακολουθήσουν τις καλές πρακτικές του προγραμματισμού, να μειώσουν το boilerplate code και να γράφουν κώδικα που δουλεύει με συνέπεια μεταξύ των εκδόσεων Android.

- Model–View–ViewModel

Η Google με την εισαγωγή του Jetpack υποστηρίζει μία αρχιτεκτονική που ονομάζεται Model-View-ViewModel (MVVM). Επειδή υποστηρίζεται εσωτερικά, είναι διαδεδομένο design pattern στις σύγχρονες εφαρμογές Android.

Στο MVVM ένα Activity ή ένα Fragment αναλαμβάνουν τον ρόλο του View.

Ο κώδικας του View περιέχει λίγα και βασικά πράγματα και όχι κάποια ιδιαίτερη λειτουργικότητα. Το ViewModel από την άλλη είναι υπεύθυνο για την ανανέωση του View και του Model. Το νέο χαρακτηριστικό αυτού του pattern είναι πως το Observable interface χρησιμοποιείται για να ενημερώσει το View όταν συμβεί μία αλλαγή στο Model. Δηλαδή το ViewModel θεωρεί τα viewable δεδομένα σαν Observables και το View απλά εγγράφεται σαν ένας Observer σε αυτά τα Observables και λαμβάνει ειδοποιήσεις για κάθε αλλαγή. Το Jetpack ονομάζει αυτά τα Observables Livedata.

Το ViewModel δεν έχει άμεσο reference στο View, απλά στέλνει ειδοποιήσεις.

Η χρήση του LiveData παρέχει μερικά πλεονεκτήματα όπως:

- Το UI ταιριάζει με την κατάσταση των δεδομένων.
- Το LiveData ακολουθεί το observer pattern. Το Livedata ειδοποιεί τα αντικείμενα Observer όταν αλλάζει κάτι στα δεδομένα. Μπορούμε να γράψουμε κώδικα για αν ανανεώσουμε το UI σε αυτά τα αντικείμενα Observer. Δεν χρειάζεται να ανανεωθεί το UI για κάθε αλλαγή στα δεδομένα, αφού το κάνει ο Observer.
- Αποφεύγονται τα memory leaks.
- Αποφεύγονται τα crashes λόγω σταματημένων activities.
- Αν το lifecycle του observer είναι ανενεργό, τότε δεν λαμβάνει ενημερώσεις από κάποιο συμβάν του LiveData
- Τα δεδομένα είναι πάντα ενήμερα.
- Ένα activity που ήταν στο παρασκήνιο, λαμβάνει τα τελευταία δεδομένα αμέσως μόλις επιστρέψει στο προσκήνιο

Κεφάλαιο 3ο: Τεχνολογίες και Βιβλιοθήκες που Χρησιμοποιήθηκαν

HTTP

Το HTTP είναι ένα πρωτόκολλο για την ανάκτηση πόρων όπως έγγραφα HTML. Βασίζεται στην αρχιτεκτονική client – server όπου ένας πελάτης μπορεί να στείλει ένα αίτημα για την απόκτηση ενός πόρου στον server και ο server να του τον φέρει. Είναι σχεδιασμένο στις αρχές της δεκαετίας του 1990 και βασίζεται στο αξιόπιστο πρωτόκολλο μεταφοράς TCP.

Websockets

Ενώ το πρωτόκολλο HTTP είναι ιδανικό για κατέβασμα σελίδων, APIs, ανέβασμα αρχείων κλπ, είναι σχετικά αργό. Κάθε φορά που στέλνουμε ένα HTTP αίτημα, πρέπει να εδραιώσουμε μια σύνδεση TCP με τον server, να περιμένουμε τον server να απαντήσει και έπειτα να κατεβάσουμε το αρχείο ή τα δεδομένα που θέλαμε. Εκτός από χρονοβόρα, η διαδικασία αυτή απαιτεί και παραπάνω πόρους συστήματος. Με τα Websockets αυτό αλλάζει καθώς μπορούμε να κρατήσουμε ανοιχτή μια σύνδεση TCP και να μπορούμε να έχουμε αμφίδρομη επικοινωνία με τον server.

ESPAsyncWebServer

Η βιβλιοθήκη ESPAsyncWebServer αποτελεί μία βιβλιοθήκη ασύγχρονου web server για το ESP8266 με πολλές δυνατότητες. Με τον όρο ασύγχρονο εννοείται ότι μπορεί να διαχειριστεί παραπάνω από ένα αίτημα την ίδια στιγμή, ενώ γνωρίζει πότε να κλείσει μία σύνδεση και να απελευθερώσει τους πόρους του συστήματος.

Arduino Websockets

Το Arduino Websockets αποτελεί μία βιβλιοθήκη για τη δημιουργία εφαρμογών με τη χρήση Websocket. Είναι γραμμένη από τον χρήστη του github Links2004 και είναι συμβατή με αρκετά Arduino boards, μεταξύ αυτών και του ESP8266.

Retrofit2

Η βιβλιοθήκη Retrofit είναι ένας type-safe HTTP client για Android και το JVM. Με το Retrofit είναι εύκολο να ανακτήσεις ή να στείλεις JSON ή άλλου τύπου δεδομένα σε ένα REST webservice, ενώ χρησιμοποιεί τη βιβλιοθήκη OkHttp για τα HTTP αιτήματα. Χρησιμοποιεί annotations για την παραμετροποίησή του. Για παράδειγμα το annotation @GET("URL HERE") δηλώνει ότι το αίτημα που θέλουμε να στείλουμε είναι του τύπου GET, ενώ στην παρένθεση μπαίνει η διεύθυνση που θέλουμε να στείλουμε το αίτημα. Από προεπιλογή μπορεί να κάνει deserialize μόνο HTTP bodies στον τύπο ResponseBody του OkHttp ενώ μπορούν να χρησιμοποιηθούν μετατροπείς για να υποστηρίξει και άλλους τύπους. Μερικοί από αυτούς τους μετατροπείς είναι το Gson, το Jackson, το Moshi, το Protobf, το Simple XML και άλλοι.

Java Websockets

Η βιβλιοθήκη Java Websockets από τον χρήστη του github TooTallNate περιέχει μια υλοποίηση για Websocket client γραμμένη σε Java. Παρέχει μια εύχρηστη και ολοκληρωμένη διεπαφή ενώ οι υποκείμενες κλάσεις είναι γραμμένες με το java.nio. Η διεπαφή περιλαμβάνει μεταξύ άλλων μεθόδους για το άνοιγμα ενός socket, το κλείσιμο, την αποστολή δεδομένων, τη λήψη, την κατάσταση του socket αν είναι ανοιχτό ή κλειστό και άλλα.

Postman

Το πρόγραμμα Postman είναι μία πλατφόρμα για την δημιουργία και δοκιμή API. Λειτουργεί σαν HTTP client με τον οποίο μπορείς να στείλεις αιτήματα μέσω μίας εύκολης στη χρήση διεπαφής, να τα αποθηκεύσεις για μελλοντική χρήση, να τα ομαδοποιήσεις και να τρέξεις αυτοματοποιημένα tests ενώ παράλληλα έχει και τη δυνατότητα επικοινωνίας μέσω websocket.

Koin

Το Koin είναι ένα ελαφρύ framework για Dependency Injection για την Kotlin. Είναι γραμμένο σε Kotlin ενώ υποστηρίζει την δυνατότητα του Kotlin DSL.

Κεφάλαιο 4ο: Ανάλυση Κώδικα της Εφαρμογής

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει ανάλυση του κώδικα των τριών τμημάτων της εφαρμογής. Τα τμήματα αυτά είναι τρία και αφορούν το κατώτερο layer που είναι για το Arduino, το μεσαίο που είναι ο websocket client και το τρίτο η εφαρμογή Android.

Καθ' όλη την παρουσίαση υπάρχουν στιγμιότυπα και γίνεται ανάλυσή τους.

4.2 Κώδικας Arduino

Παρακάτω παρουσιάζεται και αναλύεται λεπτομερώς ο κώδικας που τρέχει στο ESP8266.

```
1  #ifdef ESP8266
2  #include <ESP8266WiFi.h>
3  #elif defined(ESP32)
4  #include <WiFi.h>
5  #else
6  #error "Board not found"
7  #endif
8
9  #include <WebSocketsServer.h>
10 #include <ESPAsyncWebServer.h>
11 #include <SoftwareSerial.h>                    //we have to include the SoftwareSerial library, or else we can't use it
12
13
14 #define rx 13                                   //LORA TX
15 #define tx 15                                   //LORA RX
16
17 SoftwareSerial loraSerial(rx, tx);              //define how the soft serial port is going to work
18
19 #ifndef APSSID
20 #define APSSID "LoRaChat 1"
21 #define APPSK  "ARandomPasswordHere"
22 #endif
23
24 #define USE_SERIAL Serial1
25
26 const char *ssid = APSSID;
27 const char *password = APPSK;
28
29 AsyncWebServer server(80);
30 WebSocketsServer websockets(81);
31
32
```

Εικόνα 4.1 Κώδικας Arduino 1

Ξεκινώντας, υπάρχουν κάποια definitions και συμπεριλήψεις βιβλιοθηκών που είναι απαραίτητα για την λειτουργία της εφαρμογής. Αρχικά συμπεριλαμβάνεται η βιβλιοθήκη ESP8266WiFi η οποία είναι απαραίτητη για την λειτουργία και τη ρύθμιση του WiFi. Στη γραμμή 9-11 συμπεριλαμβάνονται οι βιβλιοθήκες WebSocketsServer που χρησιμεύει για τη δημιουργία του websocket server, το ESPAsyncWebServer που χρησιμοποιείται για τη δημιουργία ενός web server, όπως θα φανεί παρακάτω και το SoftwareSerial που χρησιμοποιείται για την σειριακή επικοινωνία του πομποδέκτη LoRa με το NodeMCU ESP8266. Στις γραμμές 14, 15 και 17 φαίνεται ακριβώς πως ορίζονται τα pins για το Receive και το Transmit και η δημιουργία ενός αντικειμένου για την σειριακή επικοινωνία του ESP με τον πομποδέκτη. Παρακάτω ορίζονται το όνομα του δικτύου WiFi που θα συνδεθεί το κινητό και ο κωδικός πρόσβασης. Τέλος δηλώνονται οι 2 μεταβλητές για τον http server και τον websocket server με παραμέτρους τις πόρτες 80 και 81 αντίστοιχα.

```
35 void initWifi(){
36     Serial.println("Configuring access point...");
37   /* You can remove the password parameter if you want the AP to be open. NOT recommended. */
38   WiFi.mode(WIFI_AP);
39   WiFi.softAP(ssid, password);
40 
41   IPAddress myIP = WiFi.softAPIP();
42   Serial.print("AP IP address: ");
43   Serial.println(myIP);
44 
45 
46   }
47 
48 
49 void notFound(AsyncWebServerRequest *request)
50 {
51   request->send(404, "text/plain", "Page Not found");
52 }
53 
```

Εικόνα 4.2 Κώδικας Arduino 2

Σε αυτό το σημείο του κώδικα γίνεται η αρχικοποίηση του WiFi βάζοντάς το σε κατάσταση Access Point, όπως φαίνεται στη γραμμή 38 και γίνεται εκτύπωση στην κονσόλα η IP. Η προκαθορισμένη IP του ESP είναι η 192.168.4.1, ενώ μπορεί να αλλάξει σε αυτό το σημείο δίνοντας τις κατάλληλες παραμέτρους. Η μέθοδος notFound() χρησιμοποιείται παρακάτω από τον web server για την περίπτωση που στείλουμε αίτημα για κάποιον πόρο που δεν υπάρχει

```
61 void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length) {
62
63     switch(type) {
64         case WStype_DISCONNECTED:
65             USE_SERIAL.printf("[%u] Disconnected!\n", num);
66             break;
67         case WStype_CONNECTED:
68             {
69                 IPAddress ip = websockets.remoteIP(num);
70                 USE_SERIAL.printf("[%u] Connected from %d.%d.%d.%d url: %s\n", num, ip[0], ip[1], ip[2], ip[3], payload);
71
72               // send message to client
73               //websockets.sendTXT(num, "Welcome");
74             }
75             break;
76         case WStype_TEXT: {
77             USE_SERIAL.printf("[%u] get Text: %s\n", num, payload);
78
79             String payload_str = String((char*) payload);
80
81             Serial.println(payload_str); //debug
82
83             if(payload_str.length()>=2){ //2 is the minimum command rylr length (AT)
84               loraSerial.println(payload_str);
85              }
86
87             break;
88             }
89         case WStype_BIN:
90             USE_SERIAL.printf("[%u] get binary length: %u\n", num, length);
91             hexdump(payload, length);
92
93             // send message to client
94             // webSocket.sendBIN(num, payload, length);
95             break;
96     }
97
98 }
```

Εικόνα 4.3 Κώδικας Arduino 3

Η μέθοδος webSocketEvent() έχει ένα switch case για τους διάφορους τύπους μηνυμάτων που μπορούμε να λάβουμε και για όταν ανοίγει και όταν κλείνει μία σύνδεση. Κυρίως όμως γίνεται χρήση του WStype_TEXT αφού στο LoRa Chat γίνεται ανταλλαγή κειμένου. Το USE_SERIAL που έχει αρχικοποιηθεί στην αρχή και το Serial χρησιμοποιούνται για debugging μέσω σειριακής σύνδεσης με τον υπολογιστή. Στη γραμμή 79 αποθηκεύεται στη μεταβλητή payload_str το μήνυμα που έλαβε ο websocket server από τον client και εφόσον είναι μήκους μεγαλύτερου ή ίσου του 2, τότε στη σειριακή σύνδεση με τον πομποδέκτη LoRa στέλνεται αυτό το μήνυμα. Το μέγεθος 2 δεν είναι τυχαίο καθώς το μικρότερο command που μπορεί να λάβει ο πομποδέκτης είναι το “AT”.

```
100 void setup() {
101
102   Serial.begin(115200);
103   initWifi();
104
105   delay(250);
106   loraSerial.begin(38400);
107   delay(250);
108   loraSerial.println("AT+BAND=869500000");
109
110
111
112   server.onNotFound(notFound);
113
114   server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
115         request->send(200, "application/json", "{\"message\": \"esp\"}");
116     });
117
118
119
120   websockets.enableHeartbeat(5000, 3000, 2);
121
122   server.begin();
123   websockets.begin();
124
125   websockets.onEvent(webSocketEvent);
126
127
128
129
130 }
```

Εικόνα 4.4 Κώδικας Arduino 4

Στη μέθοδο setup(), η οποία εκτελείται μία φορά στην αρχή δίνονται κάποιες εντολές που θέλουμε να εκτελούνται κάθε φορά που μπαίνει σε λειτουργία η συσκευή. Στη γραμμή 102 ξεκινάει η σειριακή σύνδεση στα 115200 baud (προκαθορισμένο) μεταξύ του NodeMCU ESP8266 και του υπολογιστή, κυρίως για debugging, ενώ παρακάτω ξεκινάει η σειριακή σύνδεση μεταξύ του NodeMCU ESP8266 και του πομποδέκτη και δίνονται κάποιες τιμές στον πομποδέκτη. Στη γραμμή 112 ορίζεται να εκτελείται η μέθοδος notFound(), ενώ στην 114 ορίζεται ότι στο path “http://192.168.4.1/” όταν στέλνεται GET request, τότε θα επιστρέφει ένα json μήνυμα.

```
132 void loop() {
133 websockets.loop();
134
135 //Used for debugging
136 if (Serial.available() > 0) {
137
138   String input = Serial.readString();
139
140   Serial.print(" I received: ");
141
142   Serial.println(input);
143   websockets.broadcastTXT(input);
144 }
145 //end
146
147  if(loraSerial.available() > 0){ //if I have data from lora
148   String input = loraSerial.readString();
149
150   websockets.broadcastTXT(input); //send data to the socket client - phone
151  }
152
153 }
```

Εικόνα 4.5 Κώδικας Arduino 5

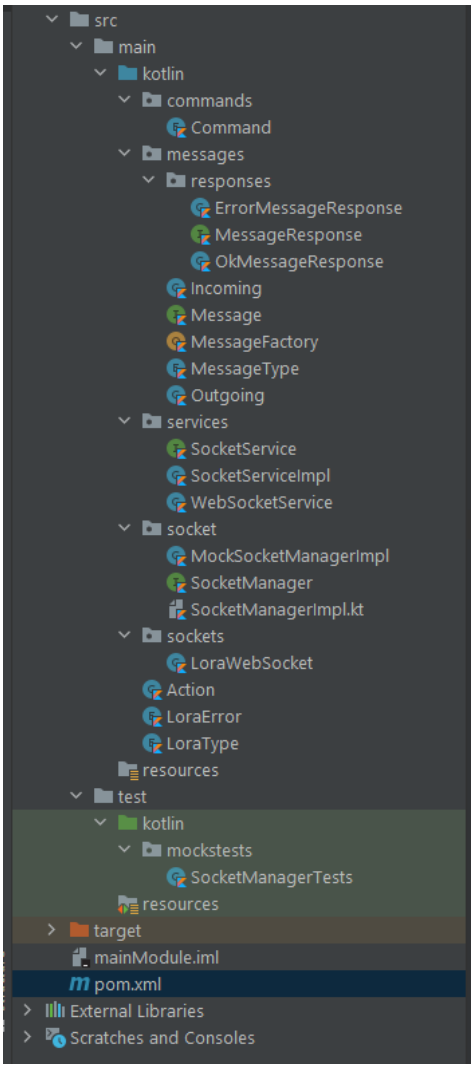
Η μέθοδος loop() είναι ένας ατέρμων βρόγχος και εκτελείται αμέσως μετά την μέθοδο setup() για όσο λειτουργεί η συσκευή. Στη γραμμή 147 γίνεται έλεγχος αν η σειριακή σύνδεση με τον πομποδέκτη έχει δεδομένα, τότε διαβάζονται αυτά τα δεδομένα και στέλνονται στον socket client.

4.3 Κώδικας του Module

4.3.1 Εισαγωγή

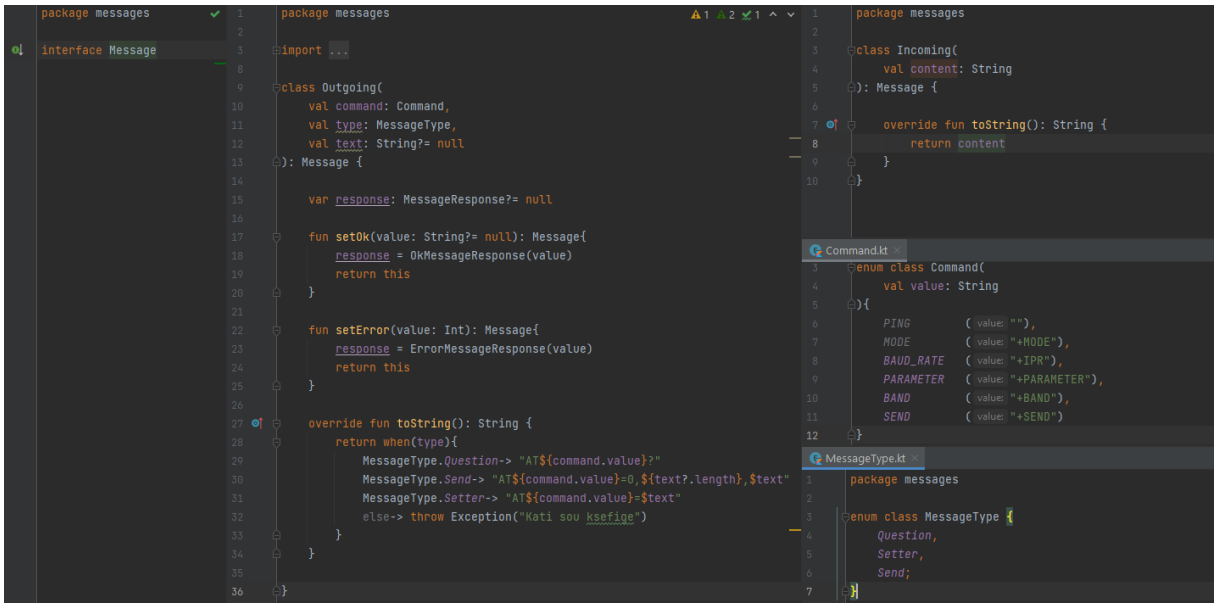
Σε αυτό το υποκεφάλαιο παρουσιάζεται η δομή του project και έπειτα θα αναλύονται ξεχωριστά οι κλάσεις και οι μέθοδοι σε αυτές.

Η παρουσίαση θα γίνει από την κορυφή εκτέλεσης του κώδικα και σταδιακά θα αναφέρονται τα επιμέρους στοιχεία, έως ότου φτάσουμε στο χαμηλότερο σημείο εκτέλεσης που είναι η επικοινωνία μέσω της βιβλιοθήκης Java Websockets.



Εικόνα 4.6 Δομή του Websocket Module

4.3.2 Message



Εικόνα 4.7 Message και οι Υποκλάσεις του

Τα μηνύματα που ανταλλάσσονται ανάμεσα στον websocket client και τον server είναι δύο τύπων. Ο πρώτος τύπος και ο πιο απλός είναι ο Incoming που αντιπροσωπεύει τα μηνύματα τα οποία θα λάβει ο πομποδέκτης μας από κάποιον άλλον πομποδέκτη. Το περιεχόμενο αυτό είναι του τύπου String.

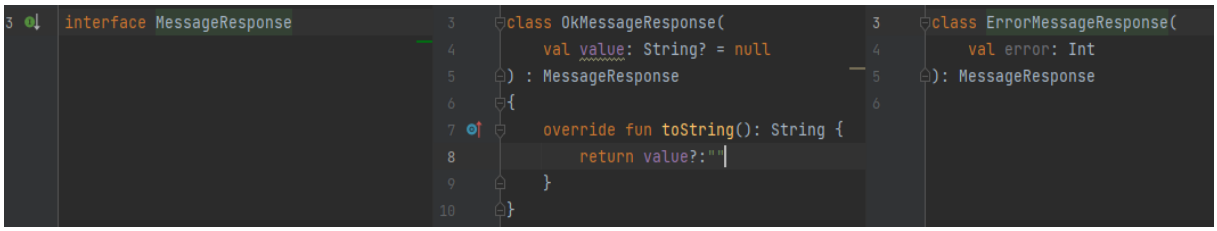
Ο δεύτερος τύπος είναι ο Outgoing και έχει μερικές μεταβλητές. Τη μεταβλητή command που αναφέρεται στον τύπο των AT commands που αναλύθηκαν στο κεφάλαιο σχετικά με το Reyax RYLR896, το type, το text και το response που θα αναλυθεί παρακάτω. Ο Outgoing τύπος (μεταβλητή type) μηνυμάτων αναφέρεται στα μηνύματα που στέλνει ο χρήστης στον πομποδέκτη και χωρίζεται σε τρεις τύπους μηνυμάτων:

1. **Question**
   Τα Outgoing μηνύματα τύπου Question στέλνουν ερώτημα στον πομποδέκτη LoRa για να πάρουμε σαν απάντηση την τιμή μιας παραμέτρου εκείνη τη στιγμή. Για παράδειγμα για να ρωτήσουμε το BAND, εσωτερικά στέλνεται ένα μήνυμα “AT+BAND?”.

2. **Setter**
   Ο τύπος Setter χρησιμοποιείται για να θέσουμε μία τιμή για κάποια συγκεκριμένη ρύθμιση στον πομποδέκτη. Εσωτερικά στέλνεται ένα μήνυμα της μορφής “AT+COMMAND=TIMH”.

3. **Send**
   Ο τύπος Send χρησιμοποιείται όταν στέλνουμε ένα μήνυμα κειμένου από τον έναν πομποδέκτη σε έναν άλλον. Το μήνυμα αυτό αποθηκεύεται στην μεταβλητή text που είναι τύπου String.

4.3.3 Message Response



Εικόνα 4.8 MessageResponse

Η μεταβλητή response είναι τύπου MessageResponse και είναι δύο τύπων.

1. **OkMessageResponse**
   OkMessageResponse παίρνουμε όταν ένα command που στείλαμε εκτελέστηκε σωστά. Αν το Command ήταν του τύπου Question, τότε στη μεταβλητή value αποθηκεύεται η τιμή που λάβαμε από τον πομποδέκτη LoRa.

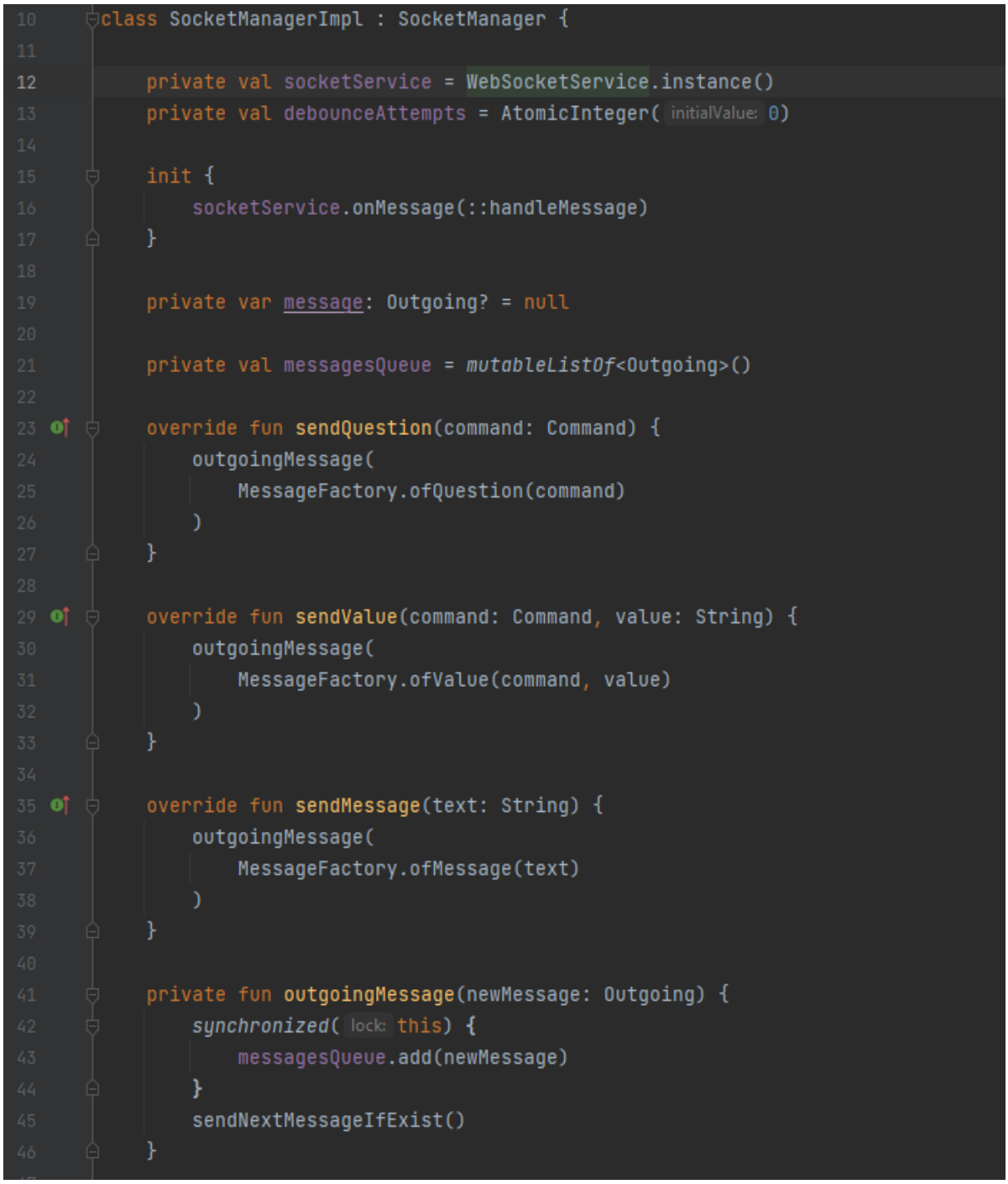
2. **ErrorMessageResponse**

ErrorMessageResponse παίρνουμε όταν ένα command που στείλαμε δεν εκτελέστηκε σωστά. Κρατάει σε μία μεταβλητή error τον κωδικό του λάθους.

4.3.4 SocketManager

Εικόνα 4.9 Interface του SocketManager

Στον παραπάνω κώδικα παρουσιάζεται το interface του SocketManager. Αυτό το interface περιέχει τις κύριες μεθόδους του module που αφορούν το socket. Με τη χρήση αυτών των μεθόδων μπορούμε να κάνουμε σύνδεση στον socket server, να στείλουμε κάποιο ερώτημα στο RYLR896, να θέσουμε μία τιμή, να στείλουμε και να λάβουμε κάποιο μήνυμα. Παρακάτω θα παρουσιαστεί αναλυτικά η υλοποίηση του interface.



Εικόνα 4.10 Υλοποίηση του SocketManager

Ο SocketManager είναι ένας wrapper ουσιαστικά του WebSocketService για την πιο εύκολη χρήση των μεθόδων της βιβλιοθήκης για το RYLR896.

Αρχικά δημιουργείται ένα αντικείμενο τύπου WebsocketService, το οποίο είναι singleton. Η private μέθοδος outgoingMessage βάζει τα μηνύματα στην ουρά messagesQueue και καλεί την sendNextMessageIfExist(). Όλες οι μέθοδοι που στέλνουν μηνύματα στον πομποδέκτη του τύπου Question, Setter ή Send καλούν την outgoingMessage().

Όπως φαίνεται χρησιμοποιείται και το factory design pattern το οποίο δημιουργεί αντικείμενα είτε Incoming είτε Outgoing ανάλογα με τις ανάγκες. Για παράδειγμα η sendQuestion(command: Command) καλεί την MessageFactory.ofQuestion(command) και εσωτερικά δημιουργείται ένα αντικείμενο τύπου Outgoing για το συγκεκριμένο Command και το messageType είναι τύπου Question, όπως φαίνεται παρακάτω.

```
5   object MessageFactory {
6
7       fun ofQuestion(command: Command): Outgoing{
8           return Outgoing(command, MessageType.Question)
9       }
10
11      fun ofMessage(text: String): Outgoing{
12          return Outgoing(Command.SEND, MessageType.Send, text)
13      }
14
15      fun ofValue(command: Command, value: String): Outgoing{
16          return Outgoing(command, MessageType.Setter, value)
17      }
18
19      fun ofIncoming(content: String): Incoming{
20
21          return Incoming(content)
22      }
23  }
```

Εικόνα 4.11 MessageFactory

Η μέθοδος sendNextMessageIfExist() αρχικά ελέγχει αν το socket είναι ανοιχτό και κάνει 3 προσπάθειες επαναποστολής.

Παρακάτω φαίνεται ο κώδικας της μεθόδου. Η χρήση του AtomicInteger γίνεται επειδή χρησιμοποιείται μέσα σε Thread.

```
97      private fun sendNextMessageIfExist() {
98          if (!socketService.isOpen()) {
99
100             debounceAttempts.incrementAndGet()
101             if (debounceAttempts.get() >= 3) {
102                 debounceAttempts.set(0)
103                 return
104             }
105
106
107             try {
108                 Thread.sleep( millis: 200L + (200 * debounceAttempts.get()))
109             } catch (e: Exception) {
110
111             } finally {
112                 sendNextMessageIfExist()
113             }
114
115         }
116
117         debounceAttempts.set(0)
118
119         synchronized( lock: this) {
120             if (message == null) {
121                 messagesQueue.removeFirstOrNull()?.let {  it: Outgoing
122                     send(it)
123                 }
124             }
125         }
126     }
```

Εικόνα 4.12 Η Μέθοδος sendNextMessageIfExist

4.3.5 Διαχείριση των Μηνυμάτων

Το RYLR896 επικοινωνεί με AT commands. Για να αξιοποιηθούν τα μηνύματα αυτά θα πρέπει να εξάγουμε από τα String που λαμβάνουμε κάποιες πληροφορίες.

Η διαχείριση των μηνυμάτων παντός τύπου που λαμβάνουμε από το RYLR896 μέσω socket γίνεται από την μέθοδο handleMessage().

Η handleMessage() ελέγχει με ένα switch case αν το μήνυμα που λάβαμε είναι εισερχόμενο μήνυμα κειμένου, είτε αν στείλαμε κάποια τιμή και πήραμε σαν απάντηση OK, αν πήραμε κάποιο μήνυμα λάθους ή αν ρωτήσαμε την τιμή ενός Command και εξάγει τις πληροφορίες που θέλουμε.

```
private fun handleMessage(content: String) {
    val msg: Message = when {
        content.startsWith( prefix: "+RCV",  ignoreCase: true) -> {
            val messageContent = content.replace( oldValue: "+RCV=",  newValue: "").split( ...delimiters: ",").getOrNull( index: 2) ?: "Error - Unreadable Message"
            MessageFactory.ofIncoming(messageContent)
        }
        content.startsWith( prefix: "+OK") -> {
            this.message!!.setOk()
        }
        content.startsWith( prefix: "+ERR") -> {
            this.message!!.setError(content.trim().replace( oldValue: "+ERR=",  newValue: "").toIntOrNull()?:-1)
        }
        else -> {
            val command = this.message!!.command
            this.message!!.setOk(content.replace( oldValue: "${command.value}=",  newValue: ""))
        }
    }

    notify(msg)
}
```

Εικόνα 4.13 Η Μέθοδος handleMessage

4.3.6 WebSocketService

Η κλάση WebSocketService αποτελεί έναν wrapper για τις βασικές μεθόδους της βιβλιοθήκης που θα παρουσιαστούν στη συνέχεια. Όπως προαναφέρθηκε και πριν είναι singleton και η δημιουργία αντικειμένου πραγματοποιείται με τη μέθοδο instance().

Στην αρχή δημιουργείται ένα αντικείμενο τύπου LoraWebsocket, το οποίο αποτελεί το κατώτερο σημείο της σύνδεσης με τον socket server. Έπειτα ακολουθούν μέθοδοι για τη σύνδεση, την αποστολή και λήψη μηνυμάτων και την κατάσταση του socket, δηλαδή αν είναι ανοιχτό ή κλειστό.

Παρακάτω φαίνεται η υλοποίηση της κλάσης WebsockeSerice.

Εικόνα 4.14 Η Κλάση WebSocketService

4.3.7 LoraWebSocket

Η κλάση LoraWebsocket κληρονομεί την κλάση WebSocketClient της βιβλιοθήκης και αποτελεί μία απλή υλοποίηση των βασικών μεθόδων της που είναι αναγκαίες για την χρήση της βιβλιοθήκης. Αυτές είναι η onOpen(), η onMessage(), η onClose() και η onError().

Η επικοινωνία ανάμεσα στην LoraWebsocket και WebsocketClient γίνεται μέσω callbacks τα οποία μεταβιβάζουν κάθε μήνυμα που υπάρχει. Τα callbacks γίνονται invoke, δηλαδή εκτελούνται από τις κύριες μεθόδους της βιβλιοθήκης. Η μέθοδος onMessage() κάνει invoke το onMessageCallback με παράμετρο το μήνυμα που λήφθηκε. Η μέθοδος onOpen() και onClose() κάνουν invoke την onStatusChangeCallback με παράμετρο τύπου Boolean ώστε να σηματοδοτήσει ότι έκλεισε ή άνοιξε το socket. Έτσι στο ανώτερο επίπεδο μπορεί κάποιος να χρησιμοποιήσει τα callbacks ώστε να πάρει το περιεχόμενο του μηνύματος που λήφθηκε και να ενημερωθεί για τις αλλαγές στο status της σύνδεσης.

```
14  class LoraWebSocket : WebSocketClient {
15      constructor() : super(URI( str: "ws://192.168.4.1:81"))
16
17      override fun onOpen(handshakedata: ServerHandshake) {
18          println("opened")
19          this.onStatusChangeCallback?.invoke(true) //closed opened
20
21      }
22
23      override fun onMessage(message: String) {
24          println("debug: $message")
25          this.onMessageCallback?.invoke(message)
26      }
27
28      private var onMessageCallback: ((String) -> Unit)? = null
29      fun onMessageCallback(onMessageCallback: ((String) -> Unit)?) {
30          this.onMessageCallback = onMessageCallback
31      }
32
33      private var onStatusChangeCallback: ((Boolean) -> Unit)? = null
34      fun onStatusChangeCallback(onStatusChangeCallback: ((Boolean) -> Unit)?) {
35          this.onStatusChangeCallback = onStatusChangeCallback
36      }
37
38      override fun onClose(code: Int, reason: String, remote: Boolean) {
39          println("closed")
40          this.onStatusChangeCallback?.invoke(false) //closed socket
41      }
42
43      override fun onError(ex: Exception) {
44          ex.printStackTrace()
45          // if the error is fatal then onClose will be called additionally
46      }
47  }
```

Εικόνα 4.15 Η Κλάση LoraWebSocket

4.4 Κώδικας Android

4.4.1 Εισαγωγή

Σε αυτό το σημείο παρουσιάζεται η δομή του Android project και στη συνέχεια ακολουθεί η περιγραφή των Activities και των ViewModel τους και παράλληλα παρουσίαση πιο συγκεκριμένων λειτουργιών και πως υλοποιήθηκαν.

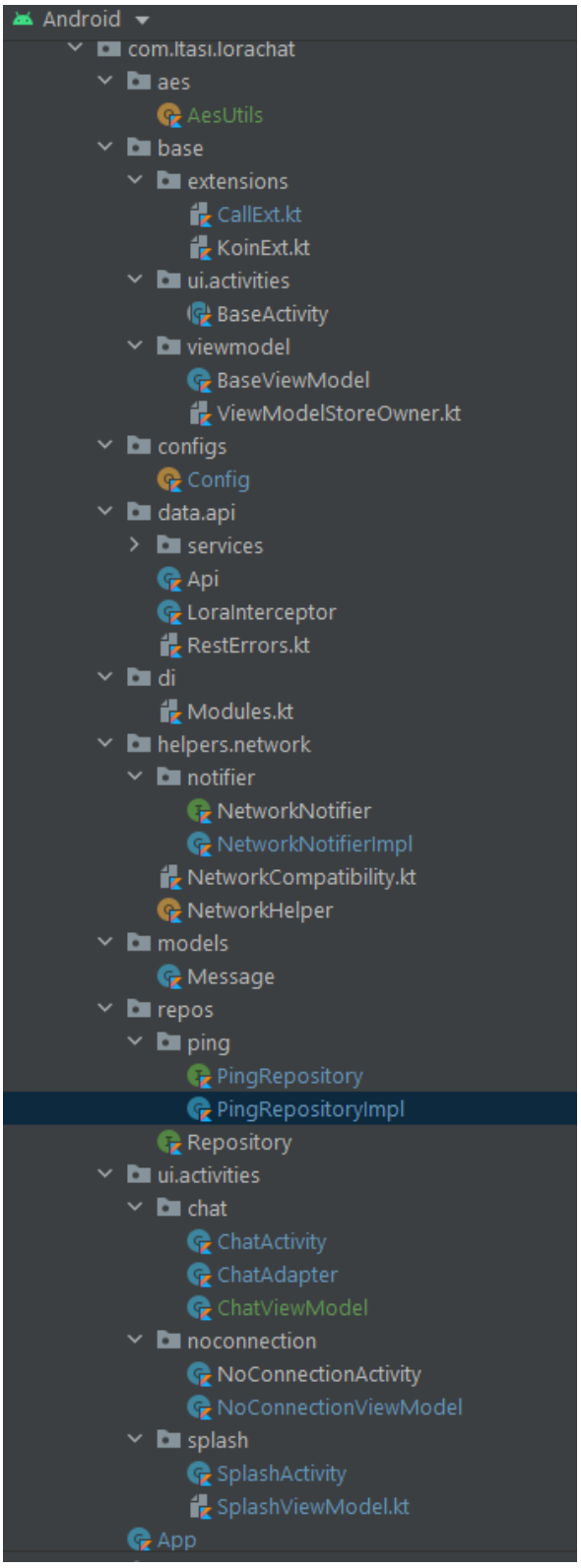
4.4.2 Ενσωμάτωση του Kotlin module στο Android

Για τη δημιουργία του Kotlin module χρησιμοποιήθηκε το εργαλείο διαχείρισης Maven ενώ για το Android χρησιμοποιήθηκε το Gradle. Για ευκολία, η ενσωμάτωση του module έγινε δημιουργώντας ένα .jar αρχείο που περιλαμβάνει τον κώδικα και τα dependencies που υπήρχαν στο pom.xml αρχείο του project και στη συνέχεια έγινε import σαν βιβλιοθήκη στο build.gradle αρχείο στο Android project. Η εισαγωγή είναι εύκολη καθώς απαιτεί μία γραμμή στο μπλοκ dependencies του build.gradle.

Αυτή η γραμμή είναι η ακόλουθη:

implementation files('path to jar file.jar')

Μετά την ενσωμάτωση του .jar πρέπει να γίνει Sync το Gradle και έπειτα μπορούμε να δούμε τις μεθόδους της βιβλιοθήκης.



Εικόνα 4.16 Δομή του Android Project

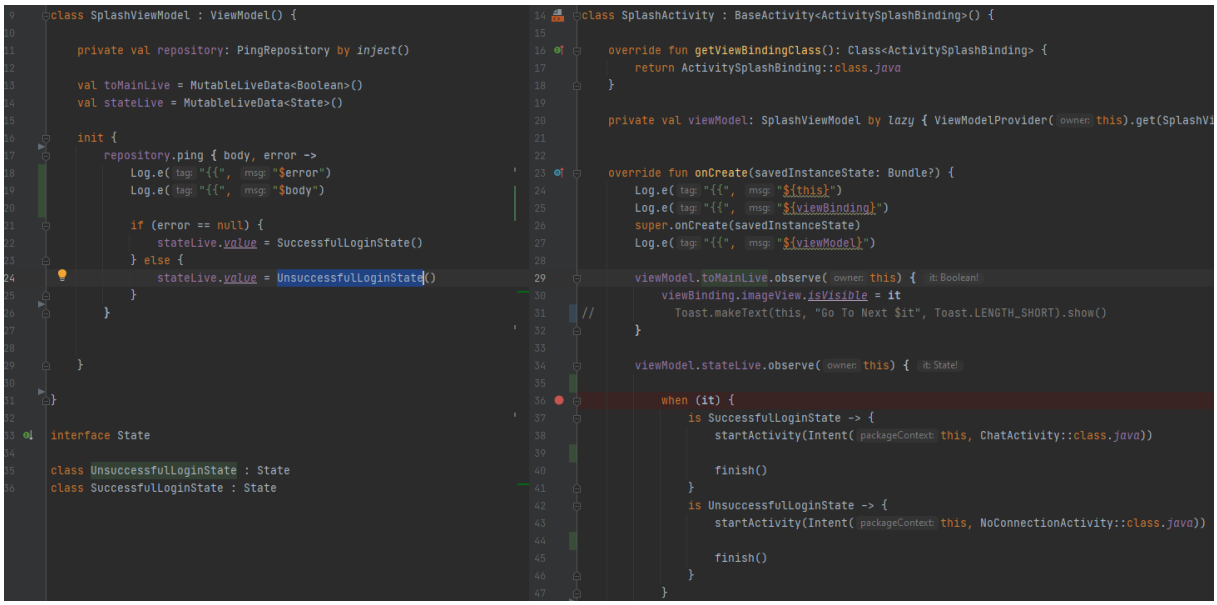
4.4.3 SplashActivity

Οι κλάσεις SplashActivity και SplashViewModel περιέχουν τη λειτουργία για την αρχικό άνοιγμα της εφαρμογής και το αν θα πρέπει να συνεχίσει στην επόμενη οθόνη, την οθόνη του Chat.

Με το άνοιγμα της εφαρμογής στέλνεται ένα GET request στη διεύθυνση http://192.168.4.1/ που τρέχει ένας web server από το SplashViewModel.

Αν ληφθεί το σωστό json response τότε στο SplashViewModel θέτεται το value του stateLive, μιας μεταβλητής τύπου MutableLiveData<State> σε SuccessfulLoginState(), αλλιώς αν για κάποιο λόγο αποτύχει το request τότε θέτει την τιμή σε UnsuccessfulLoginState().

Το SplashActivity κάνει observe τις μεταβολές αυτής της τιμής και ανάλογα με το state αν είναι Successful ή Unsuccessful ανοίγει την ChatActivity ή την NoConnectionActivity αντίστοιχα.



```
class SplashViewModel : ViewModel() {

    private val repository: PingRepository by inject()

    val toMainLive = MutableLiveData<Boolean>()
    val stateLive = MutableLiveData<State>()

    init {
        repository.ping { body, error ->
            Log.e( tag: "{{", msg: "$error")
            Log.e( tag: "{{", msg: "$body")

            if (error == null) {
                stateLive.value = SuccessfulLoginState()
            } else {
                stateLive.value = UnsuccessfulLoginState()
            }
        }


    }

}

interface State

class UnsuccessfulLoginState : State
class SuccessfulLoginState : State
```

```
class SplashActivity : BaseActivity<ActivitySplashBinding>() {

    override fun getViewBindingClass(): Class<ActivitySplashBinding> {
        return ActivitySplashBinding::class.java
    }

    private val viewModel: SplashViewModel by lazy { ViewModelProvider( owner: this).get(SplashVi

    override fun onCreate(savedInstanceState: Bundle?) {
        Log.e( tag: "{{", msg: "${this}")
        Log.e( tag: "{{", msg: "${viewBinding}")
        super.onCreate(savedInstanceState)
        Log.e( tag: "{{", msg: "${viewModel}")

        viewModel.toMainLive.observe( owner: this) {  it: Boolean!
            viewBinding.imageView.isVisible = it
//            Toast.makeText(this, "Go To Next $it", Toast.LENGTH_SHORT).show()
        }

        viewModel.stateLive.observe( owner: this) {  it: State!

            when (it) {
                is SuccessfulLoginState -> {
                    startActivity(Intent( packageContext: this, ChatActivity::class.java))

                    finish()
                }
                is UnsuccessfulLoginState -> {
                    startActivity(Intent( packageContext: this, NoConnectionActivity::class.java))

                    finish()
                }
            }
```

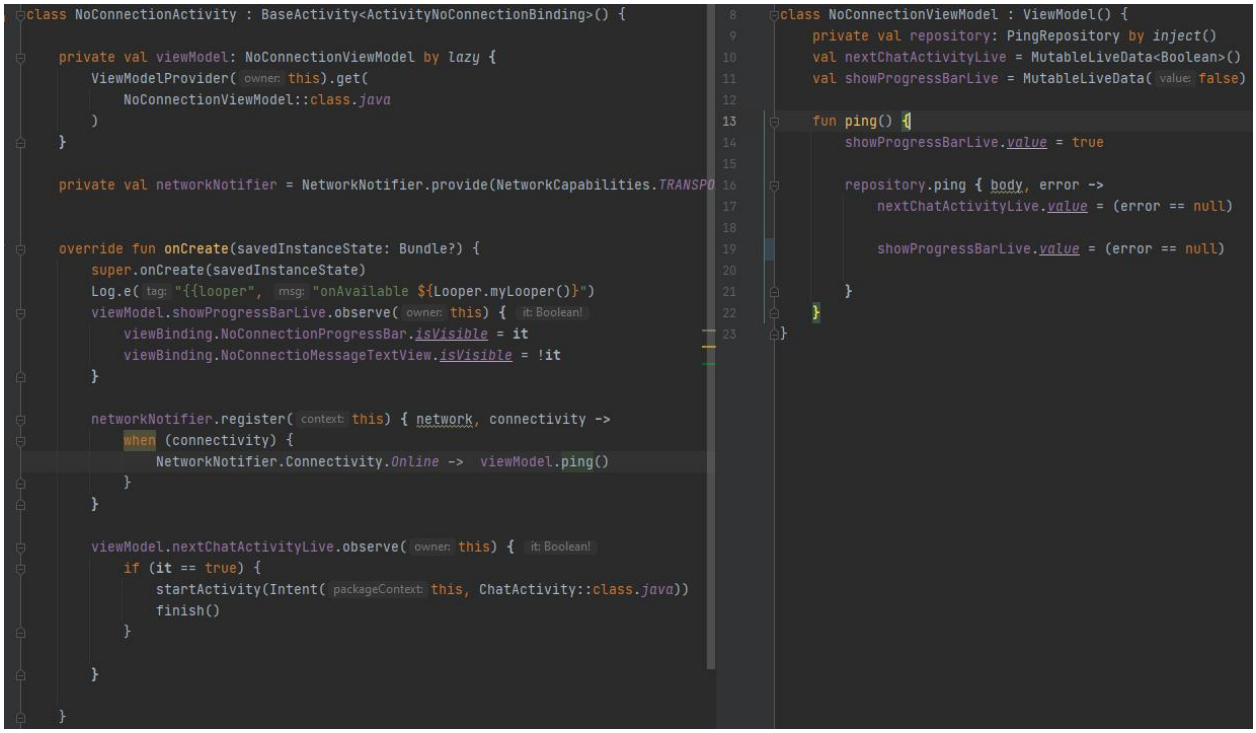
Εικόνα 4.17 SplashActivity & SplashViewModel

4.4.4 NoConnectionActivity

Οι κλάσεις NoConnectionActivity και NoConnectionViewModel περιέχουν τις λειτουργίες για την μετάβαση στο ChatActivity σε περίπτωση που επανέλθει η σύνδεση με το Wi-Fi.

Η κλάση ChatActivity χρησιμοποιεί μία βοηθητική κλάση με όνομα NetworkNotifier η οποία δίνει τη δυνατότητα ενημέρωσης όταν αλλάζει κάτι στη συνδεσιμότητα της συσκευής, δηλαδή αν συνδέθηκε στο δίκτυο και τι τύπος δικτύου είναι.

Όταν αλλάξει κάτι στη συνδεσιμότητα, τότε το NoConnectionActivity θα καλέσει την μέθοδο ping() που στέλνει GET request στον webserver και θα ανοίξει την οθόνη του Chat αν είναι το request είναι επιτυχές.



```
class NoConnectionActivity : BaseActivity<ActivityNoConnectionBinding>() {

    private val viewModel: NoConnectionViewModel by lazy {
        ViewModelProvider( owner: this).get(
            NoConnectionViewModel::class.java
        )
    }

    private val networkNotifier = NetworkNotifier.provide(NetworkCapabilities.TRANSPO

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.e( tag: "{{looper", msg: "onAvailable ${Looper.myLooper()}")
        viewModel.showProgressBarLive.observe( owner: this) {  it: Boolean!
            viewBinding.NoConnectionProgressBar.isVisible = it
            viewBinding.NoConnectioMessageTextView.isVisible = !it
        }

        networkNotifier.register( context: this) { network, connectivity ->
            when (connectivity) {
                NetworkNotifier.Connectivity.Online ->  viewModel.ping()
            }
        }

        viewModel.nextChatActivityLive.observe( owner: this) {  it: Boolean!
            if (it == true) {
                startActivity(Intent( packageContext: this, ChatActivity::class.java))
                finish()
            }

        }

    }
```

```
class NoConnectionViewModel : ViewModel() {
    private val repository: PingRepository by inject()
    val nextChatActivityLive = MutableLiveData<Boolean>()
    val showProgressBarLive = MutableLiveData( value: false)

    fun ping() {
        showProgressBarLive.value = true

        repository.ping { body, error ->
            nextChatActivityLive.value = (error == null)

            showProgressBarLive.value = (error == null)

        }
    }
}
```

Εικόνα 4.18 NoConnectionActivity & NoConnectionViewModel

4.4.5 ChatAdapter

Η κλάση ChatAdapter περιλαμβάνει τη λογική για την αναπαράσταση των μηνυμάτων στο RecyclerView. Οι τύποι των μηνυμάτων είναι Incoming ή Outgoing και αντιπροσωπεύονται από τις μεταβλητές INCOMING_VIEW_TYPE και OUTGOING_VIEW_TYPE.

Με βάση τον τύπο του μηνύματος, αυτά παρουσιάζονται αριστερά αν είναι Incoming και δεξιά αν είναι Outgoing.

```
class ChatAdapter(                                                    ⚠1 ⚠2
    val layoutInflater: LayoutInflater,
    private var items: List<Message>
) : RecyclerView.Adapter<RecyclerView.ViewHolder>() {

    private val INCOMING_VIEW_TYPE = 0
    private val OUTGOING_VIEW_TYPE = 1

    class IncomingViewHolder(
        val viewBinding: RecyclerviewIncomingMessageBinding
    ) : RecyclerView.ViewHolder(viewBinding.root) {

        fun bind(item: Message) {
            viewBinding.messageTextView.text = item.message
        }
    }

    class OutgoingViewHolder(
        val viewBinding: RecyclerviewOutgoingMessageBinding
    ) :
        RecyclerView.ViewHolder(viewBinding.root) {

        fun bind(item: Message) {
            viewBinding.messageTextView.text = item.message
        }
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
        return when (viewType) {
            INCOMING_VIEW_TYPE -> {
                IncomingViewHolder(
                    RecyclerviewIncomingMessageBinding.inflate(
                        layoutInflater,
                        parent,
                        attachToParent: false
                    )
                )
            }
            OUTGOING_VIEW_TYPE -> {
                OutgoingViewHolder(
                    RecyclerviewOutgoingMessageBinding.inflate(
                        layoutInflater,
```

Εικόνα 4.19 Τμήμα της ChatAdapter

4.4.6 ChatActivity

Η κλάση περιλαμβάνει τη λογική για την αποστολή ενός μηνύματος αφού πατηθεί το κουμπί send και τη λογική των ρυθμίσεων όταν πατηθούν τα κουμπιά SAVE και UPDATE για την εκάστοτε ρύθμιση.

Η απάντηση για τον αν ήταν επιτυχής η ενέργειά για το εκάστοτε Command λαμβάνεται στη μέθοδο onMessage. Εκεί γίνεται έλεγχος αν το μήνυμα είναι τύπου Outgoing, δηλαδή στέλνουμε κάτι, τότε γίνεται επιπλέον έλεγχος ανάλογα με το Command ώστε να γνωρίζουμε σε ποια ρύθμιση αναφέρεται η απάντηση.

Αν η απάντηση είναι τύπου OkMessageResponse, σημαίνει πως η ενέργεια ήταν σωστή και γίνεται update στο UI, αλλιώς εμφανίζεται σύντομο μήνυμα λάθους.

Όταν το μήνυμα είναι τύπου Incoming, τότε εμφανίζεται στην οθόνη. Παρακάτω φαίνεται ένα τμήμα αυτών των ελέγχων.

```
110                } else if (it.command == Command.BAND) {
111                    if (it.response is OkMessageResponse) {
112                        if ((it.response as OkMessageResponse).value.isNullOrEmpty()) {
113                            viewBinding.editText.setText(it.text)
114                            enterInputValue = it.text
115                        } else {
116                            viewBinding.editText.setText(
117                                (it.response as OkMessageResponse).value?.replace(
118                                    oldValue: "\r\n",
119                                    newValue: ""
120                                )
121                            )
122                            enterInputValue =
123                                (it.response as OkMessageResponse).value?.replace( oldValue: "\r\n",  newValue: "")
124
125                        }
126
127
128                    } else if (it.response is ErrorMessageResponse) {
129                        viewBinding.editText.setText(enterInputValue)
130                        runOnUiThread {
131                            Toast.makeText(
132                                context: this,
133                                text: "Something Went Wrong. Please try again.",
134                                Toast.LENGTH_SHORT
135                            ).show()
136
137                        }
138                    }
```

Εικόνα 4.20 Έλεγχος για το Command BAND

Κεφάλαιο 5ο: Περιγραφή και Χρήση της Εφαρμογής

5.1 Εισαγωγή

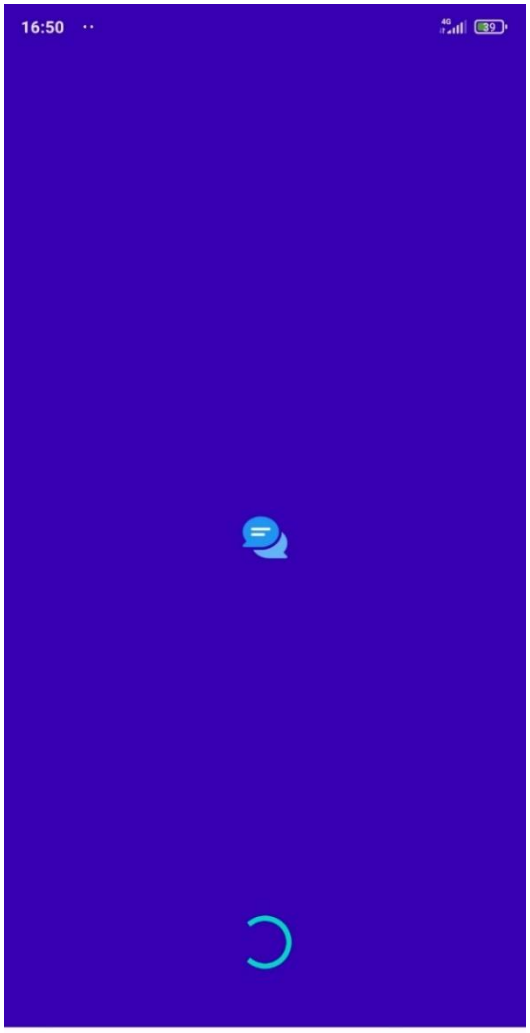
Σε αυτό το κεφάλαιο γίνεται περιγραφή των λειτουργιών της εφαρμογής βήμα-βήμα μέσω screenshots. Συγχρόνως δίνονται και περισσότερες πληροφορίες σχετικά με τα σενάρια σε περίπτωση αποσύνδεσης από το δίκτυο Wi-Fi

5.2 Οθόνη Splash

Μόλις ο χρήστης ανοίξει την εφαρμογή εμφανίζεται μία οθόνη Splash.

Σε αυτή την οθόνη φαίνεται το εικονίδιο της εφαρμογής στη μέση και ένα εικονίδιο Loading. Η οθόνη αυτή χρησιμεύει για να αποφευχθεί να ανοίξει η οθόνη του Chat αν το κινητό δεν έχει συνδεθεί στο Wi-Fi Access Point του ESP8266.

Αν ο χρήστης συνδεθεί στο Access Point τότε ανοίγει η επόμενη οθόνη για την αποστολή και λήψη μηνυμάτων, αλλιώς θα μεταφερθεί σε μία άλλη οθόνη που τον προτρέπει να συνδεθεί στο Wi-Fi Access Point για να μπορέσει να χρησιμοποιήσει κανονικά την εφαρμογή.



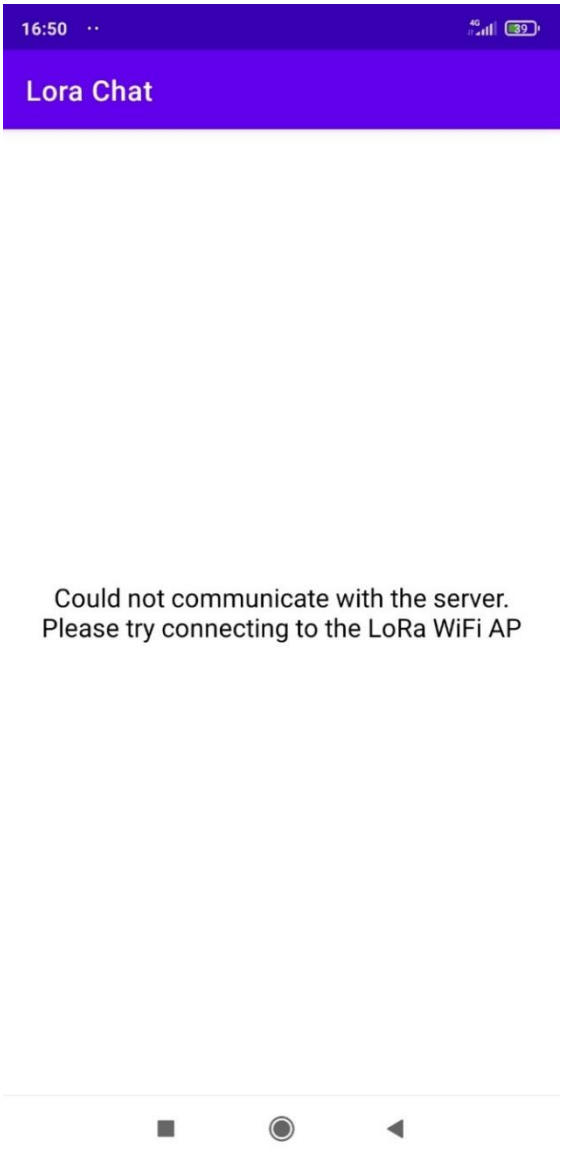
Εικόνα 5.1 Στιγμιότυπο Οθόνης Splash

5.3 Οθόνη No Connection

Σε αυτή την οθόνη ο χρήστης ενημερώνεται πως δεν είναι συνδεδεμένος στο σωστό Access Point.

Εσωτερικά γίνεται έλεγχος για το αν αλλάξει κάτι στη σύνδεση, δηλαδή συνδεθεί σε κάποιο δίκτυο Wi-Fi. Σε αυτή την περίπτωση η εφαρμογή θα προσπαθήσει εκ νέου να στείλει αίτημα στον web server που τρέχει στο ESP8266 ώστε να δει αν συνδέθηκε στο σωστό Access Point.

Αν το μήνυμα που θα λάβει είναι εκείνο που αναμένεται να στείλει ο web server, τότε γίνεται αυτόματα μεταφορά στην οθόνη του Chat.



Εικόνα 5.2 Στιγμιότυπο Οθόνης No Connection

5.4 Οθόνη Chat

Στην οθόνη Chat φαίνεται ένα άδειο πλαίσιο που προβάλλονται τα μηνύματα, ένα text box και ένα κουμπί αποστολής, ενώ πάνω δεξιά εμφανίζεται ένα εικονίδιο για την μετάβαση στις ρυθμίσεις.

Αν ο χρήστης δεν πληκτρολογήσει κείμενο τότε το κουμπί δεν εκτελεί καμία λειτουργία και εμφανίζει σύντομο μήνυμα για να γράψει κάτι, αλλιώς στέλνει το μήνυμα.

Εικόνα 5.3 Στιγμιότυπο Οθόνης Chat

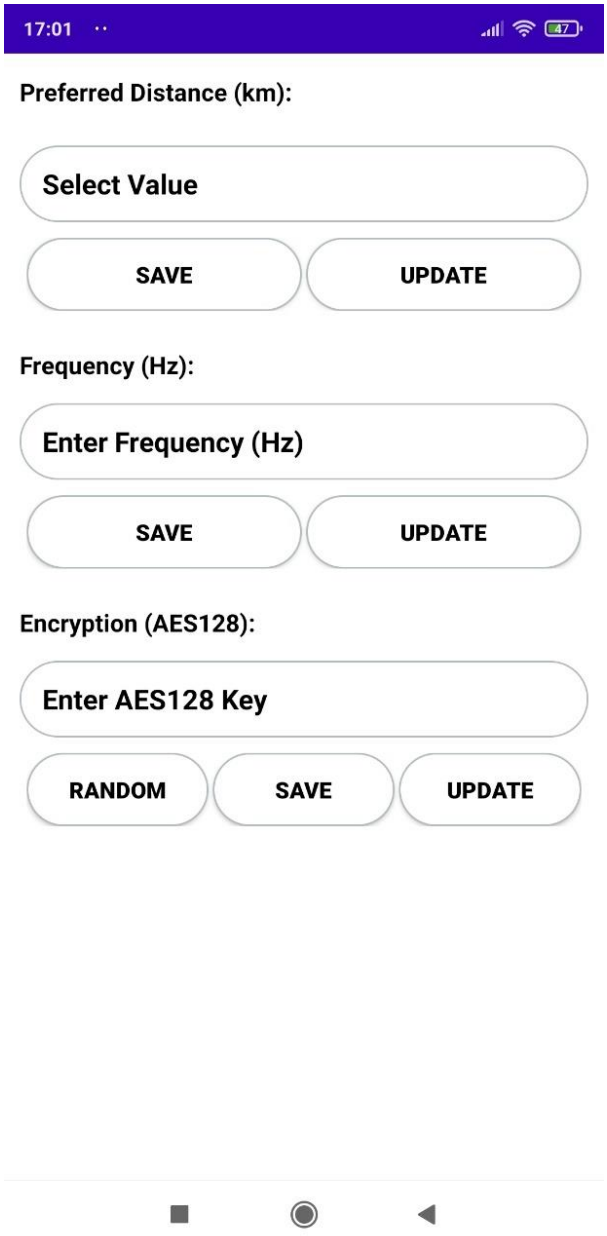
5.5 Οθόνη Ρυθμίσεων

Σε αυτή την οθόνη ο χρήστης μπορεί να αλλάξει ορισμένες ρυθμίσεις που αφορούν την απόσταση επικοινωνίας, τη συχνότητα του πομποδέκτη LoRa και να θέσει ένα κλειδί κρυπτογράφησης AES 128 bit 32 χαρακτήρων.

Κάθε επιλογή περιέχει 2 κουμπιά, το κουμπί SAVE που αποθηκεύει μία επιλογή και το UPDATE που λαμβάνει την υπάρχουσα τιμή που έχει ο πομποδέκτης. Στο dropdown για την επιλογή απόστασης υπάρχουν τρεις επιλογές, Προκαθορισμένη, Πάνω από 3 χιλιόμετρα και Κάτω από 3 χιλιόμετρα. Με την αποθήκευση της επιλογής ο χρήστης επιλέγει την απόσταση που μπορεί να απέχει από τον συνομιλητή του για την απρόσκοπτη επικοινωνία μεταξύ τους. Ωστόσο η επιλογή αυτή επηρεάζει την ταχύτητα αποστολής των δεδομένων καθώς αλλάζει το spreading factor που αναφέρθηκε στο κεφάλαιο 2. Επομένως είναι προτιμότερο αν δεν υπάρχει μεγάλη απόσταση μεταξύ των δύο, να επιλέγεται μία ενδιάμεση απόσταση από το μενού.

Στο πλαίσιο εισαγωγής της συχνότητας ο χρήστης μπορεί να βάλει μία συχνότητα στα επιτρεπτά όρια που αφορούν την ΕΕ, δηλαδή EU-863-870, παρόλο που το RYLR896 μπορεί να εκπέμψει και σε συχνότητες 915MHz. Πρέπει να σημειωθεί ότι όταν γίνεται επανεκκίνηση του ESP8266 τότε η συχνότητα επανέρχεται σε μία προκαθορισμένη τιμή των 869500000Hz για αποφυγή λαθών από τη μεριά του χρήστη.

Τέλος, στο πλαίσιο εισαγωγής κωδικού ο χρήστης μπορεί να κάνει generate με το κουμπί RANDOM έναν τυχαίο αποδεκτό κωδικό AES 128. Ο κωδικός πρέπει να είναι και από τις δύο πλευρές ίδιος για να μπορούν να αναγνωριστούν τα μηνύματα, ενώ ο κωδικός δεν είναι αναγκαίος για την ομαλή λειτουργία της εφαρμογής. Πρέπει να σημειωθεί όμως καθώς είναι σημαντικό, ότι η απόσταση, η συχνότητα και το encryption key πρέπει να είναι ίδια μεταξύ των δύο συσκευών που θα επικοινωνούν.



Εικόνα 5.4 Στιγμιότυπο Οθόνης Settings

Κεφάλαιο 6ο: Συμπεράσματα και Προτάσεις Βελτίωσης

6.1 Συμπεράσματα

Στην παρούσα εργασία μελετήθηκε το πρωτόκολλο LoRa για την ασύρματη επικοινωνία για ανταλλαγή μηνυμάτων κειμένου μεταξύ δύο συσκευών Android. Για την επίτευξη του σκοπού αυτού έγινε έρευνα για τον κατάλληλο εξοπλισμό ενώ σχεδιάστηκε και υλοποιήθηκε σε τρία τμήματα για την επαναχρησιμότητά του.

Αναλύθηκαν λεπτομερώς οι επιμέρους διαδικασίες που ακολουθήθηκαν για την τελική υλοποίησή της και δόθηκαν οδηγίες χρήσης μέσω screenshots.

Κατά τη διαδικασία δημιουργίας της πτυχιακής εργασίας αποκτήθηκαν γνώσεις όσον αφορά το πρωτόκολλο LoRa και πώς αυτό λειτουργεί, προγραμματισμό σε Arduino, την καλύτερη κατανόηση των socket και μία πρώτη εισαγωγή στο Android.

6.2 Προτάσεις Βελτίωσης

Η εφαρμογή, αν και σε τελικό στάδιο έχει περιθώρια βελτίωσης προσθέτοντας κάποιες επιπλέον δυνατότητες. Μία από αυτές θα μπορούσε να είναι η προσθήκη μηχανισμού ειδοποιήσεων για τη λήψη ενός μηνύματος, όπου ο χρήστης θα μπορεί να λαμβάνει ειδοποίηση αν έχει ανοιχτή την εφαρμογή στο παρασκήνιο. Είναι δυνατόν ακόμα να γίνει προσθήκη επιπλέον επιλογών στις ρυθμίσεις για τη διαχείριση των ρυθμίσεων του πομποδέκτη πέρα από τις τρεις που δίνονται τώρα. Με αυτόν τον τρόπο ο χρήστης της εφαρμογής θα μπορεί να παραμετροποιεί όπως θέλει τις ρυθμίσεις και να επιτυγχάνει μεγαλύτερες αποστάσεις επικοινωνίας και όχι μόνο. Μία σημαντική επέκταση θα μπορούσε να είναι η δημιουργία ομάδων – group chats αφού ο πομποδέκτης RYLR896 παρέχει τις κατάλληλες ρυθμίσεις και δυνατότητες για αυτόν τον σκοπό. Τέλος μπορεί να γίνει επέκταση δίνοντας τη δυνατότητα εναλλαγής σε ρυθμίσεις για έμπειρους χρήστες όπου ο χρήστης θα μπορεί να στέλνει απευθείας εντολές στον πομποδέκτη για την παραμετροποίησή του.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Βιβλία

[1] Programming Android with Kotlin - Achieving Structured Concurrency with Coroutines, Pierre-Olivier Laurence and Amanda Hinchman-Dominguez, with G. Blake Meike and Mike Dunn

[2] NodeMCU ESP8266 Communication Methods and Protocols : Programming with Arduino IDE ,Manoj R. Thakur

[3] Book on ESP8266, Neil Kolban

[4] Inside the AndroidTM OS Building, Customizing, Managing and Operating Android System Services G. Blake Meike Larry Schiefer

[5] Programming Android with Kotlin. Achieving Structured Concurrency with Coroutines, Pierre-Olivier Laurence and Amanda Hinchman-Dominguez, with G. Blake Meike and Mike Dunn

[6] LoRa (Long-Range) High-Density Sensors for Internet of Things, Alexandru Lavric

[7] A Study of LoRa: Long Range & Low Power Networks for the Internet of Things, Aloÿs Augustin, Jiazi Yi, Thomas Clausen and William Mark Townsley

Internet Site

[8] https://kotlinlang.org/docs/faq.html

[9] https://www.arduino.cc/en/Guide/Introduction

[10] https://tttapa.github.io/ESP8266/Chap01%20-%20ESP8266.html

[11] https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/

[12] https://reyax.com/wp-content/uploads/2020/01/Lora-AT-Command-RYLR40x_RYLR89x_EN.pdf

[13] https://www.seeedstudio.com/blog/2021/01/20/at-commands-what-why-how/

[14] https://reyax.com/tw/wp-content/uploads/2019/12/RYLR896_EN.pdf

[15] https://developer.android.com/kotlin/first/

[16] https://developer.android.com/studio/intro/

[17] https://www.androidauthority.com/history-android-os-name-789433/

[18] https://developer.android.com/jetpack

[19] https://developer-mozilla-org.translate.goog/en-US/docs/Web/HTTP/Overview

[20] https://tttapa.github.io/ESP8266/Chap05%20-%20Network%20Protocols.html

[21] https://github.com/me-no-dev/ESPAsyncWebServer#espasyncwebserver/

[22] https://github.com/Links2004/arduinoWebSockets/

[23] https://square.github.io/retrofit/

[24] https://github.com/TooTallNate/Java-WebSocket/

[25] https://www.postman.com/product/what-is-postman/

[26] https://insert-koin.io/docs/reference/introduction

ΠΑΡΑΡΤΗΜΑ Α : Software Source Code

1 Κώδικας του Arduino

1.1 loraChatFinal.ino

```
#ifdef ESP8266
#include <ESP8266WiFi.h>
#elif defined(ESP32)
#include <WiFi.h>
#else
#error "Board not found"
#endif

#include <WebSocketsServer.h>
#include <ESPAsyncWebServer.h>
#include <SoftwareSerial.h>                    //we have to include the SoftwareSerial library, or else we can't use it

#define rx 13                                  //LORA TX
#define tx 15                                  //LORA RX

SoftwareSerial loraSerial(rx, tx);             //define how the soft serial port is going to work

#ifndef APSSID
#define APSSID "LoRaChat 2"
#define APPSK  "ARandomPasswordHere"
#endif

#define USE_SERIAL Serial1

const char *ssid = APSSID;
const char *password = APPSK;
```

```
AsyncWebServer server(80);
WebSocketsServer websockets(81);

void initWifi(){
   Serial.println("Configuring access point...");
  /* You can remove the password parameter if you want the AP to be open. NOT recommended. */
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, password);

  IPAddress myIP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(myIP);

  }

void notFound(AsyncWebServerRequest *request)
{
  request->send(404, "text/plain", "Page Not found");
}

void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length) {

   switch(type) {
      case WStype_DISCONNECTED:
         USE_SERIAL.printf("[%u] Disconnected!\n", num);
         break;
      case WStype_CONNECTED:
         {
            IPAddress ip = websockets.remoteIP(num);
            USE_SERIAL.printf("[%u] Connected from %d.%d.%d.%d url: %s\n", num, ip[0], ip[1], ip[2], ip[3], payload);
```

```
            // send message to client
            //websockets.sendTXT(num, "Welcome");
          }
          break;
      case WStype_TEXT: {
          USE_SERIAL.printf("[%u] get Text: %s\n", num, payload);

          String payload_str = String((char*) payload);

          Serial.println(payload_str); //debug

          if(payload_str.length()>=2){ //2 is the minimum command rylr length (AT)
            loraSerial.println(payload_str);
           }

          break;
          }
      case WStype_BIN:
          USE_SERIAL.printf("[%u] get binary length: %u\n", num, length);
          hexdump(payload, length);

          // send message to client
          // webSocket.sendBIN(num, payload, length);
          break;
    }

}

void setup() {

  Serial.begin(115200);
  initWifi();
```

```
  delay(250);
  loraSerial.begin(38400);
  delay(250);
  loraSerial.println("AT+BAND=869500000");

  server.onNotFound(notFound);

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
       request->send(200, "application/json", "{\"message\": \"esp\"}");
    });

  websockets.enableHeartbeat(5000, 3000, 2);

  server.begin();
  websockets.begin();

  websockets.onEvent(webSocketEvent);

}

void loop() {
websockets.loop();

//Used for debugging
if (Serial.available() > 0) {

  String input = Serial.readString();

  Serial.print(" I received: ");

  Serial.println(input);
  websockets.broadcastTXT(input);
}
```

```
//end
 if(loraSerial.available() > 0){ //if I have data from lora
  String input = loraSerial.readString();

  websockets.broadcastTXT(input); //send data to the socket client - phone
 }

}
```

2 Κώδικας του Kotlin Module

2.1 Outgoing.kt

```
interface Message

class Outgoing(
    val command: Command,
    val type: MessageType,
    val text: String?= null
): Message {

    var response: MessageResponse?= null

    fun setOk(value: String?= null): Message{
        response = OkMessageResponse(value)
        return this
    }

    fun setError(value: Int): Message{
        response = ErrorMessageResponse(value)
        return this
    }

    override fun toString(): String {
```

```
        return when(type){
            MessageType.Question-> "AT${command.value}?"
            MessageType.Send-> "AT${command.value}=0,${text?.length},$text"
            MessageType.Setter-> "AT${command.value}=$text"
            else-> throw Exception("Error")
        }
    }

}
```

2.2 Incoming.kt

```
class Incoming(
    val content: String
): Message {

    override fun toString(): String {
        return content
    }
}
```

2.3 MessageType.kt

```
enum class MessageType {
    Question,
    Setter,
    Send;
}
```

2.4 Command.kt

```
enum class Command(
    val value: String
){
    PING        (""),
    MODE        ("+MODE"),
```

```
    BAUD_RATE   ("+IPR"),
    PARAMETER   ("+PARAMETER"),
    BAND        ("+BAND"),
    SEND        ("+SEND"),
    CPIN        ("+CPIN")
}
```

2.5 OkMessageResponse.kt

```
interface MessageResponse

class OkMessageResponse(
    val value: String? = null
) : MessageResponse
{
    override fun toString(): String {
        return value?:""
    }
}
```

2.6 ErrorMessageResponse.kt

```
class ErrorMessageResponse(
    val error: Int
): MessageResponse
```

2.7 MessageFactory.kt

```
object MessageFactory {

    fun ofQuestion(command: Command): Outgoing{
        return Outgoing(command, MessageType.Question)
    }

    fun ofMessage(text: String): Outgoing{
        return Outgoing(Command.SEND, MessageType.Send, text)
```

```
    }

    fun ofValue(command: Command, value: String): Outgoing{
        return Outgoing(command, MessageType.Setter, value)
    }

    fun ofIncoming(content: String): Incoming{

        return Incoming(content)
    }
}
```

2.8 SocketServiceImpl

```
class SocketManagerImpl : SocketManager {

    private val socketService = WebSocketService.instance()
    private val debounceAttempts = AtomicInteger(0)

    init {
        socketService.onMessage(::handleMessage)
    }

    private var message: Outgoing? = null

    private val messagesQueue = mutableListOf<Outgoing>()

    override fun sendQuestion(command: Command) {
        outgoingMessage(
            MessageFactory.ofQuestion(command)
        )
    }

    override fun sendValue(command: Command, value: String) {
```

```
    outgoingMessage(
        MessageFactory.ofValue(command, value)
    )
}

override fun sendMessage(text: String) {
    outgoingMessage(
        MessageFactory.ofMessage(text)
    )
}

private fun outgoingMessage(newMessage: Outgoing) {
    synchronized(this) {
        messagesQueue.add(newMessage)
    }
    sendNextMessageIfExist()
}

override fun connect() {
    socketService.connect()
}

private fun send(outgoing: Outgoing) {
    this.message = outgoing
    socketService.send(outgoing.toString())
}

private var onMessageCallback: ((Message) -> Unit)? = null
override fun onMessage(callback: (Message) -> Unit) {
    onMessageCallback = callback
}

private fun handleMessage(content: String) {
```

```
        val msg: Message = when {
            content.startsWith("+RCV", true) -> {
                val messageContent = content.replace("+RCV=", "").split(",").getOrNull(2) ?: "Error - Unreadable Message"
                MessageFactory.ofIncoming(messageContent)
            }
            content.startsWith("+OK") -> {
                this.message!!.setOk()
            }
            content.startsWith("+ERR") -> {
                this.message!!.setError(content.trim().replace("+ERR=", "").toIntOrNull()?:-1)
            }
            else -> {
                val command = this.message!!.command
                this.message!!.setOk(content.replace("${command.value}=", ""))
            }
        }

        notify(msg)
    }

    private fun notify(message: Message) {
        onMessageCallback?.invoke(message)

        if (message is Outgoing) {
            synchronized(this) {
                this.message = null
            }
            sendNextMessageIfExist()
        }

    }
```

```
    private fun sendNextMessageIfExist() {
        if (!socketService.isOpen()) {

            debounceAttempts.incrementAndGet()
            if (debounceAttempts.get() >= 3) {
                debounceAttempts.set(0)
                return
            }

            try {
                Thread.sleep(200L + (200 * debounceAttempts.get()))
            } catch (e: Exception) {

            } finally {
                sendNextMessageIfExist()
            }

        }

        debounceAttempts.set(0)

        synchronized(this) {
            if (message == null) {
                messagesQueue.removeFirstOrNull()?.let {
                    send(it)
                }
            }
        }
    }
}
```

3 Κώδικας Android

3.1 SplashActivity.kt

```
class SplashActivity : BaseActivity<ActivitySplashBinding>() {

    override fun getViewBindingClass(): Class<ActivitySplashBinding> {
        return ActivitySplashBinding::class.java
    }

    private val viewModel: SplashViewModel by lazy { ViewModelProvider(this).get(SplashViewModel::class.java) }

    override fun onCreate(savedInstanceState: Bundle?) {
        Log.e("{{", "${this}")
        Log.e("{{", "${viewBinding}")
        super.onCreate(savedInstanceState)
        Log.e("{{", "${viewModel}")

        viewModel.toMainLive.observe(this) {
            viewBinding.imageView.isVisible = it
//            Toast.makeText(this, "Go To Next $it", Toast.LENGTH_SHORT).show()
        }

        viewModel.stateLive.observe(this) {

            when (it) {
                is SuccessfulLoginState -> {
                    startActivity(Intent(this, ChatActivity::class.java))

                    finish()
                }
                is UnsuccessfulLoginState -> {
                    startActivity(Intent(this, NoConnectionActivity::class.java))

                    finish()
```

```
            }
        }
    }

}
```

3.2 SplashViewModel.kt

```
class SplashViewModel : ViewModel() {

    private val repository: PingRepository by inject()

    val toMainLive = MutableLiveData<Boolean>()
    val stateLive = MutableLiveData<State>()

    init {
        repository.ping { body, error ->
            if (error == null) {
                stateLive.value = SuccessfulLoginState()
            } else {
                stateLive.value = UnsuccessfulLoginState()
            }
        }


    }

}

interface State

class UnsuccessfulLoginState : State
```

```
class SuccessfulLoginState : State
```

3.3 NoConnectionActivity.kt

```
class NoConnectionActivity : BaseActivity<ActivityNoConnectionBinding>() {

    private val viewModel: NoConnectionViewModel by lazy {
        ViewModelProvider(this).get(
            NoConnectionViewModel::class.java
        )
    }

    private val networkNotifier = NetworkNotifier.provide(NetworkCapabilities.TRANSPORT_WIFI)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.e("{{looper", "onAvailable ${Looper.myLooper()}")
        viewModel.showProgressBarLive.observe(this) {
            viewBinding.NoConnectionProgressBar.isVisible = it
            viewBinding.NoConnectioMessageTextView.isVisible = !it
        }

        networkNotifier.register(this) { network, connectivity ->
            when (connectivity) {
                NetworkNotifier.Connectivity.Online ->  viewModel.ping()
            }
        }

        viewModel.nextChatActivityLive.observe(this) {
            if (it == true) {
                startActivity(Intent(this, ChatActivity::class.java))
                finish()
            }
```

```
        }

    }

    override fun onDestroy() {
        super.onDestroy()
        networkNotifier.unregister(this)
    }

    override fun getViewBindingClass(): Class<ActivityNoConnectionBinding> {
        return ActivityNoConnectionBinding::class.java
    }
}
```

3.4 NoConnectionViewModel.kt

```
class NoConnectionViewModel : ViewModel() {
    private val repository: PingRepository by inject()
    val nextChatActivityLive = MutableLiveData<Boolean>()
    val showProgressBarLive = MutableLiveData(false)

    fun ping() {
        showProgressBarLive.value = true

        repository.ping { body, error ->
            nextChatActivityLive.value = (error == null)

            showProgressBarLive.value = (error == null)

        }
    }
}
```