

УВОД

Автоматизацията на дома е задача с която хората се занимават още от времената преди съществуването на компютъра. Това е една от най-големите индустрии на технологичния свят. Всеки от нас си мечтае за бъдеще в което дома ще е продължение на самите нас. Огромни компании като Google и Amazon инвестират милиарди за разработката на нови продукти и технологии свързани с нея. Техните виждания за бъдещето на дома са той да се превърне в една жива система. Устройствата в него да работят като едно с такъв финес, че да са почти незабележими за човека. Да улесняват живота на всеки, независимо дали прекарват цял ден или само няколко часа в него. Именно това прави очакванията към проекти свързани с автоматизацията на дома.

Тази дипломна работа няма чак толкова мащабни цели колкото гигантите на технологичната индустрия. Няма за цел да се конкурира с подобни продукти на пазара. С тази дипломна работа имам целта да направя продукт, който най-вече да е полезен за моя лична употреба. Но все пак и дам един солиден старт на система с много потенциал за бъдещо развитие. Като за начало главната функционалност на проекта е просто да включва и изключва устройства свързани към контролната система посредством Android приложение. Едно от най-важните части от развитието на един проект е поставянето на цели. Но не само безразборно поставяне на цели. Трябва да се поставят постижими цели спрямо възможностите и времето с което разполагаш, за да не се демотивираш и чудиш как ще свършиш огромното количество работа за малко време, което имаш, и най-важното да се учиш и наслаждаваш не само на финалния продукт, но и от всеки момент в които работиш по него. Главната причина да избира тази тема за дипломната работа е да изляза от комфортната си зона. Правил съм само един embedded проект до сега, а Android приложение - никога. Разбира се, че да избира за тема на дипломната работа сфера с която съм се занимавал преди е доста по-умно и сигурно решение, но мисля че този проект е добра предпоставка да науча нови технологии с които не съм се занимавал преди, особено като се има предвид, че те са едни от най-популярните на пазара в момента каквато е Android например. С това в предвид, очаква по-голяма част от времето за разбратака да е мине в изучаване на технологии отколкото в реална

работа по продукта, но се надявам това да си струва в крайна сметка.

ПЪРВА ГЛАВА

МЕТОДИ, СРЕДСТВА И ТЕХНОЛОГИИ ЗА ДИСТАНЦИОННО КОНТРОЛИРАНЕ НА УСТРОЙСТВА ОТ ДОМА И РАЗРАБОТКА НА ANDROID ПРИЛОЖЕНИЕ - УПРАВЛЯВАЩ СОФТУЕР

1.1. Елементна база за изработка на система за контролиране на устройства от дома

Задачата да се контролира прост уред от дома като лампа, печка и врата дистанционно може да се реши по много начини. За поставената задача, аз избрах да използвам микроконтролер с Wi-Fi възможности. След дълъг проучвателен процес, конкретния микроконтролер над който се спрях е **Esp8266**, използващ Esp8266 микрочипа за свързване с Wi-Fi. Този микроконтролер е евтин и лесно програмируем, а Esp8266 микрочипа има богата документация, предостатъчно библиотеки за работа с Wi-Fi, както и много онлайн ресурси които бяха доста полезни при проучването и при решаването на проблеми при разработката на системата за дистанционно контролиране на устройства от дома. На *Фигура 1.1.* показва характеристиките на избрания микроконтролер. От тях се вижда, че микроконтролера работи на 7-12 волта. За да може да работи с устройства на по-високи напрежение, комбинираме микроконтролера с **реле**. Простата конфигурация от микроконтролера и релето е сърцето на системата за контрол. Позволява чрез прости единични команди към микроконтролера да се контролират устройствата - включване/изключване на лампа, отваряне/затваряне на врата, включване/изключване на печка. Освен това за направата на устройството са нужни: захранващ модул, регулатор на напрежение, кондензатори, транзистор, клеми, диод, светодиоди и цокъл. Подробно описание на елементната база в глава шеста.

(фигура 1.1)

- Operating Voltage: 3.3V
- Input Voltage: 7-12V
- Digital I/O Pins (DIO): 16
- Analog Input Pins (ADC): 1
- UARTs: 1
- SPIs: 1
- I2Cs: 1
- Flash Memory: 4 MB
- SRAM: 64 KB
- Clock Speed: 80 MHz
- USB-TTL based on CP2102 is included onboard, Enabling Plug n Play
- PCB Antenna
- Small Sized module to fit smartly inside your IoT projects

1.2. Основни принципи, методи, технологии и развойни среди за разработка на Android приложения.

Android е една от най-популярните операционни системи в света. Въпреки, че нямах предишен опит с него, изборът да направя управляващия софтуер на Android беше доста лесен. Огромната му популярност означава, че онлайн ресурсите са изключително богати, което прави решаването на дори и на най-редките проблеми в пъти по-бързо, отколкото ако трябваше да ги се справям с тях сам.

Големият избор, който трябва да се направи при разработката на Android приложение е този на какъв език да се напише "backend-a" / програмната част. Двата най-популярни и единствените езици между които се колебаех са **Kotlin** и **Java**. Тези два езика напълно доминират маркета на Android и да пиша първото си Android приложение на нещо различно от тях би било много лоша идея. В таб. 1.1 и 1.2 показвам основните предимства и недостатъци на двата езика. Какво точно ме накара да избера **Kotlin** за езика на Android приложението - управляващ софтуер обяснявам по-подробно в **глава 2.2.2**.

(Таб. 1.1 - Предимства и недостатъци на Java)

Предимства	Недостатъци
Работил съм с него преди.	Неясно бъдеще за съвместимост с Android
Най-популярен за Android	Бавен
Гигантска общност от разработчици на него	Труден за дебъгване
Безкрайно много онлайн ресурси.	

(Таб. 1.2 - Предимства и недостатъци на Kotlin)

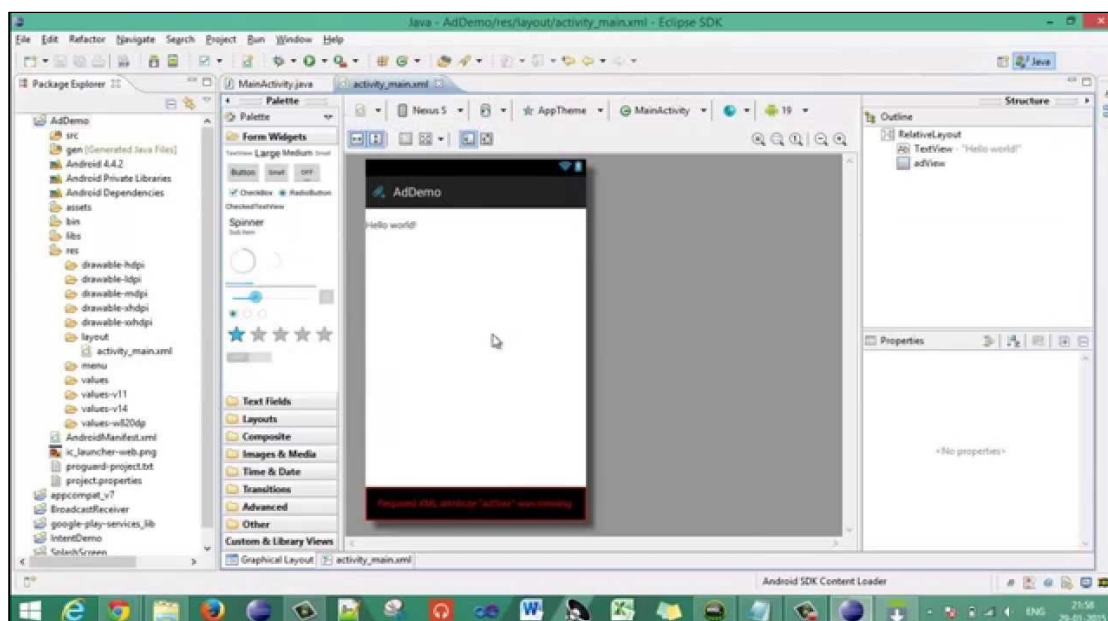
Предимства	Недостатъци
Лесен за научаване	Никога не съм го ползвал
Подобен синтаксис с Java	По-бавно компилиране
Пълна съвместимост с Java	По-малък брой разработчици работят с него.
По-компактен код.	Значително по-малко онлайн ресурси.
Сигурно бъдеще с Android	

Следващият избор, който трябваше да направя макар и не толкова важен, съвсем не без значение е този на развойна среда. Като най-популярната операционна система в света, разработчиците на Android имат пребогат избор на такива. В следващите таблици **1.3 - 1.6** ще разгледам предимствата и недостатъците на 4 от най-популярните развойни среди за Android - **Eclipse, Visual Studio Code, IntelliJ IDEA** и **Android Studio**. С помощта на тези предимства и недостатъци взех решението да използвам **Android Studio** като главна развойна част на софтуерната част - "управляващия софтуер" на дипломната работа. По-подробно обяснение защо направих този, не прекалено труден избор, може да видите в глава **2.2.2**.

(Таб. 1.3 - Предимства и недостатъци на Eclipse)

Предимства	Недостатъци
Популярен за Java	Никога не съм го ползвал
Надежден	Труден сетъп Android
Бърз Boot-up	Без вграден емулатор
Ниско потребление на памет	Остарял интерфейс

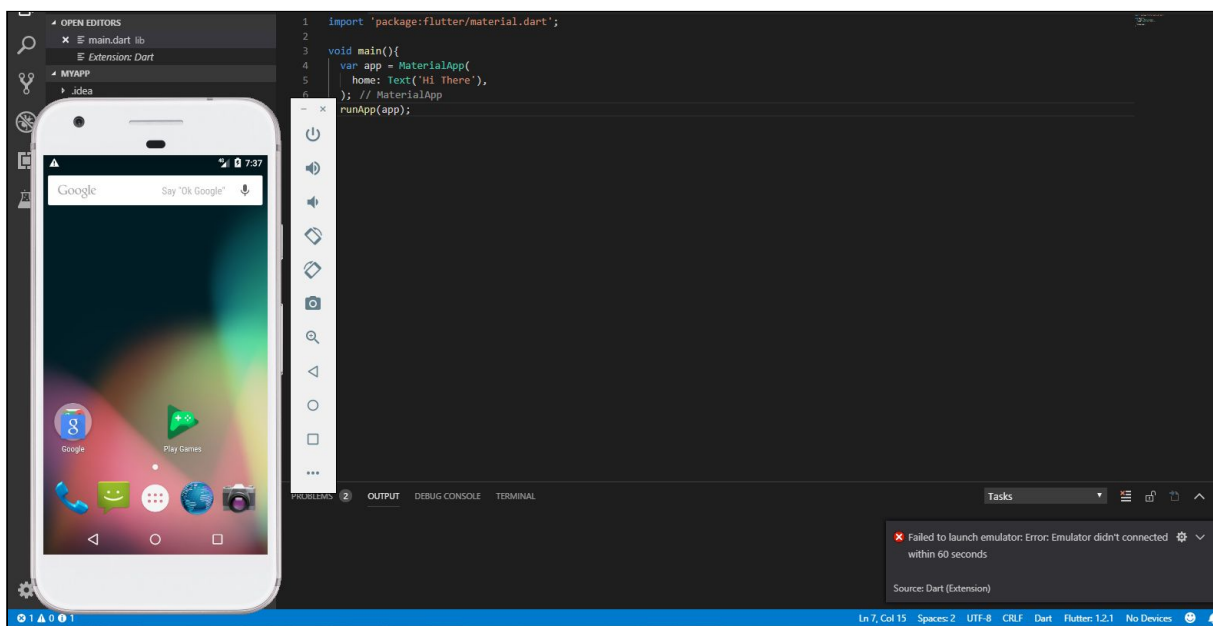
(Фигура 1.2 - Eclipse интерфейс за Android UI development)



(Таб. 1.4 - Предимства и недостатъци на Visual Studio Code)

Предимства	Недостатъци
Безплатен	Не съм го ползвал преди
Бърз	Труден setup за работа с Android
Персонализируем	Не предназначен за Kotlin/Java
	Без вграден емулатор

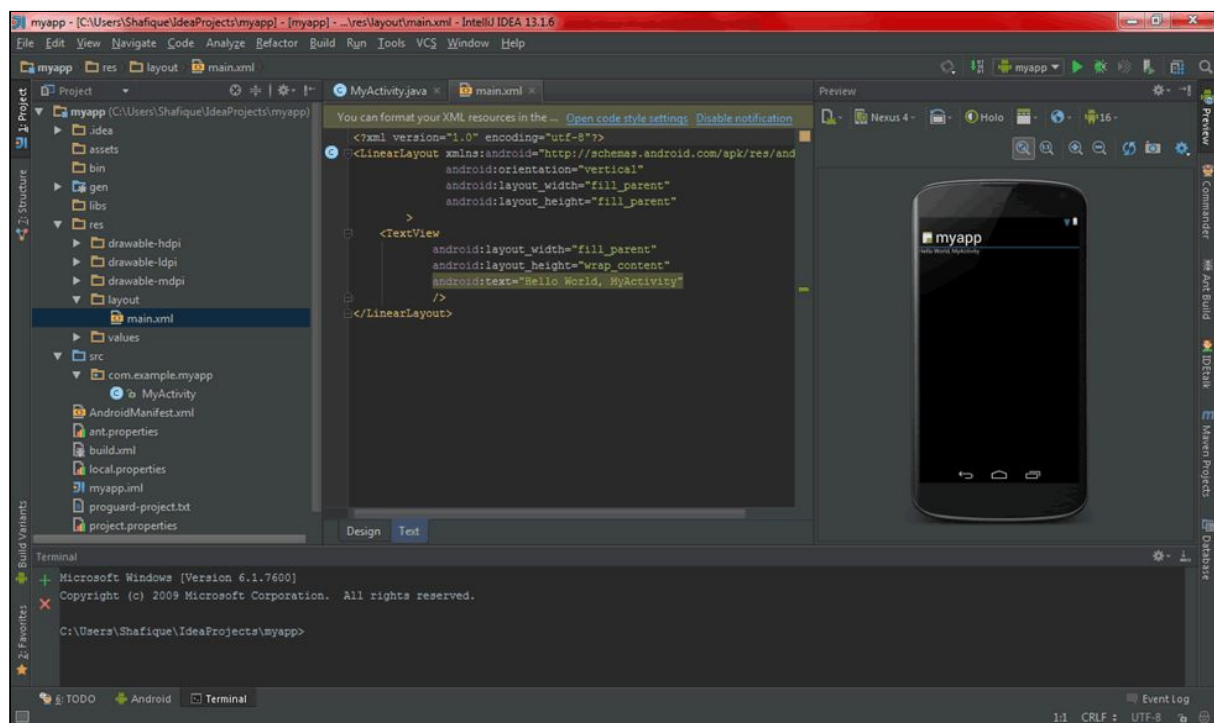
(Фигура 1.3 - Visual Studio Code интерфейс за Android UI development)



(Таб. 1.5 - Предимства и недостатъци на IntelliJ IDEA)

Предимства	Недостатъци
Официален съпорт от Google	Тежък от към памет
Лесен setup за работа с Android	Бавно зареждане
Ползвал съм го преди	Дълга компилация при първо пускане
Лесна интеграция с github	

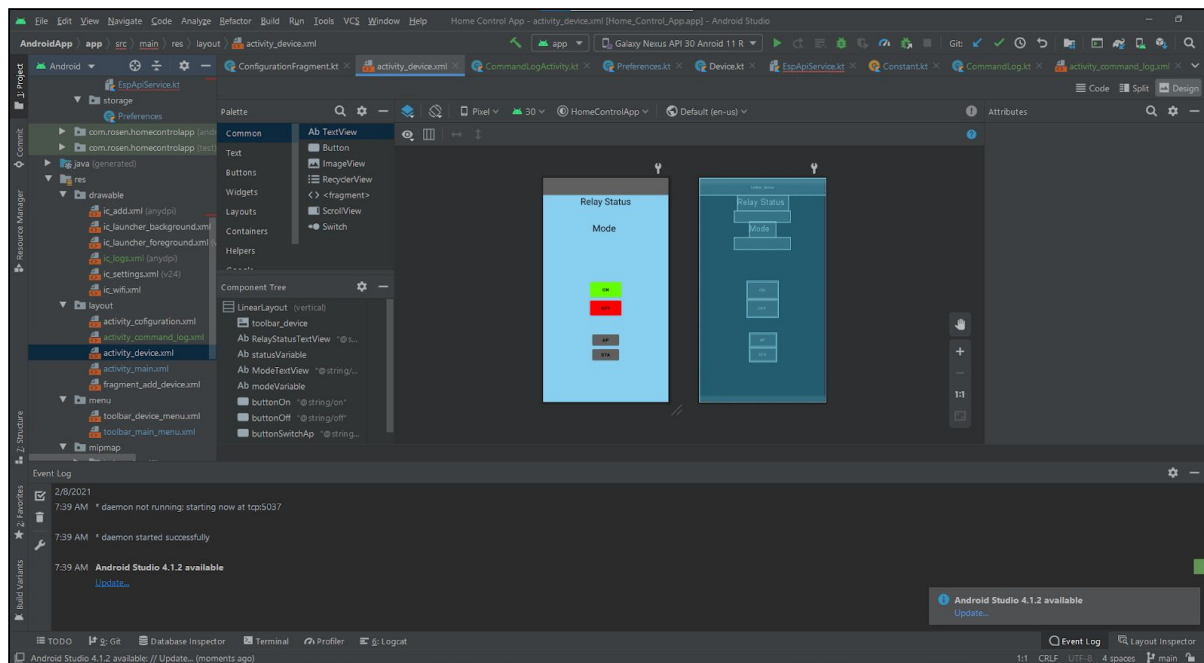
(Фигура 1.4 - IntelliJ IDEA интерфейс за Android UI development)



(Таб. 1.6 - Предимства и недостатъци на Android Studio)

Предимства	Недостатъци
Официален съпорт от Google	Тежък от към памет
Супер лесен setup за работа с Android	Бавно зареждане
Направен е за работа с Android - много добър UI за работа с Android	Дълга компилация при първо пускане
Вграден емулатор	
Безплатни ресурси (икони, логота и др.)	
Лесна интеграция с github	

(Фигура 1.5 - Android Studio интерфейс за Android UI development)



1.3. Съществуващи решения и реализации.

Home Automation-а е една много **добре** популярна и добре развита част от разработката. Почти всички големи технологични компании имат опити да навлязат в него. В тази последна част от първа глава, ще разгледаме такива устройства, за да взема вдъхновение от тях. Както казах и в увода, разбира се не може да се очаква моят продукт да се доближи до качеството и функционалността на тези на мултимилиардерите компании, но винаги можеш да научиш нещо от най-големите, макар и те да разполагат с безкрайно повече човешки и паричен ресурс.

1.3.1. Amazon Echo и Google Home

Сериите от Smartspeaker-и **Amazon Echo** и **Google Home** са хитови продукти - домашни асистенти. Тези устройствата се свързват с гласово-контролирани персонални асистент, съответно "Alexa" и "Google". Функциите на системите включват:

- търсене на мрежата за най-точно отговаряне на гласово-зададени въпроси от всякакво естество
- задаване на аларми
- пускане на музика

- контролиране на мобилини или smart tv приложение с гласови команди
- контролиране на други smart устройства в дом
- много други

(Фигура 1.6 - Amazon Echo)



(Фигура 1.7 - Google Home)



1.3.2. Smart Outlet / Light Switch

На пазара, разбира се, има и продукти, които са сравнително близки с крайната цел на тази дипломна работа. Smart ключовете за осветление дават подобни възможности, които се стремя да дам и на моето устройство - контрол над осветление чрез приложение на телефона

(фигура 1.8 - пример за Smart Light Switch)



ВТОРА ГЛАВА

ПРОЕКТИРАНЕ НА БЛЮКОВАТА СХЕМА НА СИСТЕМА ЗА ДИСТАНЦИОННО КОНТРОЛИРАНЕ НА УСТРОЙСТВА ОТ ДОМА

2.1. Функционални изисквания към микроконтролерната система за контролиране на устройства от дома

В началото на срока за извършване на дипломната работа, към нея бяха поставени следните функционални изисквания:

1. Дизайн и принципна електрическа схема на микроконтролери, управляващи поне печка, лампа и врата
2. Работа в самостоятелен и мрежов режим
3. Android приложение за управление
4. Поддръжка на известия
5. Регистриране на множество устройства
6. Запазване на историята на устройствата в timeline.

2.2. Съображения за избор на програмни средства и развойната среда

2.2.1. ESP8266 Микроконтролер

Изборът на език и развойна среда за ESP8266, беше едно от най-лесните решения в цялата разработка на дипломната работа. Както стана ясно от увода, това е едва втория ми не напълно софтуерен проект и ако трябва да съм честен, дори не съм разглеждал други възможности от тези които ползвах и първия път - развойната среда на **Arduino Ide** и стандартния за ESP-базираните микроконтролери, език **C++**. Не бих казал, че съжалявам за не особено обмисленото решение. Проекта от към embedded часта е достатъчно прост, за да не видя особени проблеми с Arduino IDE, освен това че понякога качването на кода се случваше по-бавно отколкото ми се искаше, но това отдавам по-скоро на микроконтролера, отколкото на развойната среда. Липсата на autocomplete не беше прекалено осезаема, поради малкия обем код, но виждам как може да е проблем при разширяване на мащаба на проекта. Качването на нужните библиотеки за работа с конкретния ESP8266, беше изключително лесно и като цяло съм доволен от Arduino IDE, макар и да е доста по-прост компилатор от тези с които съм свикнал да работя по принцип. C++ също не създаде проблеми. Добре познавам езика, въпреки че не съм работил с него от доста време, но Embedded часта на този проект не изисква използването на пълния обектно-ориентиран потенциал на езика, защото по-голямата част от кода е ползване на функции от изключително полезните ESP8266 библиотеки за Wi-Fi.

2.2.2. Android приложение

Както загатнах в първа глава, изборът за развойна среда и особено език за Android приложението - управляващ софтуер, далеч не беше лесен. След дълго обмисляне комбинацията над която се спрях е може би тази на бъдещето - официалната развойна среда на Google за Android - **Android Studio**. Това определено беше правилният избор. Самото име загатва, че фокуса е върху разработка на приложения точно за Android устройства и това си личи от момента в който отвориш тази развойна среда. Отне ми не повече от едно-две онлайн обучителни видия, не само за да вкарам Android Studio в готовност да компилира код, но дори да направя пълната инсталация на супер полезния вграден емулатор, идващ с него. Android Studio прави мениджирането на иначе, доста объркващите за мен пакети в един Java или Kotlin проект интуитивно. Интерфейса беше лесен за научаване, въпреки че ако нямах опит с другия продукт на разработчика JetBrains - IntelliJ IDEA, вероятно щеше да ми е по-трудно. Auto-complete-а беше задължителен за софтуерната част на проекта, не само защото ускорява процеса на писане, а и защото помогна стабилно с **отвиването** с Java синтаксиса и свикването с новият за мен Kotlin синтаксис, колкото и да са близки те. Качването на приложението на **личния ми** мобилен телефон става за секунди, без сложни маневри и главоболия. Drag and Drop опцията при дизайн на интерфейса на приложението ми помогна при дизайна на Constraint Layout-и, особено като се има предвид че го правя за пръв път, а това далеч не е просто без тази функция.

В началото на срока, когато започнах да мисля как теоретично ще се осъществи проекта, въобще не мислех че ще избягам от комфортната ми Java. Не знаех абсолютно нищо за **Kotlin** преди Октомври на 2020. След доста проучване, разбрах за предстоящ съдебен процес между собственика на Android - Google и този на Java - Oracle. През 2019, Google обявиха че Kotlin е новият им предпочитан език за Android. Разбирам, че Google нямат нито възможност, нито интерес от това да спрат внезапно поддръжката на Java-базирани приложения на Android, просто защото това са сигурно над 70% от всички. В дългосрочен план, обаче Kotlin изглежда като бъдещето на Android. Въпреки че, ако бях написал приложението на Java, това щеше да е съвсем приемливо, както казах и в увода, с тази дипломна работа не целя толкова да направя най-великото нещо на света, колкото да се вкарам в не толкова комфортни ситуации, които да ми помогнат да науча много повече.

Научих Kotlin на приемливо ниво за около седмица. Личи си че езика е до голяма степен базиран на Java, но разликите са достатъчно, за да ме накарат да си блъскам главата в стената при невиджани преди проблеми. Kotlin е съвместим с повечето библиотеки на Java, така че въобще не усетих това че езика е сравнително нов. Честно казано и до последния момент на писане на код, някои детайли между двата езика са ми като каша в главата, но това е нормално за толкова бързо научаване на каквото и да е. След **обилно** използване на Kotlin, мога да кажа че той въобще не отстъпва на Java. Даже бих казал, че е новият ми любим език. Не знам дали е заради самият език или нещо друго, но определено правя по-малко грешки и губя много по-малко време в дебъгване, което обикновено е в пъти повече от времето за писане на нов код. Проблемите които срещнах с Kotlin са подобни на тези от Java. Съобщенията за грешка изглеждат подробни, но 80% от тях са напълно безполезна информация, а останалите 20%, често дават прекалено оскъдна информация.

2.3. Проектиране на блоковата схема на система за контролиране на устройства от дома

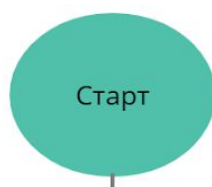
В тази част от втора глава ще покажа блокови схеми на някои от различните алгоритми в системата. Тяхната главна цел е да покаже коя част от процесите се извършват от микроконтролера и коя от Android приложението - управляващ софтуер.

За целта на лесно изобразяване на точно това, в блоковите схеми използвам следната стандартизирана цветово-кодирана **легенда** :

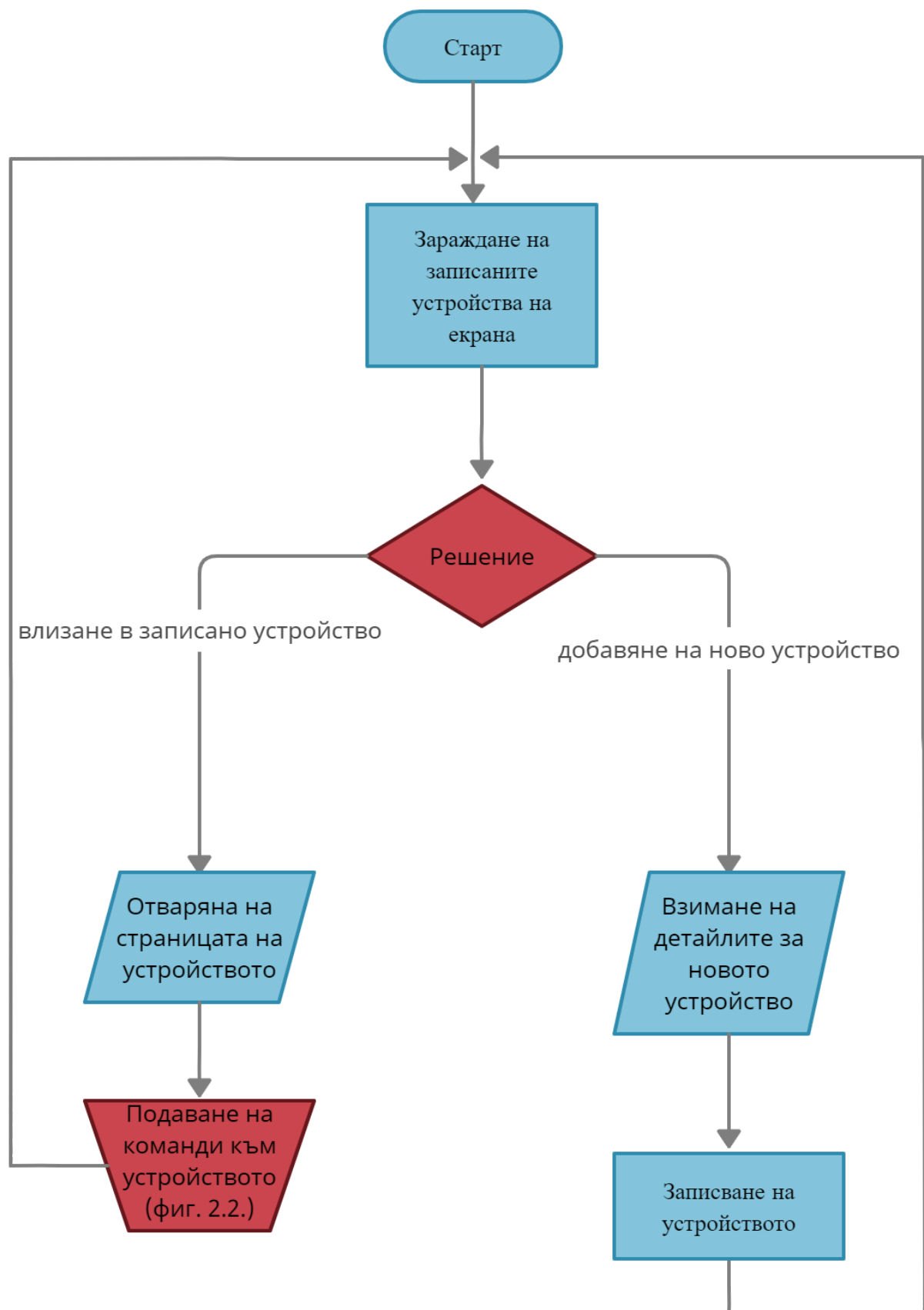
(Фигура 2.1 - цветова легенда на блоковите схеми)

Изпълнява се от микроконтролера
Изпълнява се от Android приложението
Изпълнява се ръчно от потребителя

(Фигура 2.2)



(Фигура 2.3)

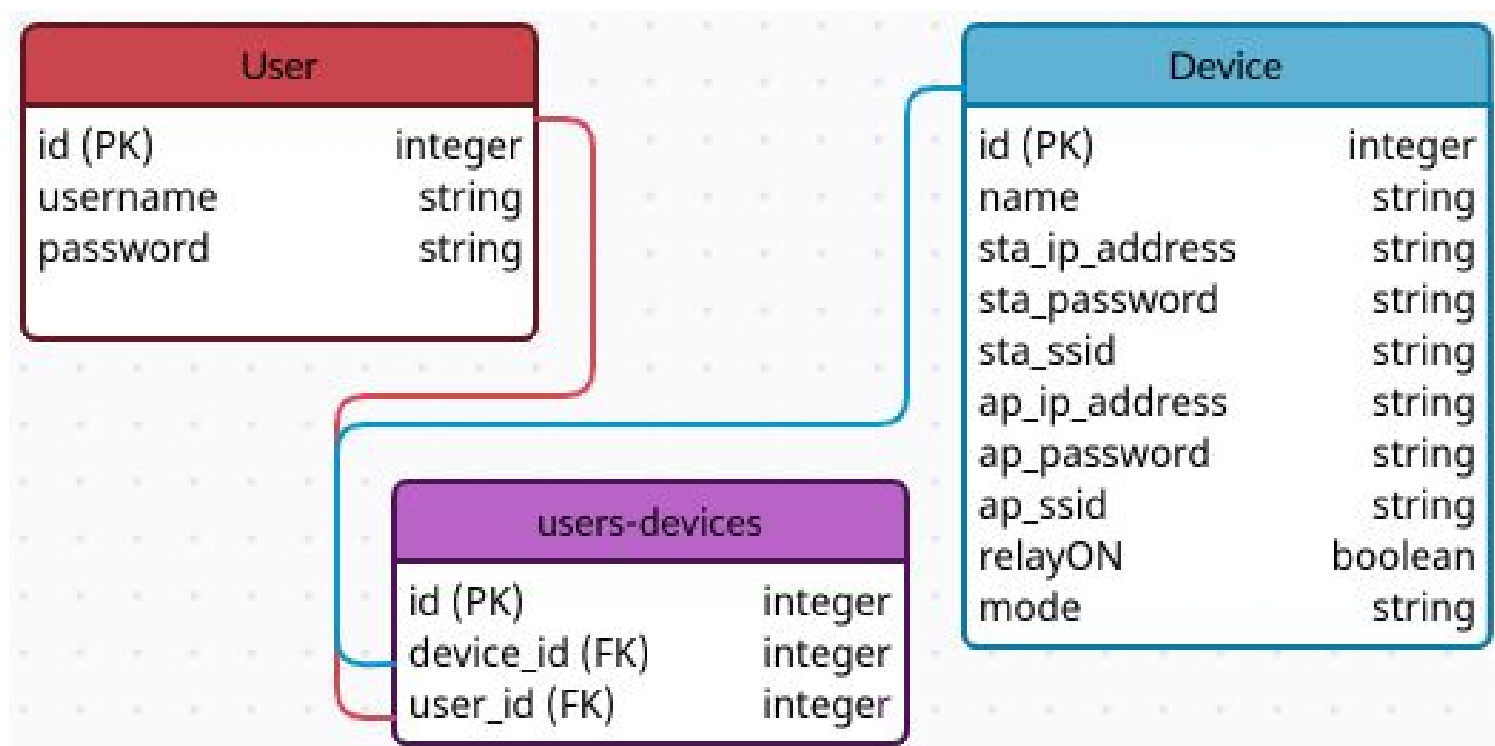


2.4 Проектира на структура на базата данни

Базата данни за този проект, определено не е от най-сложните които съм проетитал. За това **в момента на написва на това, дори** не използвам "истинска" база данни като например sqlite, а съхранявам малкото информация, от която се нуждае приложението да функционира в т.нар **SharedPreferences**. Повече информация за това в глава **4.2** . Класът device съхранява цялата информация за регистрираното в приложението устройство. Единствената връзка между таблиците е many-to-many изразена в таблицата users-devices. Един user може да има много регистрирани устройства. Едно устройство може да бъде контролирано от много потребители.

ЗАБЕЛЕЖКА: За сега системата с потребители е само теоретична и не е имплементирана в крайния продукт. По моя преценка тя не е достатъчно практична, за да влезе в крайния продукт, защото при липсата на сървър, информацията за потребителите трябва да се пази локално на мобилното устройство. **Това би означавало, че ако направим акаунт на едно устройство, няма да можем да достъпим същия акаунт от друго.**

(Фигура 2.3 - модел на базата данни)



ТРЕТА ГЛАВА

РЕАЛИЗАЦИЯ НА СИСТЕМА ЗА ДИСТАНЦИОННО КОНТРОЛИРАНЕ НА УСТРОЙСТВА ОТ ДОМА. ЕТАПИ НА РАЗРАБОТКА.

В началото на този проект, идеята за реализацията на embedded частта определено ме притесняваше най-много от всичко. Вероятно заради лимитирания ми опит си мислех, че тази от заданието ще отнеме голяма част от времето ми за разработка на проекта. Това далеч не се оказа вярно. Наистина в началото на разработката срещнах известни проблеми, но след като свикнах с начина на работа, нещата започнаха да се случват доста бързо и обемът на работата по тази част не беше прекалено голям. Главната причина за това според мен е добрият избор на микроконтролер за поставената задача. Разделих извършената работа на следните етапи.

3.1. Принципна електрическа схема.

Входна клема J2 - входното напрежение - 220V AC се подава към модул **HLK-PM01** преобразуващ го към 5V DC. Кондензатора **F1**, стабилизира напрежението. Тези 5 волта захранват:

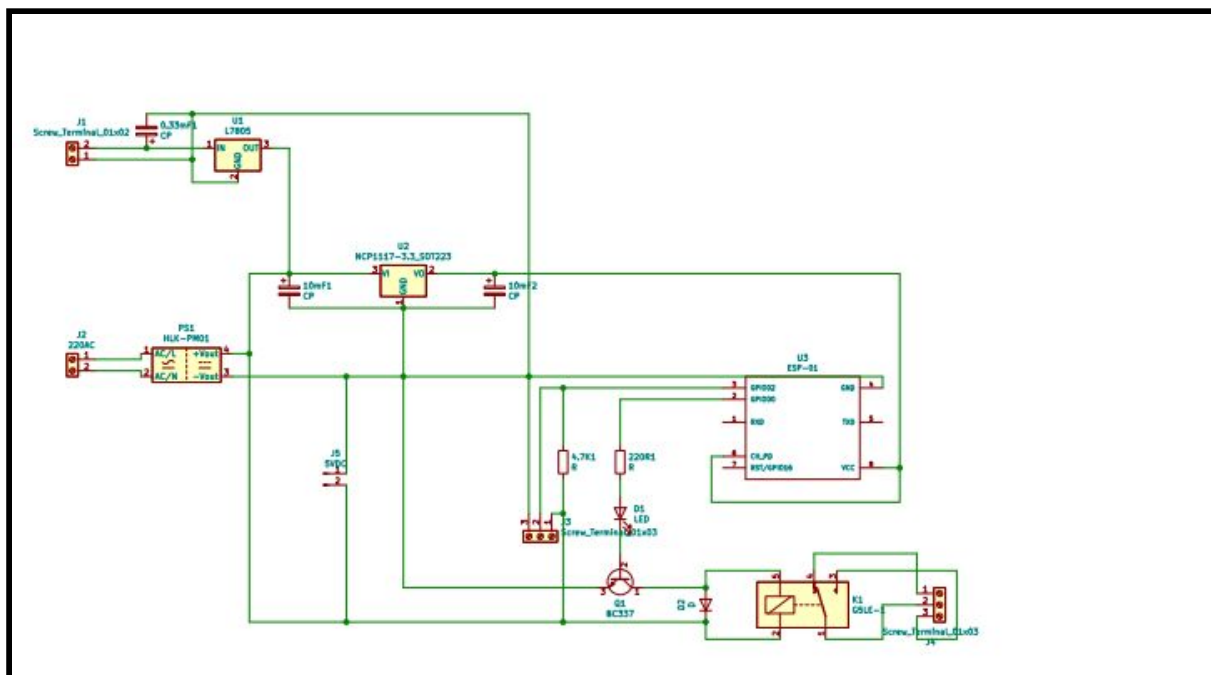
-втория преобразувател на напрежение **NCP1117-3.3_SOT223** , на изхода на които получаваме 3.3V за захранване на микроконтролера **ESP-01**

-положителния плюс на релето **G5LE-1**

Изхода на контролера минава през **R1** = 220Ω (зависи от типа на избрания транзистор), след това минава през светодиода **D1**, който индикира състоянието му. След което отива на базата на транзистора. На емитера на NPN транзистора се подава GND. При подава на + от изхода на контролера към базата, на колектора излиза GND. Тоест се образува ключ между емитер и колектор. Изхода на транзистора управлява GND на релето. Изхода на релето е изведен на клемата **J4**.

Забележка: Платката може да се захранва и на ниско напрежение през входната клема J1. Също към схемата може да се добави температурен датчик, който за сега не се използва в разработката.

(Фиг 3.1 - принципна електрическа схема на устройството)



3.2 Избор и изучаване на ESP8266 библиотеки за работа с Wi-Fi.

За да използвам ESP Wi-Fi модула на микроконтролера по предназначение и да се за вързвам за свързване в мрежа или да създавам собствена използвам стандартната **ESP8266WiFi** библиотека. Тя дава множество методи, които улесняват свързването към съществуваща мрежа или създаването на такава (access point mode). Методите са само 2: access point и station mode или и двата заедно. Не пиши щуротии, че ще ти зададат въпрос и няма да можеш да отговориш.

За превръщане на микроконтролера в своеобразен сървър, който може да приема информация от андроид приложението по протокола wifi, чрез http, използвам библиотеката ESP8266WebServer, която улеснява създаването на endpoint-и който микроконтролера да "слуша" за да получи командите и да при задаването на съответната им функционалност.

3.3 Изработване на тестова версия на устройството.

Голяма част от цикъла на разработка на проекта, използвах прототипна версия устройството.

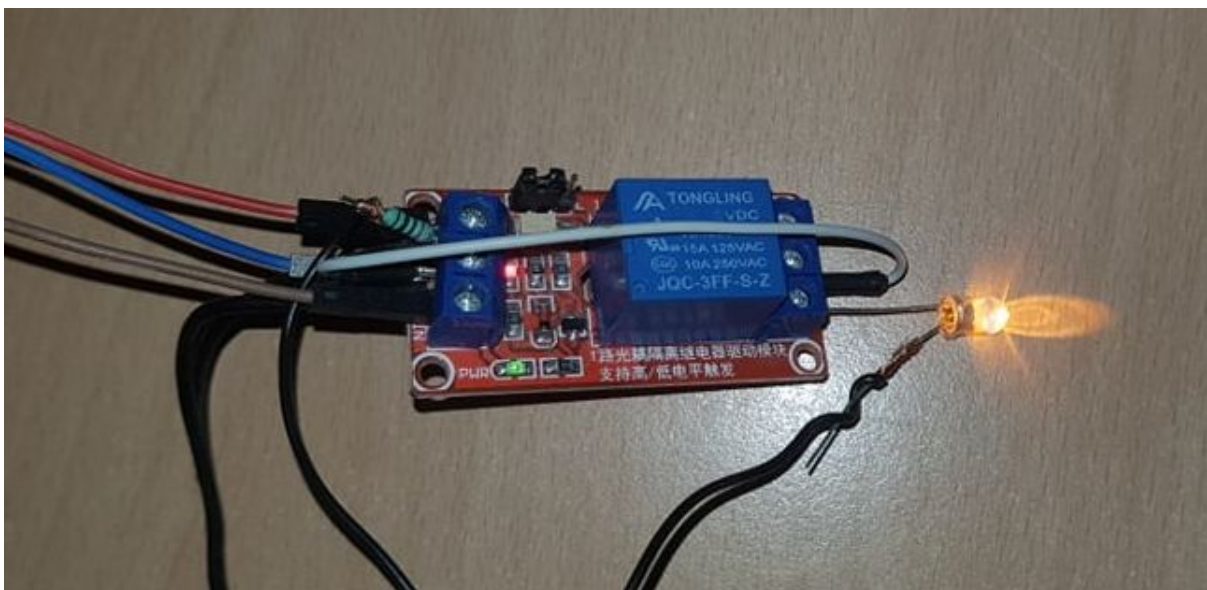
Първоначалната версия беше само микроконтролера. Той е напълно достатъчен за работа в мрежов и стационарен режим. Използвах вградения led (ограден в червено) на микроконтролера, за да му изпращам команди от андроид приложението да му подавам high или low, което симулира работата на едно реле.

(Фиг 3.2)



Впоследствие, вързах реле с led диод към контролера и това е прототипа, който почти до края на разработката.

(Фиг 3.3)



3.4 Писане на кода за микроконтролера.

Ето как работи управляващия микроконтролера код стъпка по стъпка:

1. Инициализация на следните параметри:

ssid_sta, password_sta - детайлите на мрежата към която устройството се свързва в мрежов режим

ssid_ap, password_ap - детайлите на мрежата към която устройството се свързва в стационарен режим

relay_pin - пинът на контролера, който управлява релето

Ip, gateway, netmask - нужни за метода за пускане на микроконтролера в стационарен режим - softAPConfig

server - инстанция на класа ESP8266WebServer, от библиотеката ESP8266WebServer. 80 е порта на който да работи сървъра.

(Фиг 3.4 - от embedded - sta_and_ap_mode)

```
String ssid_sta = "";
String password_sta = "";
String ssid_ap = "Esp8266AP";
String password_ap = "Esp8266AP";

uint8_t Relay_Pin = 0;

IPAddress ip(192, 168, 4, 1);
IPAddress gateway(192, 168, 4, 1);
IPAddress netmask(255, 255, 255, 0);

ESP8266WebServer server(80);
```

2. Стартова конфигурация в setup-а:

Задавам променливата, отговаряща на пина за релето като output и му подавам LOW, което практически означава че уреда който управлява е изключен. Задавам микроконтролера да работи в стационарен режим с WiFi.mode. Конфигурирам **ip, gateway и netmask** с **WiFi.softAPConfig** и пускам мрежата с **WiFi.softAP** с ssid - **ssid_ap** и парола - **password_ap**.

(Фиг 3.5 - от embedded - sta_and_ap_mode)

```
pinMode(Relay_Pin, OUTPUT);
digitalWrite(Relay_Pin, LOW);
WiFi.mode(WIFI_AP);
WiFi.softAPConfig(ip, gateway, netmask);
WiFi.softAP(ssid_ap, password_ap);
```

3. Все още в setup-a, задавам 3-те endpoint-a, който сървър да слуша със **server.on** (обяснение за какво се използва всеки от тях, в следващите точки от главата). Стартирам сървъра с **server.begin**. В loop единствено използвам метода **handleClient** на сървъра, както е описано в примерите от документацията на **ESP8266WebServer** библиотеката.

(Фиг 3.6 - от embedded - sta_and_ap_mode)

```
server.on("/led", LedSwitch);  
server.on("/switch", ModeSwitch);  
server.on("/config", ConfigSwitch);  
server.begin();  
  
}  
  
void loop(void) {  
    server.handleClient();  
}
```

4. Endpoint-a, контролиращ релето е /led. Асоциираната с него функция е **LedSwitch**. Заявките от Android приложението към сървъра изглеждат по следния начин: `ipAddress/led?state=newState` (on или off). Функцията **LedSwitch** обработва заявката. Ако state е on, подава HIGH на релето, тоест включва устройството, което то контролира. Ако state е off, подава LOW на релето, тоест изключва устройството, което контролира. Връща отговор 200 към адреса, при успешно извършване. Ако се подаде заявка различна от on или off, връща с код 404 и "Route not found".

Забележка: Името на този endpoint ще бъде променен от **led** на **relay** в финалната версия на проекта.

(Фиг 3.7 - от embedded - sta_and_ap_mode)


```

void LedSwitch() {
  if(server.arg("state") == "on")
  {
    digitalWrite(Relay_Pin, HIGH);
    server.send(200, "text/plain", "Led state changed to on");
  }
  else if(server.arg("state") == "off")
  {
    digitalWrite(Relay_Pin, LOW);
    server.send(200, "text/plain", "Led state changed to off");
  }
  else
  {
    server.send(404, "text/plain", "Route not found");
  }
}

```

5. Endpoint-а, контролиращ режима на устройството е mode. Асоциираната с него функция е **ModeSwitch**. Заявките от Андроид приложението към сървъра изглеждат по следния начин: ipAddress/switch?mode=newMode (sta или ap). Функцията **ModeSwitch** обработва заявката. Ако mode е sta, сменя режима в station, ако е ap, сменя режима в access point. Връща отговор 200 към адреса, при успешно извършване. Ако се подаде заявка различна от on или off, връща с код 404 и "Route not found".

(Фиг 3.8 - от embedded - sta_and_ap_mode)


```

void ModeSwitch() {
  if(server.arg("mode") == "sta")
  {
    WiFi.softAPdisconnect();
    WiFi.disconnect();
    WiFi.mode(WIFI_STA);
    delay(100);
    WiFi.begin(ssid_sta, password_sta);
    server.send(200, "text/plain", "Mode set to station");
  }
  else if(server.arg("mode") == "ap")
  {
    WiFi.mode(WIFI_AP);
    WiFi.softAPConfig(ip, ip, netmask);
    WiFi.softAP(ssid_ap, password_ap);
    server.send(200, "text/plain", "Mode set to access point");
  }
  else
  {
    server.send(404, "text/plain", "Route not found");
  }
}

```

6. Endpoint-a, контролиращ настройките на устройството е config. Асоциираната с него функция е **ConfigSwitch**. Функцията обработва заявката. Ако тя е за смяна на ssid при ap или sta mode, прави промяната. Ако е за парола, проверява дали тя е над 6 символа, което е минималното изискване. След това проверява дали е "none", който е ключовата дума, която съм избрал да означава, че мрежата няма парола. След това сменя паролата на дадената, ако предишните две не са изпълнени.

(Фиг 3.9 - от embedded - sta_and_ap_mode)

```

void ConfigSwitch()
{
    if(server.arg("ssid_ap") != "")
    {
        ssid_ap = server.arg("ssid_ap");
    }
    if(server.arg("password_ap").length() > 7)
    {
        password_ap = server.arg("password_ap");
    }
    if(server.arg("password_ap") == "none")
    {
        password_ap = "";
    }
}

```

(Фиг 3.10 - от embedded - sta_and_ap_mode)

```

    if(server.arg("ssid_sta") != "")
    {
        ssid_sta = server.arg("ssid_sta");
    }
    if(server.arg("password_sta").length() > 7)
    {
        password_ap = server.arg("password_sta");
    }
    if(server.arg("password_sta") == "none")
    {
        password_sta = "";
    }
}

```

ЧЕТВЪРТА ГЛАВА

ПРОГРАМНА РЕАЛИЗАЦИЯ НА ANDROID ПРИЛОЖЕНИЕ – УПРАВЛЯВАЩ СОФТУЕР. ЕТАПИ НА РАЗРАБОТКА

Както казах и в увода, реших да се вкарам в непознати води с този проект. За първи път се заех да пиша Android приложение и това се оказа трудна, но забавна задача. Разделих извършената работа по него в следните етапи. В тези етапи, се налага да говоря успоредно за backend и front-end (user interface), защото иначе смисъла на някои части се губят.

ЗАБЕЛЕЖКА: Етапите не следват хронологичния ред на разработка. Наредени са по избрания начин с цел, читателя да има най-добър шанс да следва логиката на разработката.

4.1 Метод за дългосрочно съхранение на данни.

Въпреки че проекта не се нуждае от комплексна база данни, съхраняването на малкото данни които използва приложението трябва да бъде дългосрочно или поне те да не се губят при затваряне на приложението. Решението до което достигнах е с използването на т. нар **Shared Preferences**. SharedPreferences обекта е глобален за цялото приложение. Данните които се записват чрез него са в формат "ключ-стойност". Те се съхраняват в файл с име зададено от нас или default файл, както е в случая.

За по-лесна работа с SharedPreferences, имплементирах класа

Preferences (намира се в пакета com.rosen.homecontrolapp.storage). Класа има следните полета:

ctx - Context обект, които е характерен за класовете от тип Activity. В него се съхранява информация за приложението. Preference класът не е от тип Activity и няма свой context. За да се справим с този проблем, при създаване на Preference обект, му предаваме context-а на Activity-то, от което сме го създали.

preferences - Инстанция на SharedPreferences. Викаме SharedPreferences с default настройките. За целта се нуждаем от context и използваме ctx, които получаваме от класът в който е създаден обекта.

(Фиг. 4.1 - от com.rosen.homecontrolapp.storage - Preferences)

```
class Preferences(ctx: Context) {  
  
    val preferences: SharedPreferences = PreferenceManager.getDefaultSharedPreferences(ctx)
```

За добавяне на нова или промяна на съществуващ запис в SharedPreferences, имплементирам метода getSharedPreferencesValue в Preferences. Той приема обект от помощния data class Storage - фиг. 4.2. За да променяме стойностите в SharedPreferences обект, викаме инстанция на SharedPreferences. Editor обекта чрез .edit(), задаваме стойността с .put() и запазваме промените с .apply() - фиг. 4.3

(Фиг. 4.2 от com.rosen.homecontrolapp.model - Storage)

```
data class Storage(val key: String, val default: String)
```

(Фиг. 4.3 от com.rosen.homecontrolapp.storage - Preferences)

```
private fun changeSharedPreferencesValue(key : String, newValue : String) {  
    preferences.edit().putString(key, newValue).apply()  
}
```

За цел лесно извличане на записи от SharedPreferences, имплементирам метода `getSharedPreferencesValue`. Той приема единствен аргумент - обект от data class `Storage` - фиг. 4.2, в който се съхраняват ключа и `defaultValue`-то на записа който искаме да достъпим. Достъпването се случва с викане на `SharedPreferences.getString()`, с аргументи полетата в `storage` обекта.

(Фиг. 4.4 от com.rosen.homecontrolapp.storage - Preferences)

```
private fun getSharedPreferencesValue(storage: Storage): String {  
    val value = preferences.getString(storage.key, storage.default)!!  
    return if (value.isBlank()) storage.default else value  
}
```

Конкретните параметри, които пазим в `SharedPreferences` са показани в фигура 4.6. Целта на този файл е пазене на `key-defaultValue` двойки за удобство при достъпване на конкретен запис от в `SharedPreferences` през гореописания `getSharedPreferencesValue` метод. За какво се използва все един от записите в `SharedPreferences`:

ключ	Какво съхранява
device_id	Уникалното id на последното записано устройство. С всяко ново устройство нараства с 1 чрез метода

	generateDeviceId от Preferences
devices	Всички записани device обекти в json format.
command_logs	Всички записани commandLogs обекти в json format.
sta_ssid, sta_wifi_password, sta_ip_address, sta_port	Настройките на активното устройство в sta mode.
ap_ssid, ap_wifi_password, ap_ip_address, ap_port	Настройките на активното устройство в ap mode.

(Фиг. 4.5 от com.rosen.homecontrolapp.constant - Constant.kt

```
val DEVICE_ID_STORAGE = Storage( key: "device_id", default: "0")
val DEVICE_STORAGE = Storage( key: "devices", default: "")
val COMMANDS_LOG_STORAGE = Storage( key: "command_logs", default: "")

val STA_SSID_STORAGE = Storage( key: "sta_ssid", default: "")
val STA_WIFI_PASSWORD_STORAGE = Storage( key: "sta_wifi_password", default: "")
val STA_IP_ADDRESS_STORAGE = Storage( key: "sta_ip_address", default: "")
val STA_PORT_STORAGE = Storage( key: "sta_port", default: "")

val AP_SSID_STORAGE = Storage( key: "ap_ssid", default: "")
val AP_WIFI_PASSWORD_STORAGE = Storage( key: "ap_wifi_password", default: "")
val AP_IP_ADDRESS_STORAGE = Storage( key: "ap_ip_address", default: "")
val AP_PORT_STORAGE = Storage( key: "ap_port", default: "")
```

За удобство, имплементирам методите от фиг. 4.6. Тяхната функция е да улесняване на извличането на данни за активното устройство. Тяхната имплементация е чрез getSharedPreferencesValue - фиг. 4.3 с аргумент един от обектите от Constant.kt - фиг 4.5

(Фиг. 4.6 от com.rosen.homecontrolapp.storage - Preferences)

```
fun getIpAddressSta(): String = getSharedPreferencesValue(Constant.STA_IP_ADDRESS_STORAGE)
fun getPortSta(): String = getSharedPreferencesValue(Constant.STA_PORT_STORAGE)
fun getPasswordSta(): String = getSharedPreferencesValue(Constant.STA_WIFI_PASSWORD_STORAGE)
fun getSsidSta(): String = getSharedPreferencesValue(Constant.STA_SSID_STORAGE)
fun getIpAddressAp(): String = getSharedPreferencesValue(Constant.AP_IP_ADDRESS_STORAGE)
fun getPortAp(): String = getSharedPreferencesValue(Constant.AP_PORT_STORAGE)
fun getPasswordAp(): String = getSharedPreferencesValue(Constant.AP_WIFI_PASSWORD_STORAGE)
fun getSsidAp(): String = getSharedPreferencesValue(Constant.AP_SSID_STORAGE)
```

4.2 Добавяне на устройство. Визуализация на записаните устройства.

Добавянето и запазването на устройства е едно от фундаменталните изисквания към проекта. Позволява бърза работа с повече от едно устройство, което прави потенциала на приложението доста по-голям. Въпреки това, тази функционалност беше имплементирана сравнително късно в процеса на разработка. През голяма част от него тестването на приложението се случваше с еднинично и `hardcode`-нато устройство.

Класът модел за устройството е `Device` , намиращ се в пакета. Кратко обяснение на полетата му:

-context – Context обект на класа в който е създадено устройството, нужен е за създаване на Preferences обект - фиг. 4.1 чрез който се генерира уникалното `id` на устройство

ap_ip, ap_ssid, ap_password, sta_ip, sta_ssid, sta_password – настройките на устройството

id – уникален идентификатор на устройството

relayON = моментното състояние на релето

mode – режима на работа на устройството. DeviceMode е **enum** с полета ACCESS_POINT и STATION

Създаването и визуализацията на устройствата се случва в класа MainActivity (намира се в пакета com.rosen.homecontrolapp.activity).

Визуализацията на устройствата се случва по следния начин:

1. Създава се Preferences обект - prefs фиг. 4.7. От него използвам метода getDevices - фиг. 4.8. Метода работи като взима записа от SharedPreferences с ключ devices и чрез функцията на библиотеката Gson - fromJson(), го преобразува от json формат в който се пази в SharedPreferences в ArrayList<Devices> с който ни е удобно да работим.

(Фиг. 4.7 от com.rosen.homecontrolapp.activity - MainActivity)

```
val prefs = Preferences( ctx: this)
devices = prefs.getDevices()
```

(Фиг. 4.8 от com.rosen.homecontrolapp.storage - Preferences)

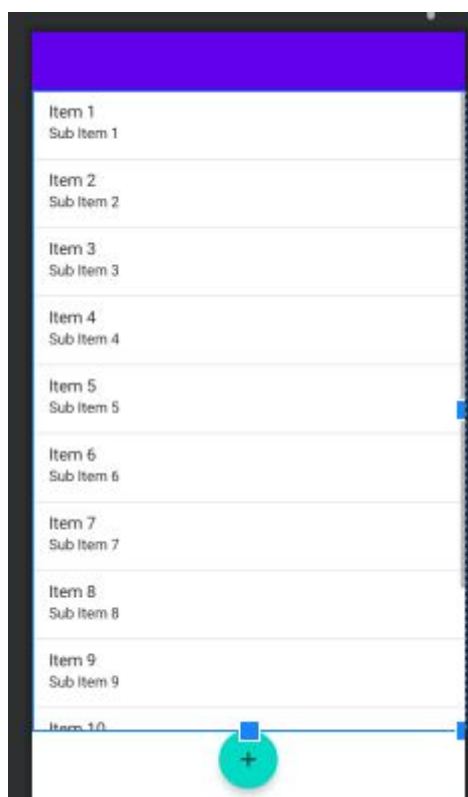
```
fun getDevices(): ArrayList<Device>{
    if(Gson().fromJson<ArrayList<Device>>(getSharedPreferencesValue(Constant.DEVICE_STORAGE), object : TypeToken<ArrayList<Device>>() {}.type) == null) return ArrayList<Device>()
    else return Gson().fromJson<ArrayList<Device>>(getSharedPreferencesValue(Constant.DEVICE_STORAGE), object : TypeToken<ArrayList<Device>>() {}.type)
}
```

2. Създавам listView обект, отговарящ на ListView-то с id = device_list от layout-а на страницата - фиг 4.10 и 4.11. От ArrayList-а от устройствата, който взехме в миналата стъпка, чрез stream извличам имената на устройствата подредени в ArrayList<String> - deviceNames. Създавам ArrayAdapter обект и зареждам имента на устройствата в него и го задавам като адаптера на listView обекта.

(Фиг. 4.9 от com.rosen.homecontrolapp.activity - MainActivity)

```
val listView = findViewById<ListView>(R.id.device_list)
val deviceNames = devices.map { it.name }.toMutableList()
val arrayAdapter = ArrayAdapter<String>(context: this, android.R.layout.simple_list_item_1, deviceNames)
listView.adapter = arrayAdapter
```

(Фиг. 4.10 и 4.11 от res/layout - activity_main.xml)



```
<ListView
    android:id="@+id/device_list"
    android:layout_width="wrap_content"
    android:layout_height="607dp"
    tools:layout_editor_absoluteX="79dp"
    tools:layout_editor_absoluteY="232dp" />
```

3. За финал правя така, че изписаните имена на устройствата, да могат да бъдат натискани и това да води до страницата на устройството. Това става, чрез `setOnClickListener`. Взимам името на избраното устройство. Намирам го в списъка от устройства и изпращам `id`-то му на класа отговарящ за визуализацията на отделните устройства - `deviceActivity` чрез метода `putExtra` на обекта `Intent`. Подробно описание на защо трябва да го изпратя в следващата трета част на тази глава.

(Фиг. 4.12 от com.rosen.homecontrolapp.activity - MainActivity)

```

listView.setOnItemClickListener{parent, view, position, id ->
    val deviceName = parent.getItemAtPosition(position)
    val device = devices.find { it.name.equals(deviceName)}
    intent = Intent( packageContext: this, DeviceActivity::class.java)
    intent.putExtra( name: "Device Id", (device as Device).id)
    startActivity(intent)
}

```

Добавянето на ново устройство се случва по следния начин:

1. Създавам инстанция на `floatingActionButton` - `addBtn`, отговаряща на бутона от `activity_layout.xml` с `id = addDeviceButton`. Задавам `setOnClickListener`-а на бутона да отваря диалог - `AddDeviceDialog` - фиг 4.13.

(Фиг. 4.13 от `com.rosen.homecontrolapp.activity` - `MainActivity`)

```

addBtn.setOnClickListener {
    var dialog = AddDeviceDialog()
    dialog.show(supportFragmentManager, "addDeviceDialog")
}

```

2. Създавам инстанция на `Preferences`, за да заредя запазените в `SharedPreferences` `devices` чрез `getDevices` (обяснен в глава 4.1). Задавам `view`-то на диалога да е `fragment_add_device.xml`.

(Фиг. 4.14 от `com.rosen.homecontrolapp.activity` - `AddDeviceDialog`)

```

val prefs = Preferences(this.context as Context)
devices = prefs.getDevices()

val rootView: View = inflater.inflate(R.layout.fragment_add_device, container, false)

```

3. Създавам обекти за бутоните и текстовите полета, отговарящи на тези в `fragment_add_device.xml`.

(Фиг. 4.15 от `com.rosen.homecontrolapp.activity` - `AddDeviceDialog`)

```

val rootView: View = inflater.inflate(R.layout.fragment_add_device, container, attachToRoot: false)

val cancelBtn = rootView.findViewById<Button>(R.id.cancelButton)
val saveBtn = rootView.findViewById<Button>(R.id.saveButton)
val deviceName = rootView.findViewById<EditText>(R.id.DeviceNameEditText)
val StaSSID = rootView.findViewById<EditText>(R.id.StaSSIDEditText)
val StaPassword = rootView.findViewById<EditText>(R.id.StaPasswordEditText)
val StaIp = rootView.findViewById<EditText>(R.id.StaIPEditText)
val ApSSID = rootView.findViewById<EditText>(R.id.ApSSIDEditText)
val ApPassword = rootView.findViewById<EditText>(R.id.ApPasswordEditText)
val ApIp = rootView.findViewById<EditText>(R.id.ApIPEditText)

```

4. Задавам `onClick`Listener-и на бутона `cancelBtn`, който затваря диалога и на `saveBtn`, който създава нов `Device` обект с полетата зададени от потребителя - фиг 4.17.

(Фиг. 4.16 от `AddDeviceDialog`)

(Фиг. 4.17 от `add_device_fragment`)

```

cancelBtn.setOnClickListener()
{ it: View!
    dismiss()
}

saveBtn.setOnClickListener()
{ it: View!
    devices.add(Device(
        this.context as Context,
        name = deviceName.text.toString(),
        sta_ssid = StaSSID.text.toString(),
        sta_password = StaPassword.text.toString(),
        sta_ip = StaIp.text.toString(),
        ap_ssid = ApSSID.text.toString(),
        ap_password = ApPassword.text.toString(),
        ap_ip = ApIp.text.toString()))
    prefs.saveDevices()
    dismiss()
}

return rootView

```

5. Запазвам новия `Device` в локалната инстанция на `Devices`. Запазвам локалните промени по `Devices` в `SharedPreferences` чрез `saveDevices` метода на `Preferences`, който съм имплементирал за удобство. Той преобразува новата версия на `Devices` от `ArrayList<Device>` в псевдо json формат (реално низ) чрез метода `toJson` на библиотеката `Gson` и записва

промените в SharedPreferences с ключ devices чрез метода changeSharedPreferencesValue (обяснен в глава 4.1)

(Фиг. 4.18 от com.rosen.homecontrolapp.storage - Preferences)

```
fun saveDevices(){  
    val jsonString = Gson().toJson(devices)  
    changeSharedPreferencesValue( key: "devices", jsonString)  
}
```

4.3 Изпращане на команди към устройството. Работа с множество устройства.

За цел работа с много устройства, всяко записано в паметта устройство, трябва да има своя страница. Нямам как обаче да направя отделна и различна xml страница за всяко устройство, защото броят на устройствата е неопределен. За това имам само една страница - device_activity.xml и един клас контролер - DeviceActivity. Ето как работи DeviceActivity, така че да създава различна страница за всяко от устройствата:

1. Приема **педаденото** от MainActivity device id. Запълва празните textView-тата с името, режима и състоянието на релето на устройството, което отговаря на това id. Също записва настройките на устройството в SharedPreferences чрез Preferences.loadDevice. Правя това, заради следващата част от главата - промяна на настройките на устройството.

(Фиг. 4.19 от com.rosen.homecontrolapp.activity - DeviceActivity)

```

override fun onCreate(savedInstanceState: Bundle?) {
    val preferences = Preferences( ctx: this)
    deviceId = intent.getIntExtra( name: "Device Id", defaultValue: 0)
    device = devices.find { it.id.equals(deviceId) }
    Preferences( ctx: this).loadDevice(device as Device)

    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_device)
    setSupportActionBar((findViewById(R.id.toolbar_device)))
    supportActionBar?.setTitle(device?.name)
    val relayStatusTextView = findViewById<TextView>(R.id.statusVariable)
    val modeTextView = findViewById<TextView>(R.id.modeVariable)
}

```

(Фиг. 4.20 от com.rosen.homecontrolapp.activity - DeviceActivity)

```

fun loadDevice(device: Device){
    changeSharedPreferencesValue( key: "sta_ssid", device.sta_ssid)
    changeSharedPreferencesValue( key: "sta_wifi_password", device.sta_password )
    changeSharedPreferencesValue( key: "sta_ip_address", device.sta_ip)
    changeSharedPreferencesValue( key: "sta_port", device.sta_port )
    changeSharedPreferencesValue( key: "ap_ssid", device.ap_ssid)
    changeSharedPreferencesValue( key: "ap_wifi_password", device.ap_password)
    changeSharedPreferencesValue( key: "ap_ip_address", device.ap_ip )
    changeSharedPreferencesValue( key: "ap_port", device.ap_port)
}

```

2. Създавам бутон обектите, отговарящи на тези от layout-a на страницата. Задавам onClickListener-и, за функционалността им.

(Фиг. 4.21 от com.rosen.homecontrolapp.activity - DeviceActivity)

```

val btnON = findViewById<Button>(R.id.buttonOn)
val btnOFF = findViewById<Button>(R.id.buttonOff)
val btnAP = findViewById<Button>(R.id.buttonSwitchAp)
val btnSTA = findViewById<Button>(R.id.buttonSwitchSta)

```


За **заявките** към устройството използвам библиотеката Retrofit2 - фиг 4.22.

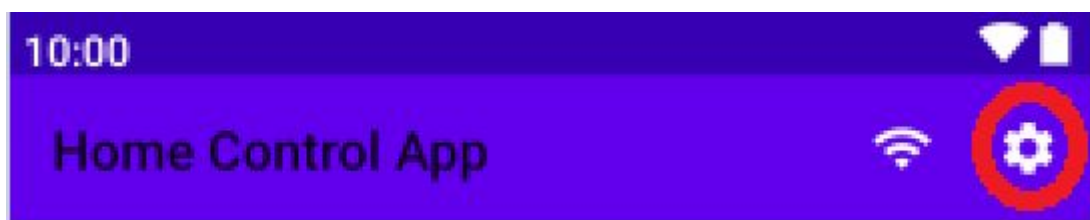
(Фиг. 4.22 от com.rosen.homecontrolapp.service - EspApiService)

```
@GET( value: "/led")
fun ledState(@Query( value: "state") ledState: String): Call<Void>
@GET( value: "/switch")
fun switchMode(@Query( value: "mode") newMode: String): Call<Void>
@GET( value: "/config")
fun switchConfig(@Query( value: "ssid_ap") ssidAp: String?,
    @Query( value: "password_ap") passwordAp: String?,
    @Query( value: "port_ap") portAp: String?,
    @Query( value: "ssid_sta") ssidSta: String?,
    @Query( value: "password_sta") passwordSta: String?,
    @Query( value: "port_sta") portSta: String?): Call<Void>
```

4.4 Промяна на настройките на устройството.

Промяната на настройките на устройството се случва по сходен начин. При натискане на бутона за настройки - фиг 4.23, се отваря ново activity - ConfigurationActivity и към него се праща id-то на устройството.

(Фиг. 4.23 от device_toolbar.xml)



ConfigurationActivity работи с preference.xml. В editTextPreferences полетата в него са директно свързани със съответстващите на тях записи в SharedPreferences. Вече заредих настройките на устройството което управлявам в момента в SharedPreferences в миналата част на тази глава чрез Preferences.loadDevice метода. Това може да изглежда малко странно и ненужно, но е нужно, за да използвам една preferences.xml станица за множество устройства и техните различни настройки.

(Фиг. 4.24 от res/xml preferences.xml)

```
<EditTextPreference
    android:dialogMessage="Set SSID for STA mode"
    android:dialogTitle="SSID"
    android:inputType="text"
    android:key="sta_ssid"
    android:title="SSID" />

<EditTextPreference
    android:dialogMessage="Set password for STA mode"
    android:dialogTitle="Password"
    android:inputType="textPassword"
    android:key="sta_wifi_password"
    android:title="Password" />

<EditTextPreference
    android:dialogMessage="Set port for STA mode"
    android:dialogTitle="Port"
    android:inputType="text"
    android:key="sta_port"
    android:title="Port" />
```

При **промяна** на текста в тези полета, се вика onSharedPreferencesChanged метода на ConfigurationActivity. Той изпраща заявка за промяна на съответната настройка към устройството. Записва промяната в настройката в локалния за приложението device обект, както и в SharedPreferences чрез saveDevices метода на Preferences.

(Фиг. 4.25 от com.rosen.homecontrolapp.activity - ConfigurationActivity)

```
override fun onSharedPreferencesChanged(sharedPreferences: SharedPreferences, key: String) {
    val prefs = Preferences( ctx, this)
    when (key) {
        "ap_ssid" -> {
            espConnector.sendConfigChangeRequest(ssidAp = sharedPreferences.getString(key, Constant.AP_SSID_STORAGE.default))
            device!!.ap_ssid = sharedPreferences.getString(key, Constant.AP_SSID_STORAGE.default)!!
        }
        "ap_wifi_password" -> {
            espConnector.sendConfigChangeRequest(passwordAp = sharedPreferences.getString(key, Constant.AP_WIFI_PASSWORD_STORAGE.default))
            device!!.ap_password = sharedPreferences.getString(key, Constant.AP_WIFI_PASSWORD_STORAGE.default)!!
        }
    }
}
```

4.5 Записване и визуализация на изпратените команди в timeline

(Фиг. 4.26 от com.rosen.homecontrolapp.model - CommandLog)

```
class CommandLog(val deviceName: String, val command: String) {  
  
    var timestamp: String = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT).format(LocalDateTime.now())  
  
    fun get_info(): String{  
        return "$deviceName      $command      $timestamp"  
    }  
}
```

deviceName - името на устройството асоциирано с командата

command - вида на командата

timestamp - времето на изпращане на командата

get_info() - улеснява визуализация

Визуализация: Зареждам запазените в SharedPreferences команди чрез метода getLogs. Подобно на част две на тази глава. Зареждам информацията от командите в ListView-то на страницата.

(Фиг. 4.27 от com.rosen.homecontrolapp.activity - CommandActivity)

```
val prefs = Preferences( ctx: this)  
  
commandLogs = prefs.getLogs()  
val commandLogs_info = commandLogs.map { it.get_info() }.toMutableList()  
val listView = findViewById<ListView>(R.id.LogListView)  
val arrayAdaper = ArrayAdapter<String>(context: this, android.R.layout.simple_list_item_1, commandLogs_info)  
listView.adapter = arrayAdaper
```

Записване: Всеки път когато пращам команда към устройството, я записвам в SharedPreferences чрез метода saveLog.

(Фиг. 4.28 от com.rosen.homecontrolapp.activity - DeviceActivity)

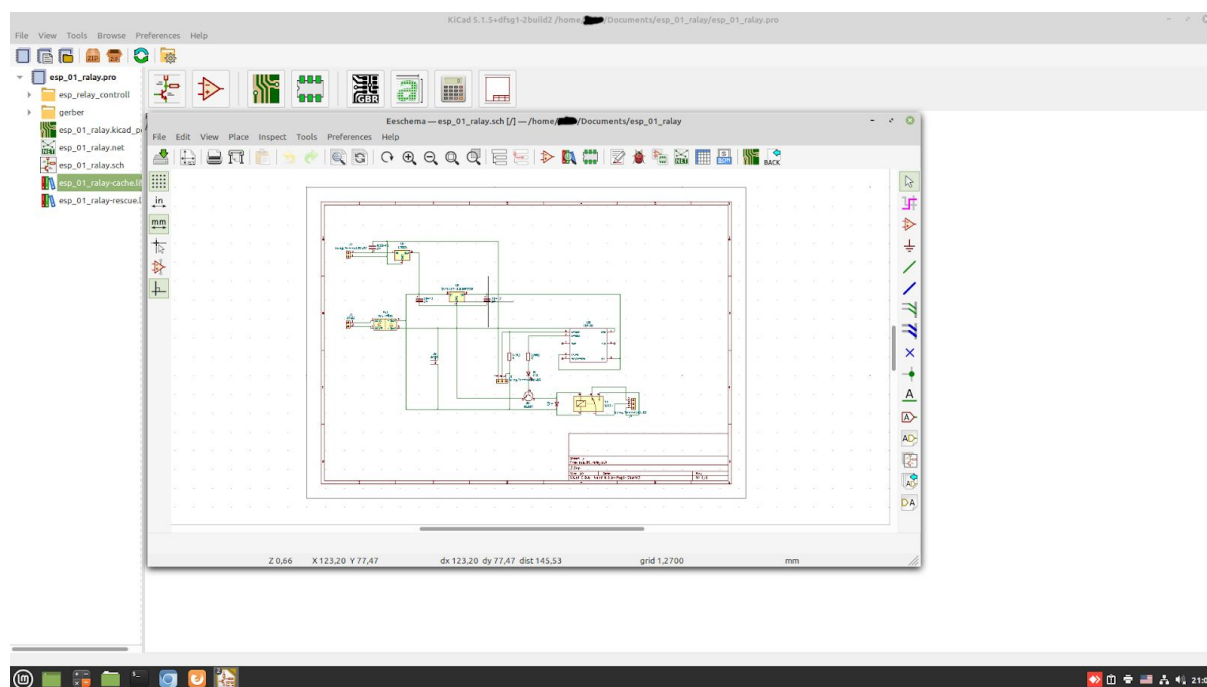
```
btnON.setOnClickListener { it: View! -> {  
    espConnector.sendLedRequest(state = "on")  
    device!!.relayOn = true  
    relayStatusTextView.setText("ON")  
    preferences.saveDevice(device)  
    preferences.addLog(CommandLog(deviceName = device!!.name, command = "Relay ON"))  
}
```


ПЕТА ГЛАВА

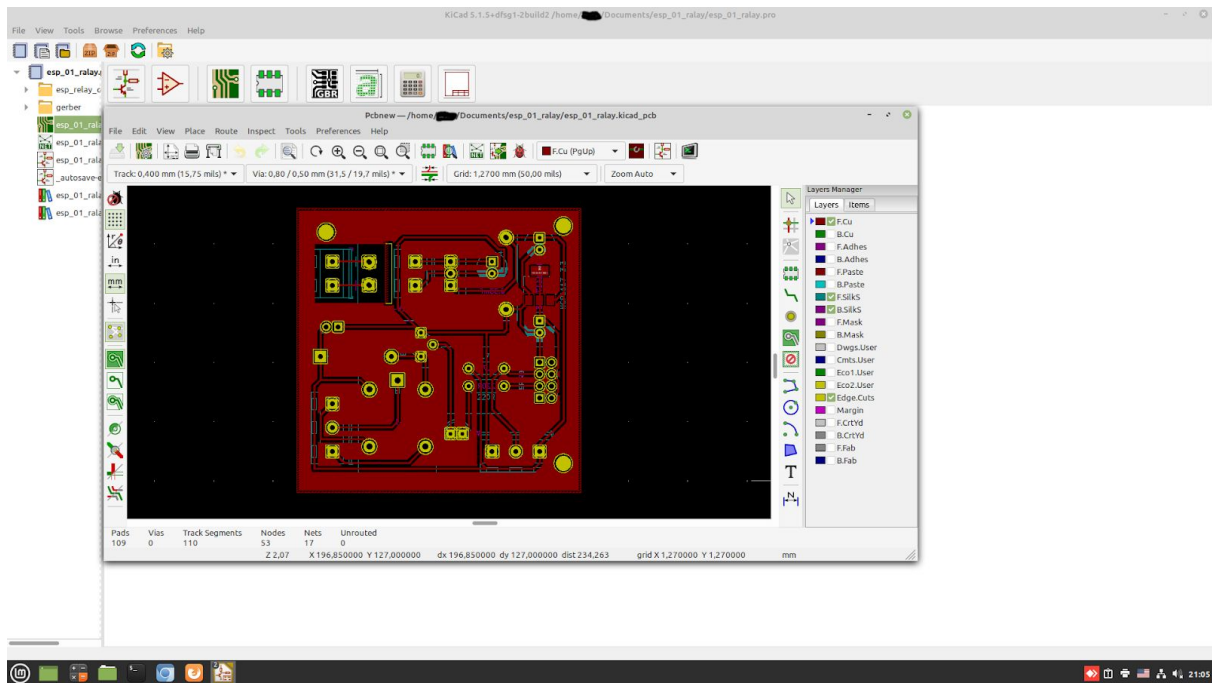
ПРОЕКТИРАНЕ НА ГРАФИЧНИ ОРИГИНАЛИ НА ПЕЧАТНИ ПЛАТКИ

Печатната платка е разработена с KiCad. Най-голямата особеност при него са библиотеките със символи, които идват с безплатен пълен лиценз за комерсиална и некомерсиална работа. Иначе софтуера е напълно стандартен и има всички нужни системи за създаването на схемата по която е направена печатната платка.

(Фиг 5.1)



(Фиг 5.2)



ШЕСТА ГЛАВА

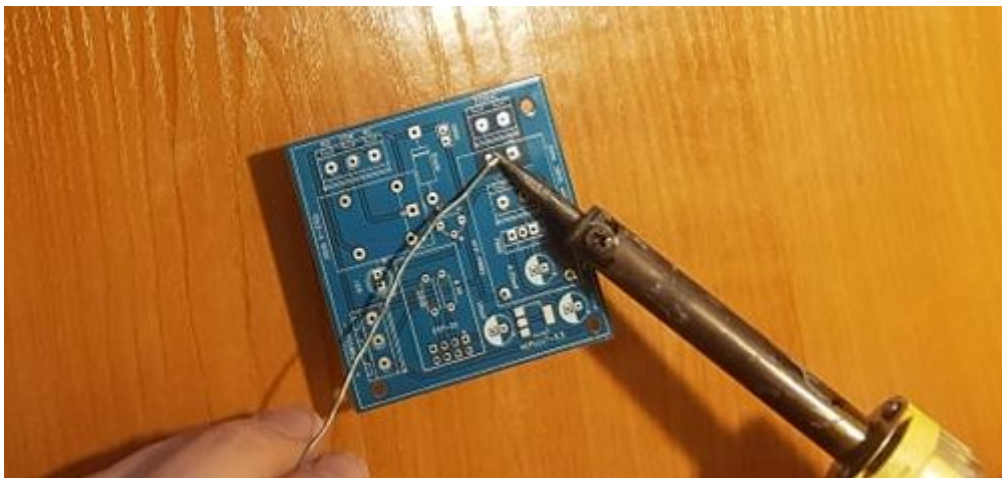
СЪЗДАВАНЕ НА РАБОТОСПОСОБЕН МОДЕЛ НА МИКРОКОНТРОЛЕРНА СИСТЕМА ЗА ДИСТАНЦИОННО КОНТРОЛИРАНЕ НА УСТРОЙСТВА ОТ ДОМА

Елементна база за изработване на работоспособен модел:

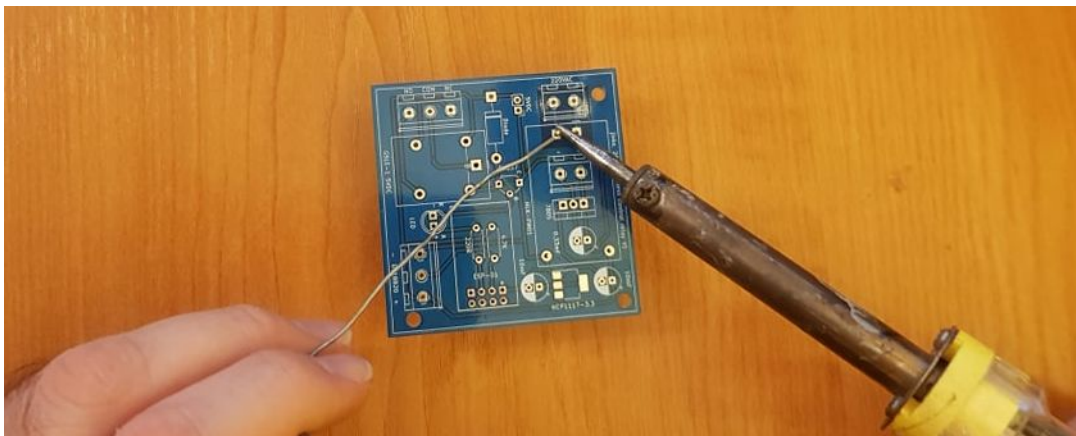
- микроконтролер ESP8266
- реле
- захранващ модул - вход 220V AC, изход 5V DC (може да липсва)
- регулатор на напрежение - вход 5V DC, изход 3.3V DC
- 2 кондензатора 10 μ F, 25V
- транзистор NPN
- клеми
- диод
- цокъл
- светодиод

Забележка: Финалната схема е предвидена за работа с датчик за температура, който не се използва в проекта.

(фиг 6.1)



(Фиг 6.2)



(Фиг 6.3)



(Фиг 6.4)



РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ

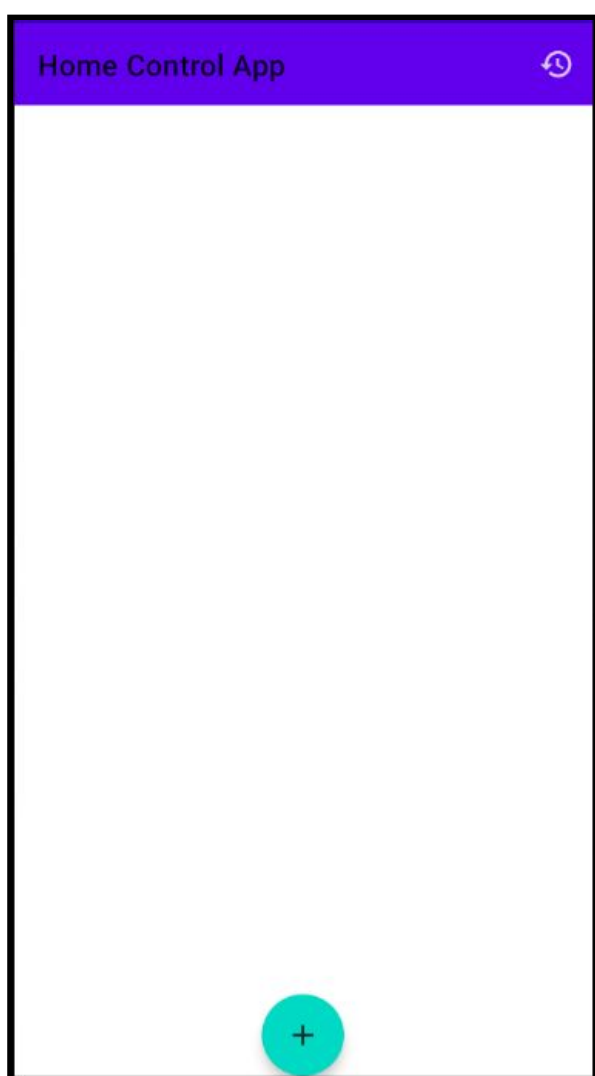
7.1 Инсталация на софтуера за управление на устройството

Инсталацията на софтуера е сравнително проста. Той е под формата на арк. Качва се на мобилното устройство, след това се run-ва. Това е.

7.2 Добавяне на устройства

След като инсталирате приложението, трябва да добавим устройството си. При пускане на приложението за пръв път, началният екран ще изглежда по следния начин:

(Фиг 7.1)

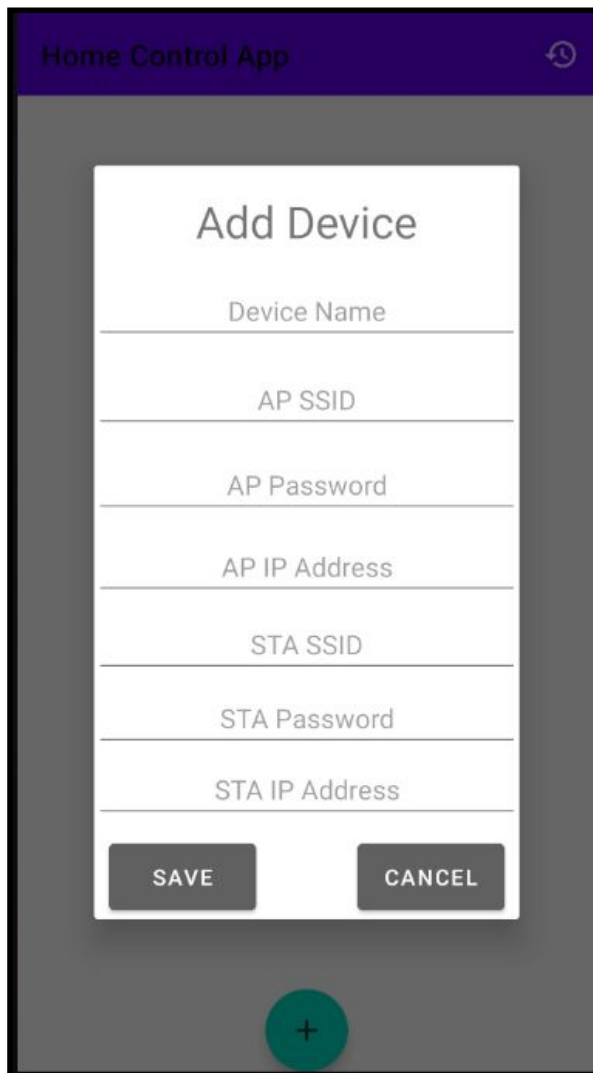


За да се добави устройство, се натиска зеления бутон, което отваря диалога за добавяне на устройство. Попълваме ги. Това може да се случи и след добавяне на устройството чрез настройките му.

Зададените начални настройки на устройството са то да работи в access point режим с ip 192.168.4.1, ssid Esp8266AP и парола Esp8266AP.

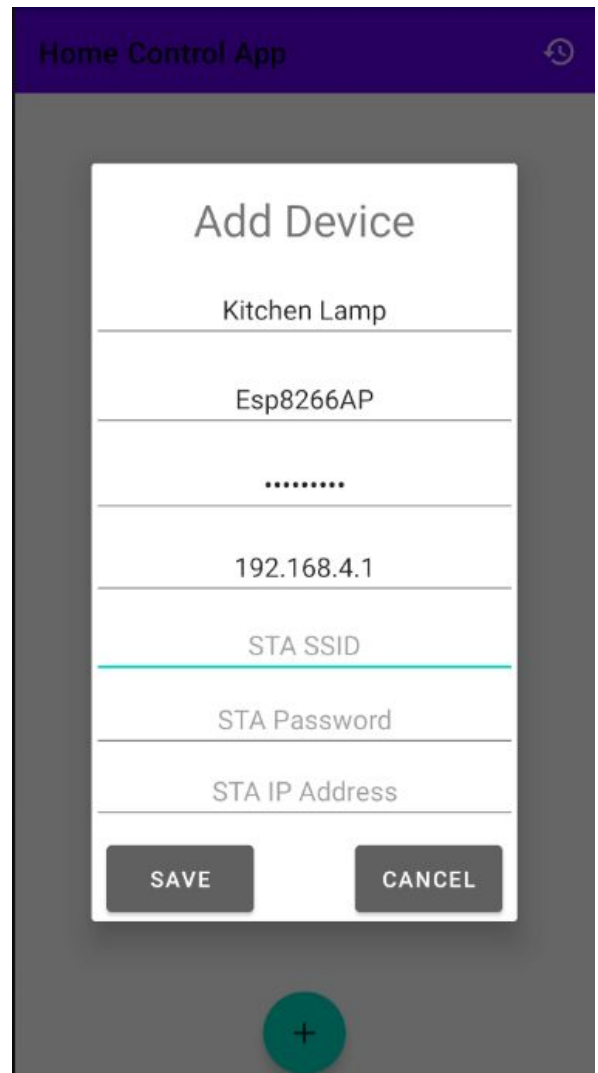
Забележка: при разработване на още подобни устройства, ip-то няма да е hardcode-нато.

(Фиг 7.2)



The screenshot shows the 'Add Device' form in the Home Control App. The form has a purple header with the app name and a refresh icon. The form itself is white and contains eight input fields: Device Name, AP SSID, AP Password, AP IP Address, STA SSID, STA Password, and STA IP Address. At the bottom of the form are two buttons: 'SAVE' and 'CANCEL'. Below the form is a green circular button with a white plus sign.

(Фиг 7.3)



The screenshot shows the 'Add Device' form in the Home Control App, now with data entered. The input fields are filled with: Kitchen Lamp, Esp8266AP, a masked password (seven dots), 192.168.4.1, STA SSID (highlighted with a red border), STA Password, and STA IP Address. The 'SAVE' and 'CANCEL' buttons are still at the bottom of the form, and the green plus button is at the bottom of the screen.

След натискане на Save, устройството вече е записано.

(Фиг 7.4)



7.3 Контролиране на записаните устройства

След като вече сме записали устройството си, можем да започнем да изпращаме команди към него. Първо трябва да се свържем към мрежата на устройството - фиг 7.6. След което сме готови да изпращаме заявки към него. Натискаме върху името на устройството, което отваря неговата страница. Обяснение на всеки елемент на страницата:

Името на устройството

Информация за статуса на релето

Режима в който е устройството

Бутоните за изпращане на заявки към микроконтролера за релето.

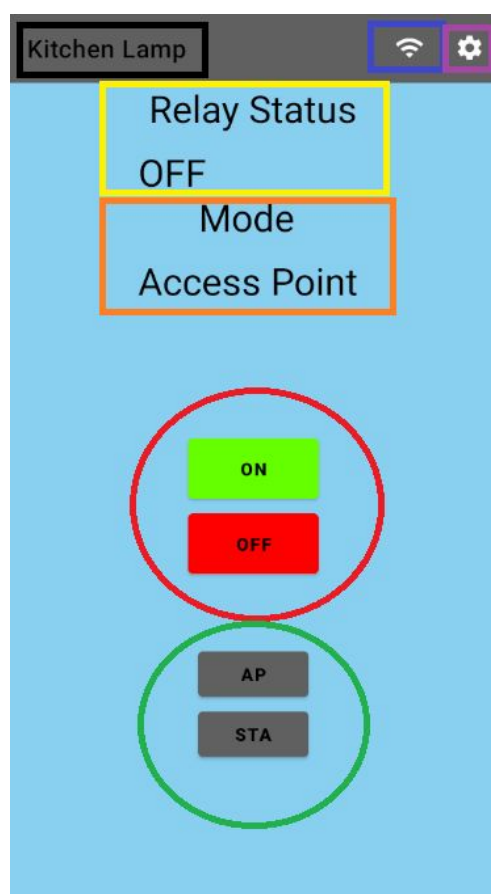
Бутони за промяна на режима на устройството

Бутон за отваряне на Wi-Fi менюто на Android

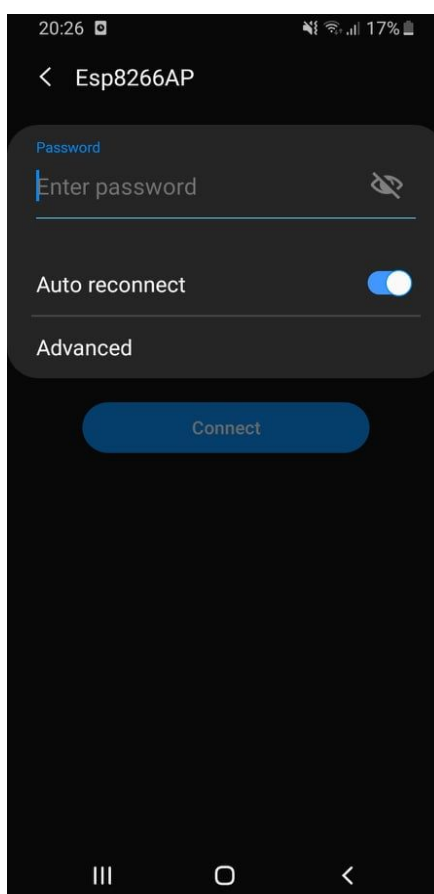
Бутон за отваряне на настройките на устройството

Забелжка: UI-а не е финален и може да претърпи промени.

(Фиг 7.5)



(Фиг 7.6)



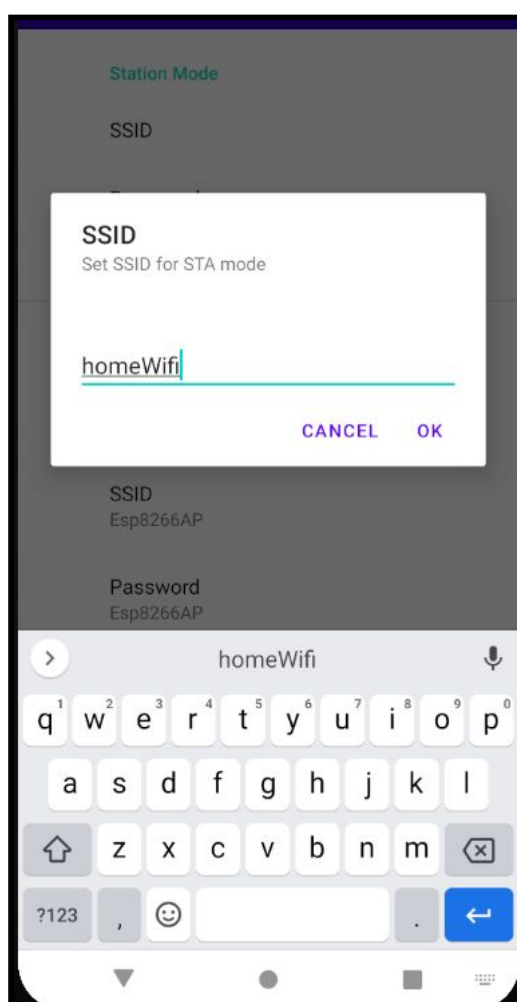
7.4 Промяна на настройките на устройство

За да променим настройките на устройството, влизаме в неговата страница и натискаме бутона за настройки (в лилаво на фигура 7.5).

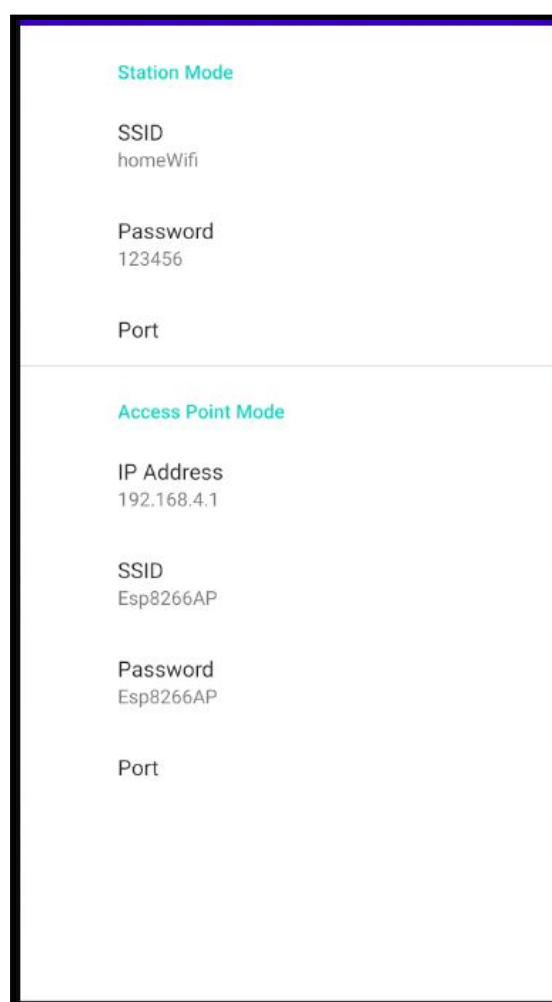
Това отваря следната страница за настройки. За демонстрация ще добавим настройки за работа в мрежов режим на устройството.

Забележка: Настройката за port, не се използва в момента. Тя вероятно ще бъде премахната във финалната версия на проекта.

(Фиг 7.7)

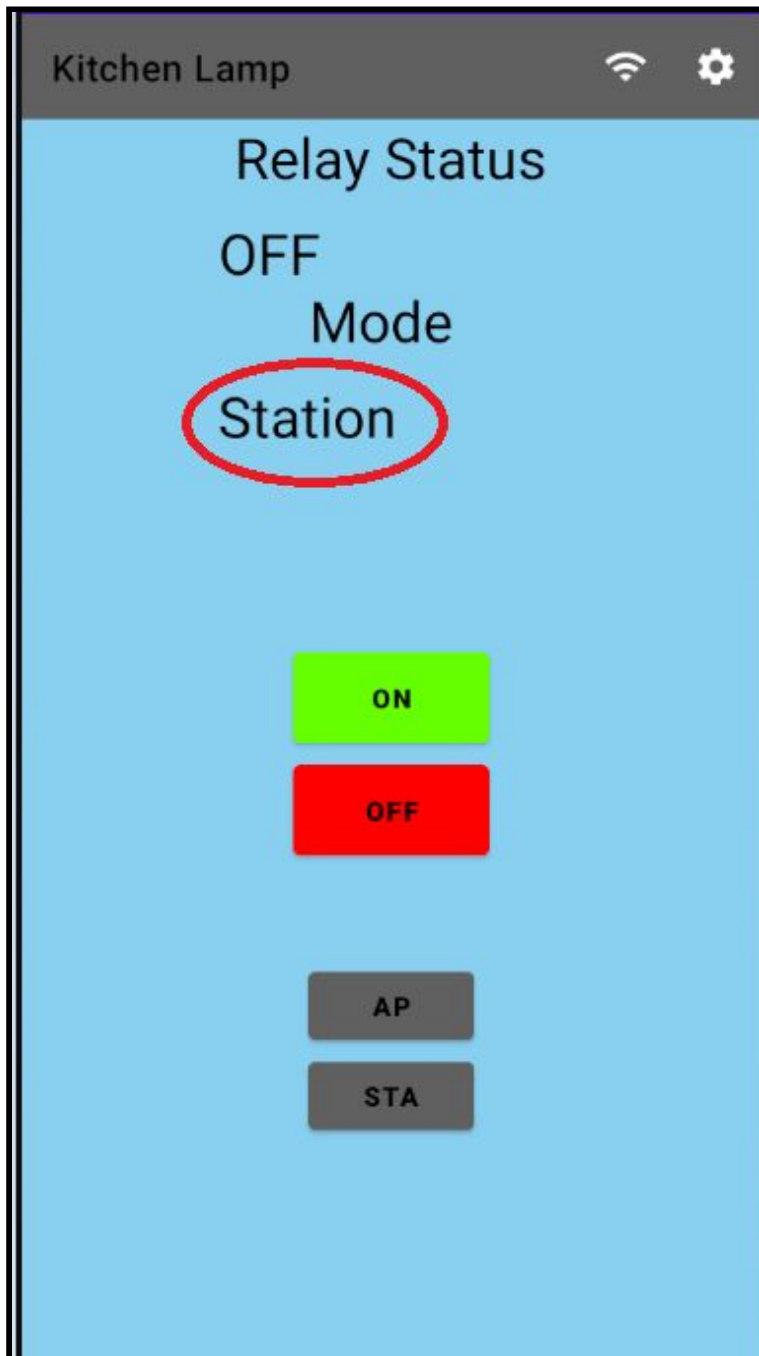


(Фиг 7.8)



След това можем да променим режима на работа на устройството като изпратим заявка чрез бутона "STA" (в зелено на фиг 7.5) Когато го направим може да видим, че това променя индикатора "Mode" (в оранжево на фиг. 7.5) от "Access Point" на "Station". След като **сменяне** на режима, трябва да се свържем мрежата която сме задали с нашето мобилно устройство.

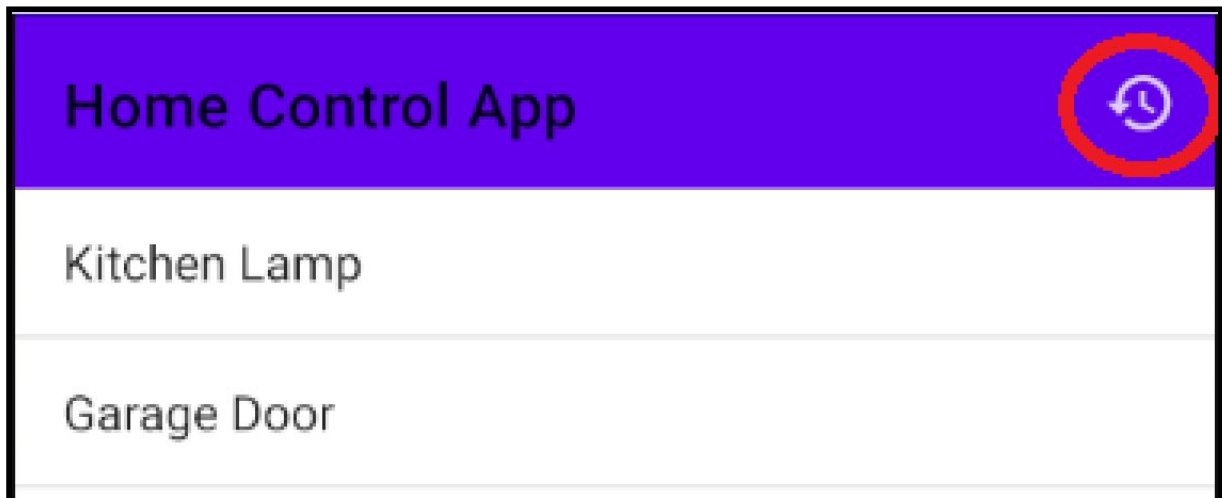
(фиг 7.9)



7.5 Timeline на изпратените команди

За достъпване на timeline-а на изпратените команди, натискаме този бутон в главното меню:

(фиг 7.10)



При натискането му се отваря следната страница, където изпратените команди са визуализирани в формата:

Име на устройството Вид Команда Дата и време

(фиг 7.11)

Kitchen Lamp PM	Relay ON	2/14/21, 8:20
Kitchen Lamp PM	Relay OFF	2/14/21, 8:20
Kitchen Lamp PM	Relay ON	2/14/21, 8:20
Kitchen Lamp PM	Relay OFF	2/14/21, 8:20
Kitchen Lamp PM	Relay ON	2/14/21, 8:20
Kitchen Lamp PM	Relay OFF	2/14/21, 8:20
Kitchen Lamp PM	Relay ON	2/14/21, 8:20
Kitchen Lamp PM	Relay OFF	2/14/21, 8:22
Kitchen Lamp 8:22 PM	Mode to STA	2/14/21,
Garage Door PM	Relay OFF	2/14/21, 8:31
Garage Door	Relay ON	2/14/21, 8:31 PM
Garage Door PM	Relay OFF	2/14/21, 8:31
Garage Door	Relay ON	2/14/21, 8:31 PM
Garage Door PM	Relay OFF	2/14/21, 8:31
Garage Door	Mode to AP	2/14/21, 8:31

ЗАКЛЮЧЕНИЕ

Дипломния проект изпълни главната си задача, като ме накара да се запозная с нови технологии и да изляза от комфортната си зона. Доволен съм от постигнатото, като се има предвид мащаба на задачата и лимитираното време и човешки ресурс. Разбира се разработката е в доста ранен етап и може да се усъвършенства много. Ще продължавам да работя по нея. Ето какви цели съм си поставил за бъдещето на проекта:

1. Да се оправят всички съществуващи проблеми.
2. Да може да се контролира повече аспекти на едно устройство. Например да може да се зададе една круша да свети на 50% от максималното.
3. Да се подобри User Interface-а на приложението.
4. Да се изработят устройства с различна функционалност, които да се контролират от същото приложение. Например охранителна система с датчици за движение.
5. Да се премине към ползване на истинска база данни за приложението, вместо всичко да се пази в SharedPreferences.
6. Да се премине към използване на сървър за комуникацията в мрежов режим.

ИЗПОЛЗВАНА ЛИТЕРАТУРА

няма.

СЪДЪРЖАНИЕ

