

**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ  
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: Home Automation чрез ESP8266 микроконтролер и Android  
приложение.

Дипломант:  
*Росен Тенев*

Научен ръководител:  
*Асен Леков*

СОФИЯ

2021





## **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 човек си мечтае за бъдеще в което дома ще е продължение на него самия. 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. Технологиите разработвани за цел автоматизация на дома са също толкова широко разпространени навсякъде - от животновъдството до тежката промишленост. Потенциалът на автоматизацията на дома в бъдещето е огромен, особено с навлизането на т. нар Internet of Things и със сигурност това ще превърне тази индустрия в една от най-големите. Именно за това проекти свързани с автоматизация на дома са толкова важни.

Тази дипломна работа няма чак толкова мащабни цели колкото тези на гигантите на технологичната индустрия. Няма за цел да се конкурира с подобни продукти на пазара. Целта на тази дипломна работа е най-вече да е полезен на ентузиасти, както и за лична употреба, и да даде един добър старт на система с много потенциал за бъдещо развитие.

## 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, в проекта се използва микроконтролер с Wi-Fi възможности. След проучвателен процес, конкретният микроконтролер, който бе избран е **Esp8266** [5], използващ Esp8266 микрочипа за свързване с Wi-Fi. Този микроконтролер е евтин и лесен за програмиране, 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 фиг. 1.1 показва характеристиките на избрания микроконтролер. Простата конфигурация от микроконтролера и **реле** 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, светодиод и цокъл. Подробно описание на елементната база има в глава шеста.

- 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

**Фиг. 1.1** - спецификации на Esp8266

## **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 той да е използван за 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 те трябваше да се решават без тази помощ.

Големият избор, който трябва да се направи при разработката на Android приложение е този на какъв език да се напише "backend-а" / програмната част. Двата най-популярни езика за разработка на Android са **Kotlin** [1] [2] и **Java** [6]. Тези два езика напълно доминират маркета на Android и да правиш първото си Android приложение на нещо различно от тях би било много лоша идея. Таб. 1.1 и 1.2 показват основните предимства и недостатъци на двата езика. Защо **Kotlin** е избран за езика на Android приложението - управляващ софтуер е обяснено по-подробно в **глава 2.2.2.**

Предимства	Недостатъци
Работил съм с него преди.	Неясно бъдеще за съвместимост с Android
Гигантска общност от разработчици на него	
Безкрайно много онлайн ресурси.	

**Таб. 1.1 - Предимства и недостатъци на Java**

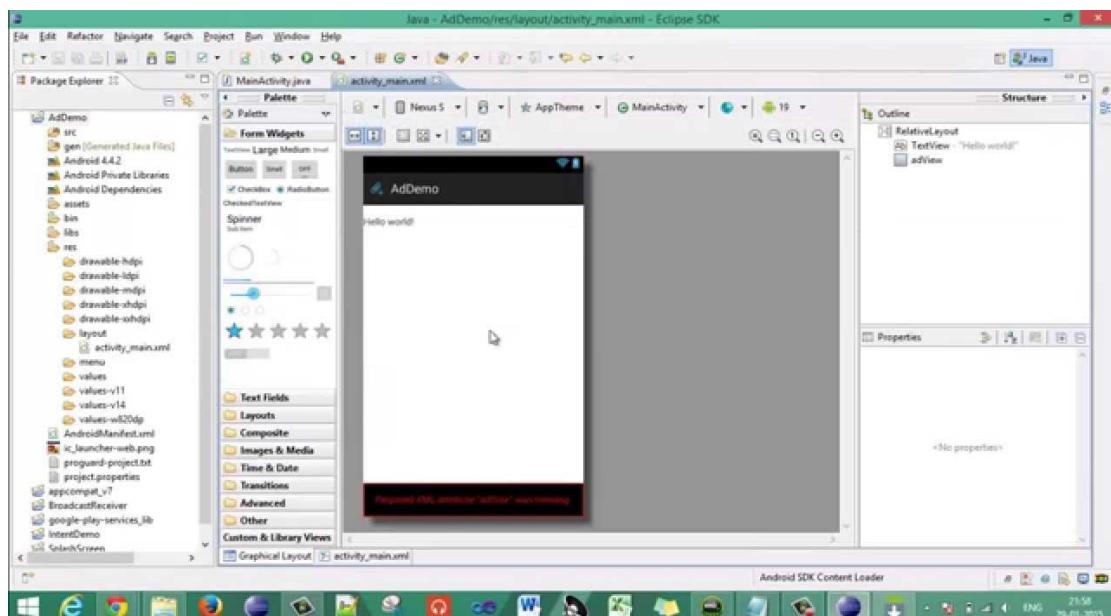
Предимства	Недостатъци
Лесен за научаване	Никога не съм го ползвал
Подобен синтаксис с Java	По-бавно компилиране
Пълна съвместимост с Java	По-малък брой разработчици работят с него.
По-компактен код.	Значително по-малко онлайн ресурси.
Сигурно бъдеще с Android	

**Таб. 1.2 - Предимства и недостатъци на Kotlin**

Следващият избор, които трябва да се направи, 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** се описват предимствата и недостатъците на 4 от най-популярните развойни среди за Android - **Eclipse, Visual Studio Code, IntelliJ IDEA** и **Android Studio** [7]. С помощта на тези предимства и недостатъци е взето решението за проекта да се използва **Android Studio** като главна развойна среда на софтуерната част - "управляващия софтуер" на дипломната работа. По подробно обяснение на този избор е дадено в **глава 2.2.2**.

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

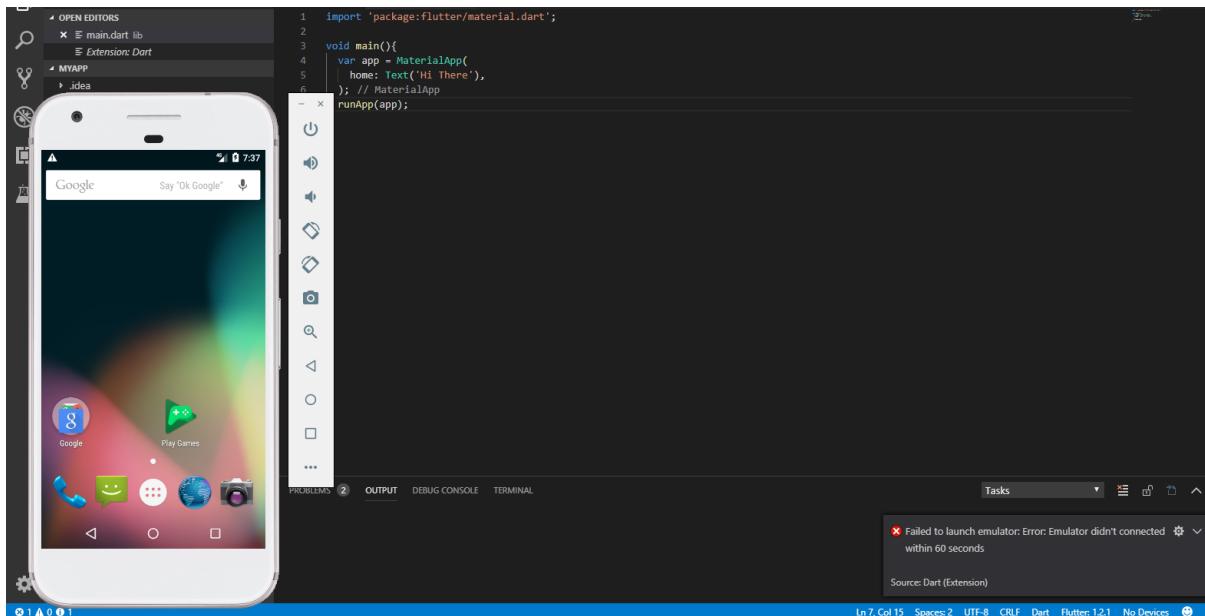
**Таб. 1.3 - Предимства и недостатъци на Eclipse**



**Фиг. 1.2 - Eclipse интерфейс за Android UI development**

Предимства	Недостатъци
Бесплатен	Не съм го ползвал преди
Бърз	Труден setup за работа с Android
Персонализирам	Не предназначен за Kotlin/Java
	Без вграден емулятор

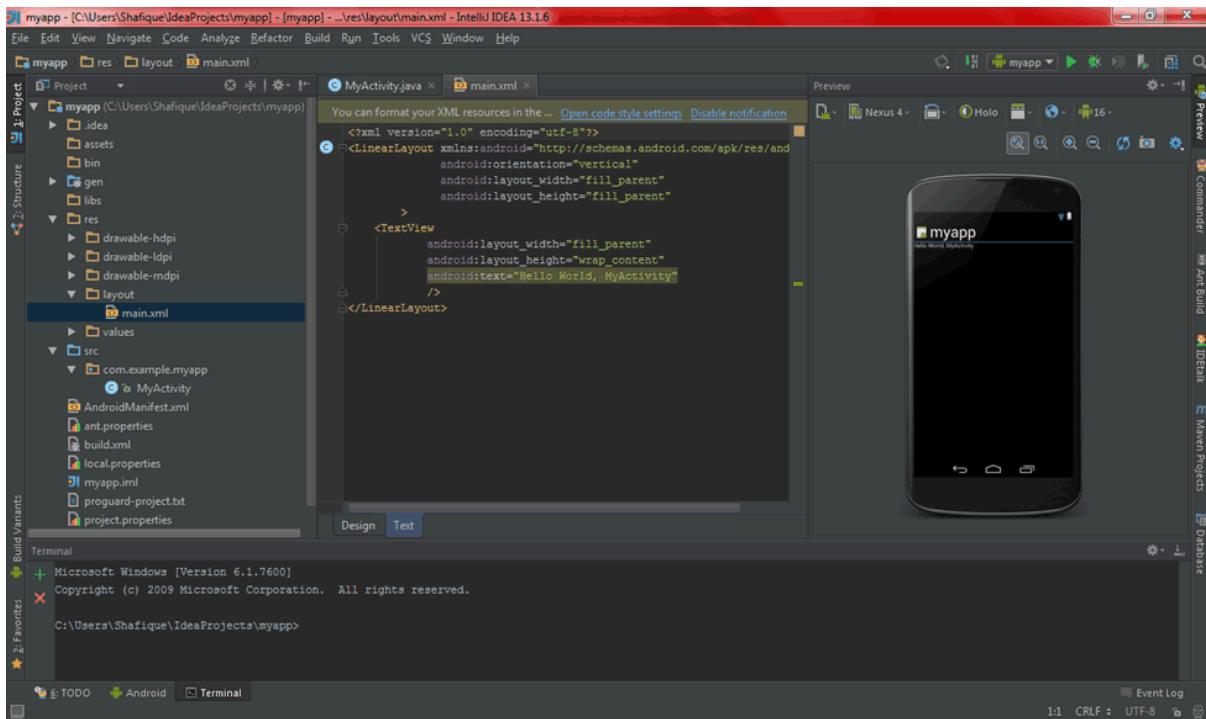
**Таб. 1.4 - Предимства и недостатъци на Visual Studio Code**



**Фиг. 1.3 - Visual Studio Code интерфейс за Android UI development**

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	

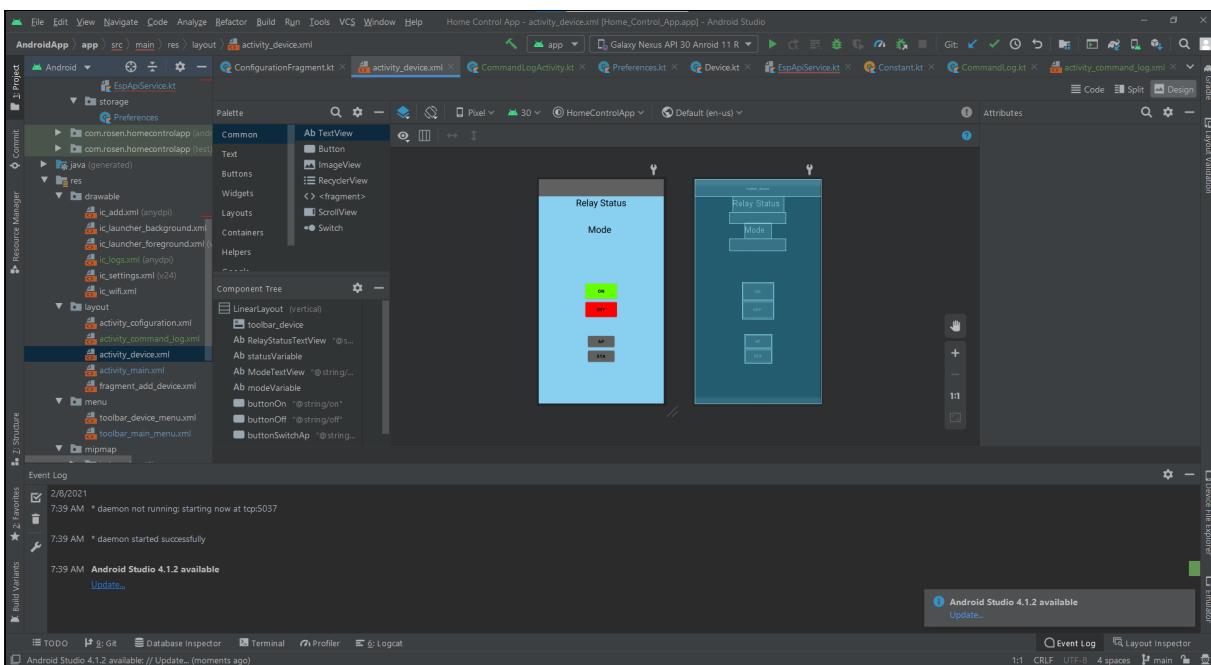
**Таб. 1.5 - Предимства и недостатъци на IntelliJ IDEA**



**Фиг. 1.4 - IntelliJ IDEA интерфейс за Android UI development**

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 емулятор	
Бесплатни ресурси (икони, логота и др.)	
Лесна интеграция с github	

**Таб. 1.6 - Предимства и недостатъци на Android Studio**



**Фиг. 1.5 - Android Studio интерфейс за Android UI development**

### **1.3. Съществуващи решения и реализации.**

Home Automation-а е една много 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, се разглеждат решенията разработени от тези гиганти, както и на по-малки независими компании в сферата на автоматизацията на дома.

#### **1.3.1. Домашни аистенти - Amazon Echo и Google Home**

Сериите от Smartspeaker-и **Amazon Echo** и **Google Home** са хитови продукти - домашни аистенти. Тези устройства се свързват с гласово-контролирани персоналени аистент, съответно "Alexa" и "Google". Функциите на системите включват:

- търсене на мрежата за най-точно отговаряне на гласово-зададени въпроси от всякаакво 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



**Фиг. 1.6 - Amazon Echo**



**Фиг. 1.7 - Google Home**

### **1.3.2. Smart Outlet / Light Switch**

На пазара се предлагат и много устройства с функционалност, близка до тази коята се стреми да се постигне и от дипломния проект. Многобройни по-малки компании разработват контакти и ключове, често контролирани от Android приложение или дори от домашен асистент като тези показани в **глава 1.3.1.** Най-често това се случва посредством wifi или bluetooth. Тези устройства се монтират на мястото на обикновенните ключове или контакти в дома. Функционалностите им варират, като някой от най-често срещаните такива са:

- Включване/изключване посредством приложението.
- Задимяване на светлините посредством приложението.
- Задаване и запазване на режими

**Фиг. 1.8 - Smart Light Switch**



## **ВТОРА ГЛАВА**

### **ПРОЕКТИРАНЕ НА БЛОКОВАТА 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**

За развойна среда за ESP8266 микроконтролера проекта използва **Arduino IDE**, която е съвместима с повечето микроконтролери на пазара. Езика на който е написан управляващия микроконтролер код е **C++**. Проекта от към embedded часта е достатъчно прост, за да няма особени проблеми с Arduino IDE, освен това че понякога качването на кода се случва сравнително бавно, но това е отданено по-скоро на микроконтролера, отколкото на развойната среда. Липсата на autocomplete не беше прекалено осезаема, поради малкия обем код, но това може да се окаже проблем при бъдещо разширяване на управляващия микроконтролера код. Качването на нужните библиотеки за работа с конкретния ESP8266, беше изключително лесно и Arduino IDE, свърши добра работа за проекта, въпреки че е сравнително прост в сравнение с други популярни развойни среди в момента. C++ също не създаде проблеми. 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 полезните ESP8266 библиотеки за Wi-Fi.

## 2.2.2. Android приложение

Изборът за развойна среда и особено език за Android приложението - управляващ софтуер, далеч не е лесен. След дълго обмисляне, избраната за целта комбинацията над която се спрях е 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. Отнема не повече от едно-две онлайн обучителни видеа, не само за да вкараш Android Studio в готовност да компилира код, но дори да направиш пълната инсталация на вградения емулатор, идващ с него. Android Studio прави мениджирането на иначе, доста объркващите пакети в едни Java или Kotlin проект интуитивно. Интерфейса е лесен за научаване и доста подобен с този на другия продукт на компанията която разработва Android Studio - IntelliJ Idea. Auto-complete-а е задължителен за софтуерната част на проекта, не само защото ускорява процеса на писане, а и защото помогна много с отвикването от Java синтаксиса и свикването с новият Kotlin синтаксис, колкото и да са близки те. Качването на приложението на мобилен телефон става за секунди, без сложни маневри и главоболия. Drag and Drop опцията при дизайн на интерфейса на приложението помаг при дизайна на Layout-ите на страниците, особено като се има предвид че го правя за пръв път, а това далеч не е просто без тази функция.

В началото на срока, се очакваше за проекта да се използва Java, поради изградение от дълго ползване комфорт. Не знаех абсолютно нищо за **Kotlin** преди Октомври на 2020. След 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. Разбира се, Google нямат нито възможност, нито интерес от това да спрат внезапно поддръжката на Java-базирани приложения на Android, просто защото това са голям процент от всички. В дългосрочен план, обаче Kotlin изглежда като бъдещето на Android. Въпреки че, използването на Java за тази дипломна работа би било съвсем приемливо, както е описано в увода, този проект има за цел да ме вкара в различни от познатите ситуации и да ме подготви за бъдещето.

Kotlin е съвместим с повечето библиотеки на Java и това че езика е сравнително нов не е осезаем в този аспект. Все пак разликите в

синтаксиса на между Kotlin и Java са достатъчни, за да създаде трудности при смяната от единия към другия След използване на Kotlin, мога да кажа че той въобще не отстъпва на Java. Даже бих казал, че е новият ми любим език. С Kotlin поне за мен, се забелязва по-бързото имплементиране на функционалности, заради по-бързото дебъгване и по-малко проблеми като цяло. Проблемите които създаде 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 осъдна информация.

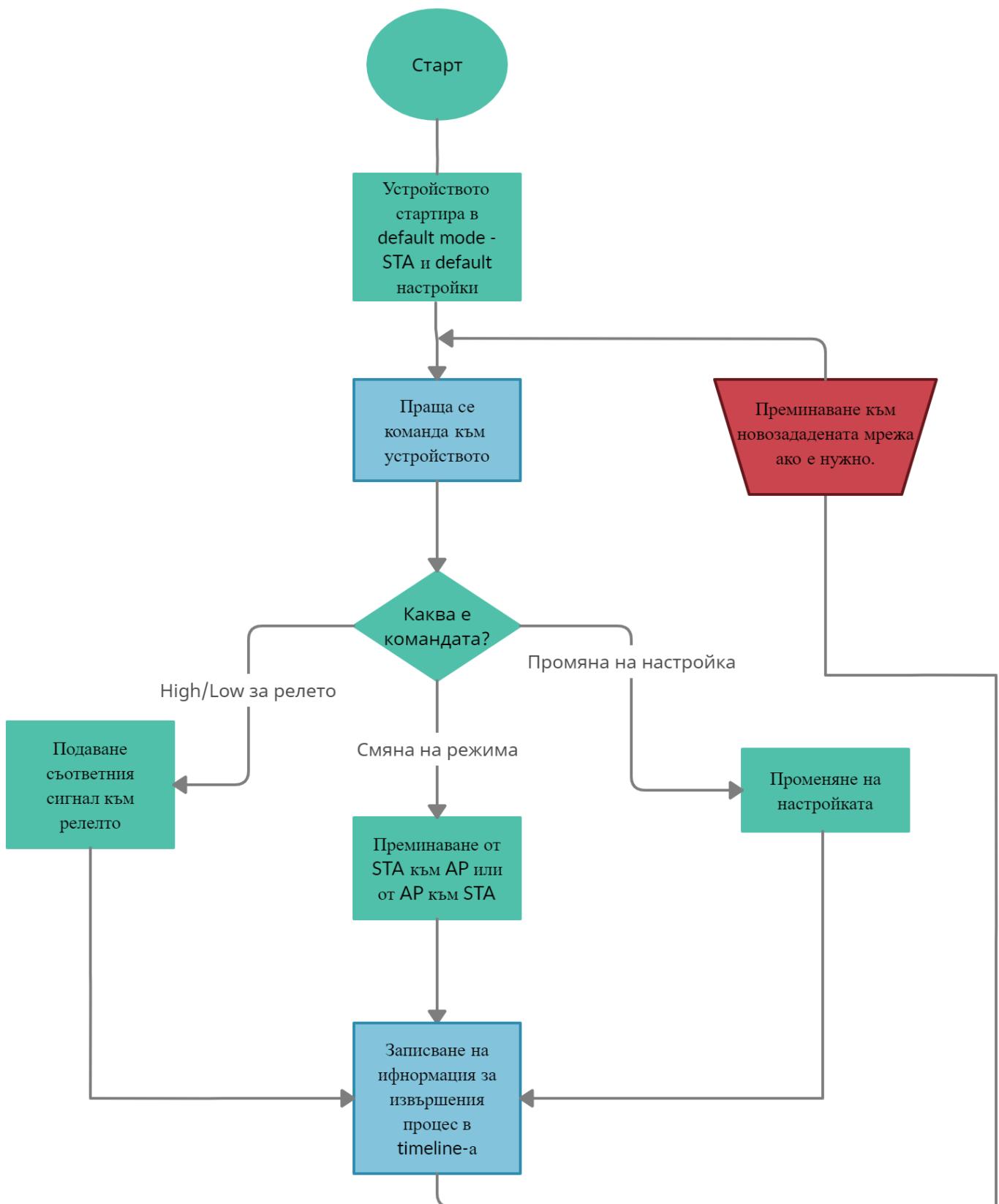
### **2.3. Проектиране на блоковата схема на система за контролиране на устройства от дома**

Тази трета част на втора глава съдържа 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.

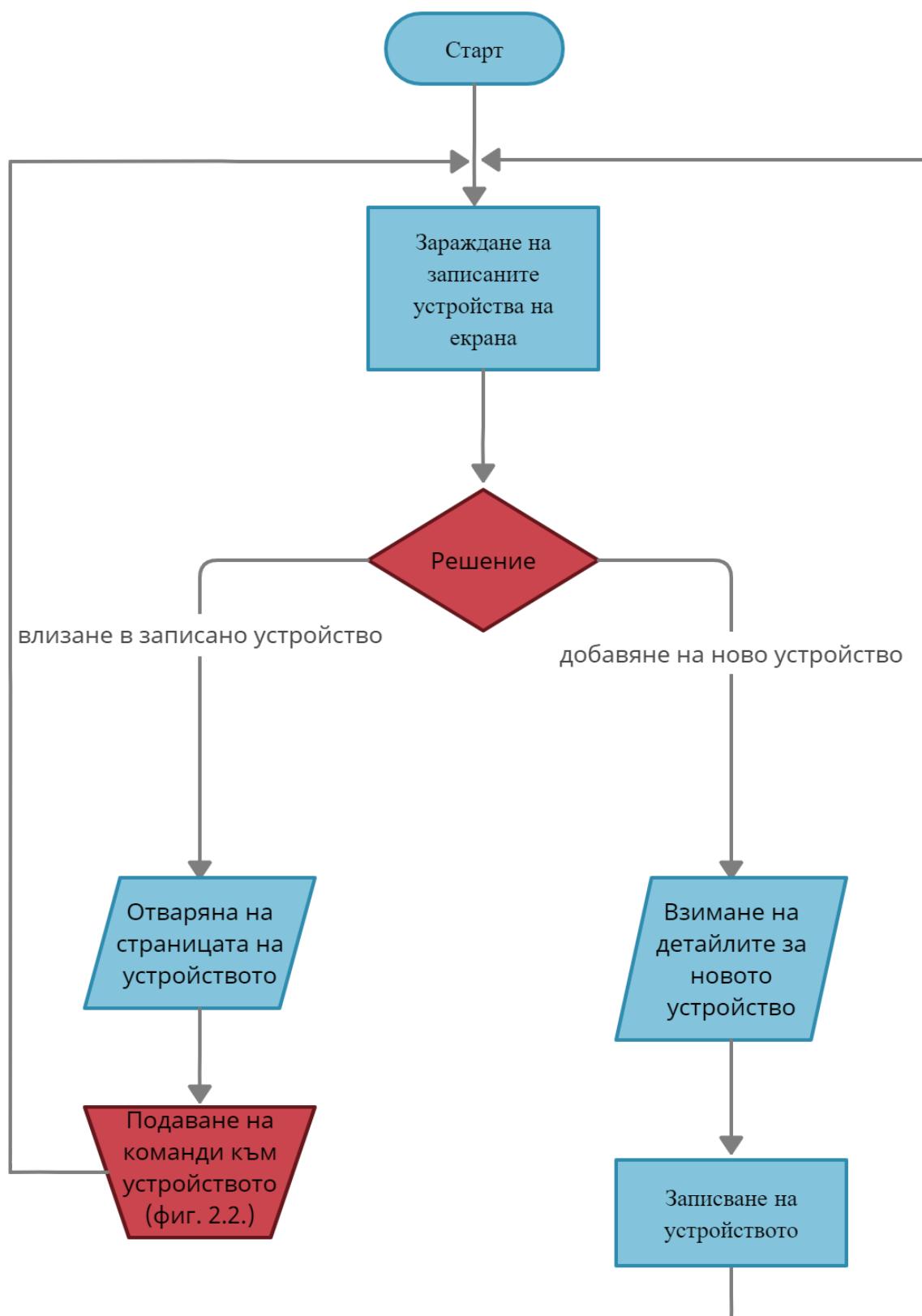
За целта на лесно изобразяване на точно това, в блоковите схеми използвам следната стандартизирана цветово-кодирана **легенда** :

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

**Фиг. 2.1 - цветова легенда на блоковите схеми**



**Фиг. 2.2 - блокова схема за изпращане, изпълняване и записване на команда**

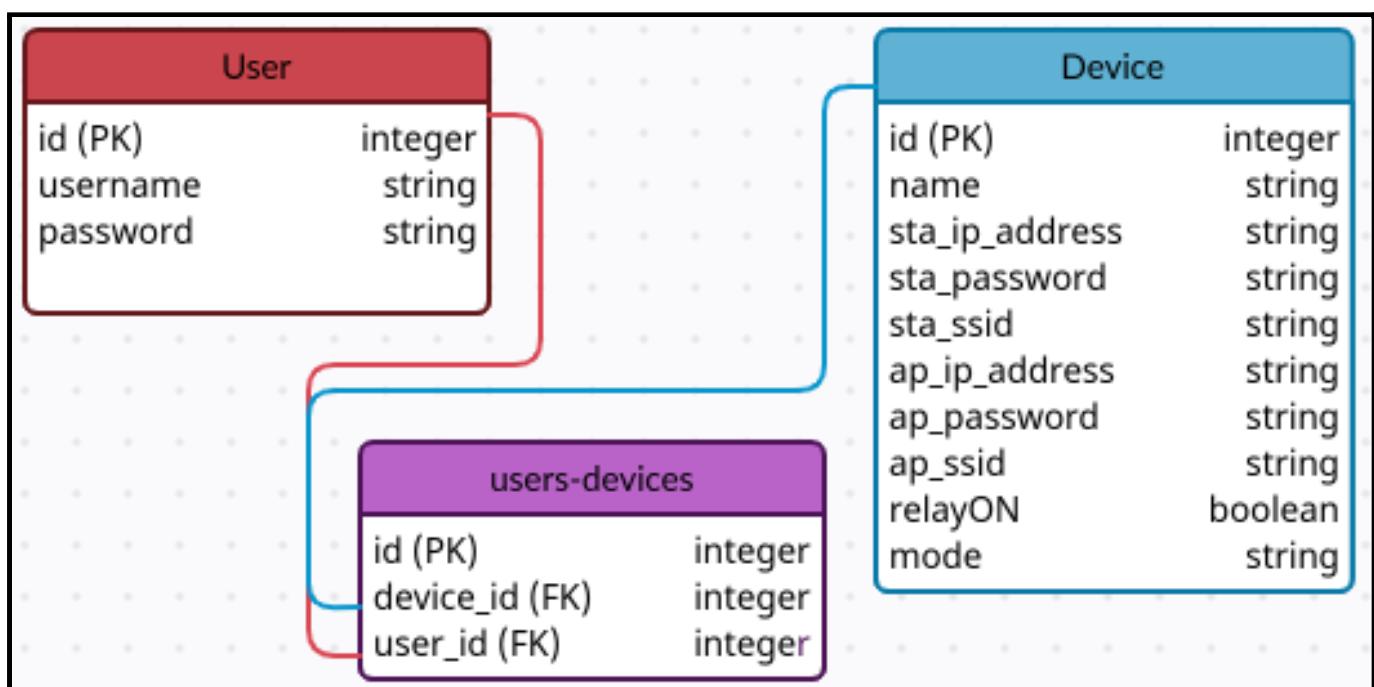


**Фиг. 2.3 - блокова схема на работата на Android приложението**

## 2.4 Проектира на структура на базата данни

Базата данни за този проект, определено не е от най-сложните. За това не използва "истинска" база данни като например sqlite, а съхранява малкото информация, от която се нуждае приложението да функционира в **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. Причината за това е, че 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 се направи акаунт на едно устройство, този акаунт няма да може да бъде достъпен от друго.



**Фиг. 2.3 - модел на базата данни**

## ТРЕТА ГЛАВА

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

Embedded частта на проекта се състои от ESP8266 микроконтролер, свързан с реле и съответното устройство, което да контролираме дистанционно. Свързването на микроконтролера и релето може да се случи директно само с проводник, но за да се свържат двата елемента с устройството, което да трябва да контролират, трябва да се състави по-сложна схема. В тази глава е описано как работи схемата и как работи управляващия код на микроконтролера.

### 3.1. Принципна електрическа схема.

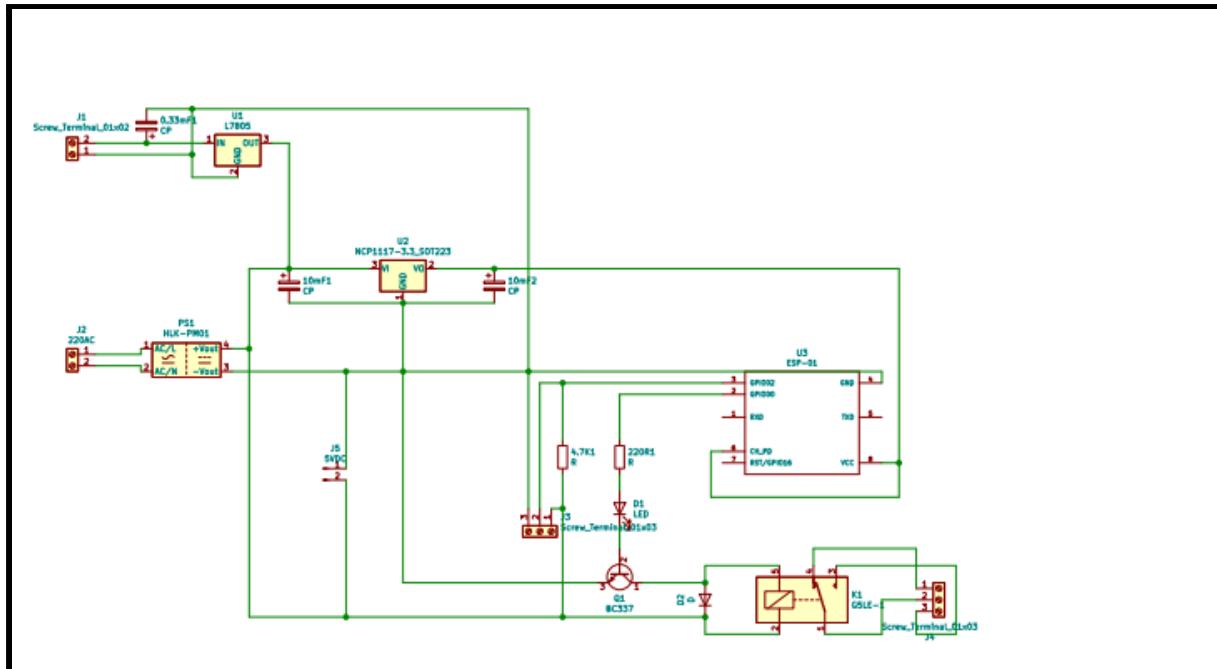
**Входна клема J2** - входното напрежение - 220V AC се подава към модул **HLK-PM01** преобразуващ го към 5V DC. Кондензатора **F1**, стабилизира напрежението. Тези 5 волта захранват:

-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 се използва ESP Wi-Fi модула на микроконтролера за свързване в мрежа или за той да създава собствена мрежа, проекта използва стандартната **ESP8266WiFi** библиотека. Тя дава множество методи, които улесняват свързването към съществуваща мрежа или създаването на такава (access point mode).

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

### **3.3 Изработка на тестова версия на устройството.**

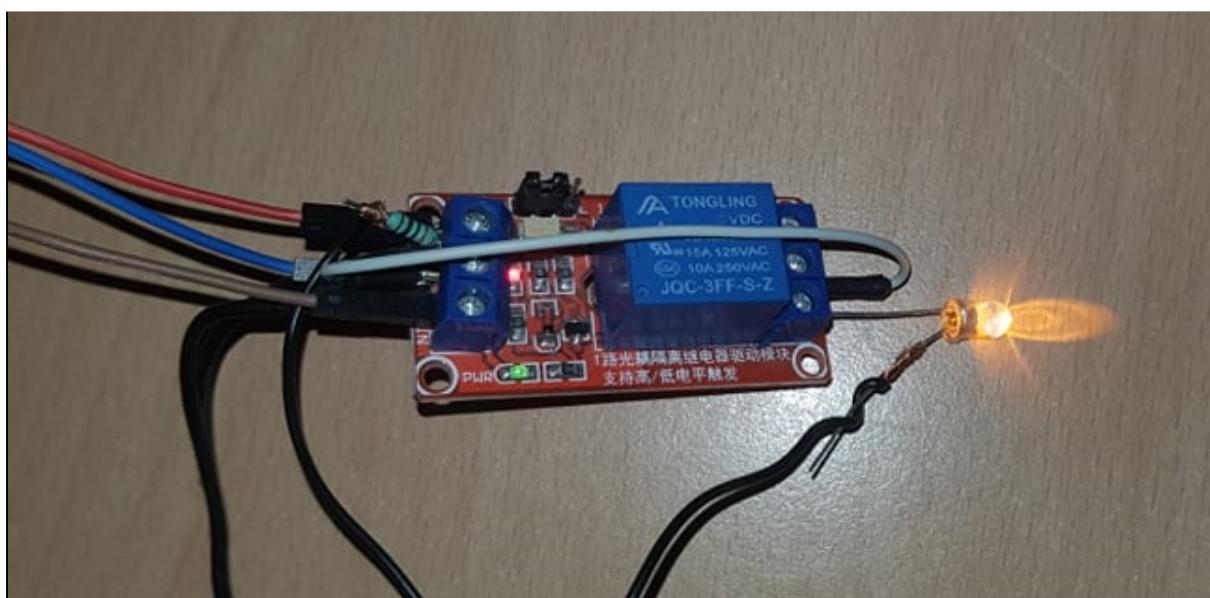
Голяма част от цикъла на разработка на проекта, се използваше прототипна версия на устройството.

Първоначалната версия беше само микроконтролера. Той е напълно достатъчен за работа в мрежов и стационарен режим. Използваше се вградения led (ограден в червено) на микроконтролера, за да му се изпращат команди от Android приложението, за подаване на high или low към пина на релето.



**Фиг. 3.2** - първата тестова версия на устройството - *Esp8266* микроконтролер.

Впоследствие, към микроконтролера беше свързано реле с led диод. Тази прототипна версия на устройството беше използвана през голяма част на разработката.



**Фиг. 3.3** - втора тестова версия на устройството

### 3.4 Написване на управляващия микроконтролера код.

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.

```
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);
```

**Фиг 3.4 - от embedded/sta\_and\_ap\_mode**

## 2. Стартова конфигурация в setup-а:

Задава се променливата, отговаряща на пина за релето като output и му се подава LOW, което практически означава че уреда който управлява е изключен. Задава се микроконтролера да работи в стационарен режим с WiFi.mode. Конфигурира се ip, gateway и netmask с WiFi.softApConfig и се пуска мрежата с WiFi.softAP с ssid - ssid\_ap и парола - password\_ap.

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

**Фиг. 3.5** - от *embedded/sta\_and\_ap\_mode*

## 3. Все още в setup-а, се задава 3-те endpoint-а, който сървъра да слуша със server.on (обяснение за какво се използва всеки от тях, в следващите точки от главата). Стартира се сървъра с server.begin. В loop единствено се използва метода handleClient на сървъра, както е описано в примерите от

```
server.on("/led", LedSwitch);
server.on("/switch", ModeSwitch);
server.on("/config", ConfigSwitch);
server.begin();

}

void loop(void) {
    server.handleClient();
}
```

документацията на **ESP8266WebServer** библиотеката.

**Фиг 3.6 - от embedded/sta\_and\_ap\_mode**

4. Endpoint-а, контролиращ релето е /led. Асоциираната с него функция е **LedSwitch**. Заявките от Андроид 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.

```
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");  
    }  
}
```

**Фиг. 3.7 - от embedded - sta\_and\_ap\_mode**

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".

```

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");
    }
}

```

**Фиг 3.8 - от embedded - sta\_and\_ap\_mode**

6. Endpoint-а, 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", която е ключовата дума, избрана да означава, задаване на мрежата да е без парола. След това сменя паролата на дадената, ако предишните две не са изпълнени.

```
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. 3.9 - 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 = "";
}
```

**Фиг 3.10 - от embedded - sta\_and\_ap\_mode**

## ЧЕТВЪРТА ГЛАВА

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

Изграждането на ефективен и интуитивен управляващ софтуер е не по-малко важна задача от изработването на функционално устройство. За това голяма част от времето за разработка е посветена за него. Подточките на тази глава описват етапите на който е разделена извършената работа по приложението. На моменти 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. За тази цел, приложението използва **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, е имплементиран 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 справяне с този проблем, при създаване на Preference обект, му се предава context-а на Activity-то, от което е създаден.

**preferences** - Инстанция на SharedPreferences. За извикване на инстанция на SharedPreferences с default настройки се използва ctx.

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

**Фиг. 4.1** - от *com.rosen.homecontrolapp.storage - Preferences*

За добавяне на нова или промяна на съществуващ запис в SharedPreferences, е имплементиран метода getSharedPreferencesValue в Preferences. Той приема обект от помощния data class Storage (фиг. 4.2). За да се променят стойностите в SharedPreferences обект, се вика инстанция на SharedPreferences. Editor обекта чрез .edit(), задава се стойността с .put() и промените се запазват с .apply() (фиг. 4.3)

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

**Фиг. 4.2** от *com.rosen.homecontrolapp.model - Storage*

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

**Фиг. 4.3** от *com.rosen.homecontrolapp.storage - Preferences*

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 трябва да се достъпи. Достъпването се случва чрез извикване на SharedPreferences.getString(), с аргументи полетата в storage обекта.

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

**Фиг. 4.4** от *com.rosen.homecontrolapp.storage - Preferences*

Конкретните параметри, които се пазят в SharedPreferences са показани на фиг. 4.5. 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 всеки един от записите в SharedPreferences е описано в следната таблица:

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.

**Таб. 4.1 - обяснение на параметрите в SharedPreferences**

```

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: "")

```

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

За удобство, са имплементирани методите от фиг. 4.6. Тяхната функция е да улеснят извлечането на данни за активното устройство. Имплементацията им м е чрез getSharedPreferencesValue (фиг. 4.3) с аргумент един от обектите от Constant.kt (фиг 4.5)

```
fun getAddressSta(): 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 getAddressAp(): 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)
```

**Фиг. 4.6** от com.rosen.homecontrolapp.storage - Preferences

## **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 едениично, *hardcode*-нато устройство.

Класът модел за устройството е `Device`, намиращ се в пакета `com.rosen.homecontrolapp.model`. 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 се използва метода `getDevices`. Метода работи като взима записи 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>`, който е удобен за работа.

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

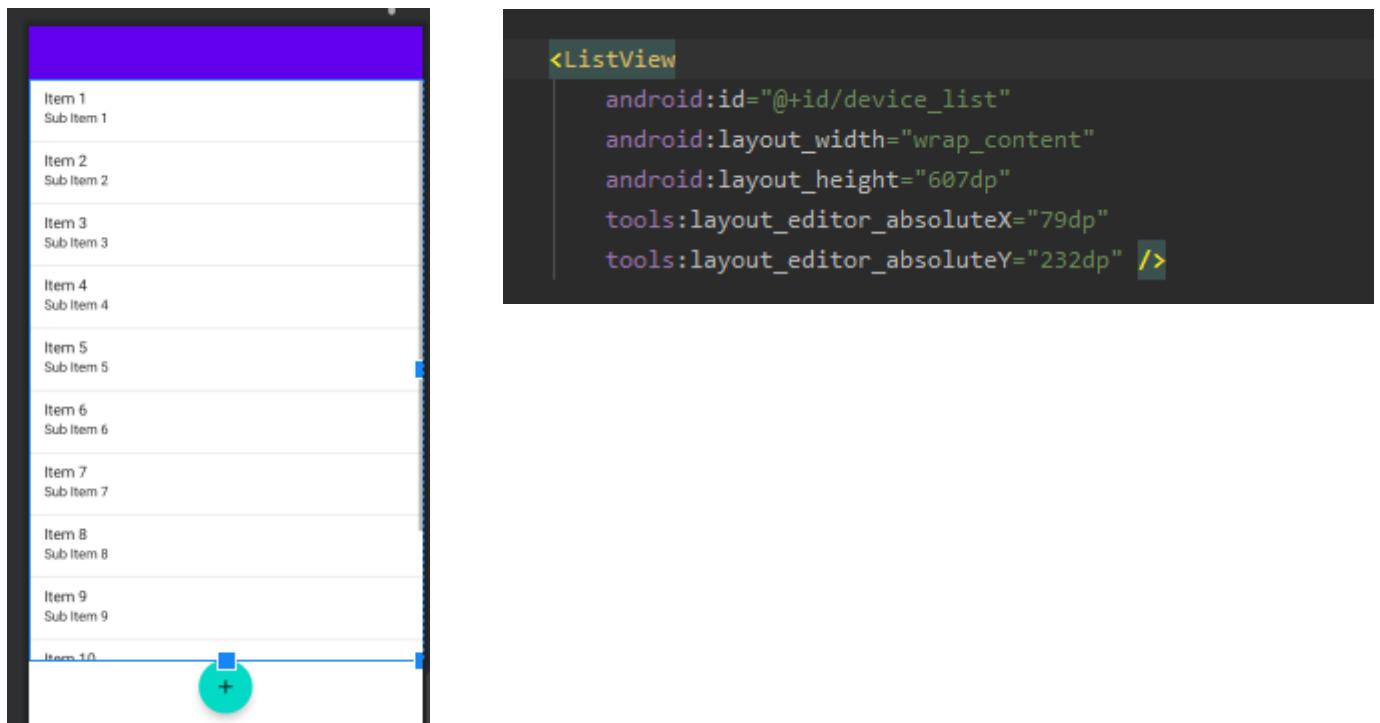
**Фиг. 4.7** - от `com.rosen.homecontrolapp.activity` - `MainActivity`

2. Създава се 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, взет в миналата стъпка, чрез stream се извличат имената на устройствата подредени в ArrayList<String> - deviceNames. Създава се ArrayAdapter обект и имената на устройствата се зареждат в него, след което той се задава като адаптер на listView-то.

```
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. 4.9 - от com.rosen.homecontrolapp.activity - MainActivity**

3. За финал се прави така, че изписаните имена на устройства, да могат да бъдат натискани и това да води до страницата на устройството. Това става, чрез setOnItemClickListener. Взима се името на избраното устройство. Намира се в списъка от устройства и id-то му се изпраща на класа отговарящ за визуализацията на отделните устройства - deviceActivity чрез метода putExtra на обекта Intent. Трета част на главата описва за какво се използва изпратеното id.



**Фиг. 4.10 и 4.11 - от res/layout - activity\_main.xml**

```
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)
}
```

**Фиг. 4.12** от com.rosen.homecontrolapp.activity - MainActivity

**Добавянето** на ново устройство се случва по следния начин:

- Създава се инстанция на floatingActionButton - addBtn, отговаряща на бутона от activity\_layout.xml с id = addDeviceButton. Задава се setOnItemClickListener-а на бутона, който отваря диалог - AddDeviceDialog (фиг 4.13).

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

**Фиг. 4.13** - от com.rosen.homecontrolapp.activity - MainActivity

- Създава се инстанция на Preferences, за да се заредят запазените в SharedPreferences devices обекти чрез getDevices (обяснен в глава 4.1). Задава се view-то на диалога да е fragment\_add\_device.xml.

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

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

**Фиг. 4.14** - от com.rosen.homecontrolapp.activity - AddDeviceDialog

3. Създават се обекти за бутонаите и текстовите полета, отговарящи на тези в fragment\_add\_device.xml.

```
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.StaSSIDEEditText)
val StaPassword = rootView.findViewById<EditText>(R.id.StaPasswordEditText)
val StaIp = rootView.findViewById<EditText>(R.id.StaIPEditText)
val ApSSID = rootView.findViewById<EditText>(R.id.ApSSIDEEditText)
val ApPassword = rootView.findViewById<EditText>(R.id.ApPasswordEditText)
val ApIp = rootView.findViewById<EditText>(R.id.ApIPEditText)
```

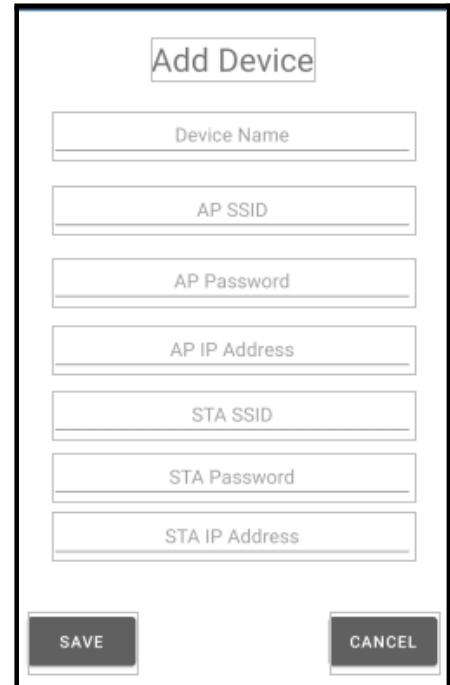
**Фиг. 4.15 - от com.rosen.homecontrolapp.activity - AddDeviceDialog**

4. Задава се onClickListener-и на бутона cancelBtn, който затваря диалога и на saveBtn, който създава нов Device обект с полетата зададени от потребителя (фиг 4.17).

```
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
```



**Фиг. 4.16 - от AddDeviceDialog**

**Фиг. 4.17 - от add\_device\_fragment**

5. Запазва се новия Device в локалната инстанция на Devices. Запазват се локалните промени по Devices в SharedPreferences чрез saveDevices метода на Preferences, който е имплементиран 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 changeSharedPreferenceValue (обяснен в глава 4.1)

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

**Фиг. 4.18** - от com.rosen.homecontrolapp.storage - Preferences

#### **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 подаденото от MainActivity device id. Запълва празните textView-ата с името, режима и състоянието на релето на устройството, което отговаря на това id. Също записва настройките на устройството в SharedPreferences чрез Preferences.loadDevice. Това се прави заради следващата част от главата - промяна на настройките на устройството.

```
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. 4.19 - 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)
}

```

**Фиг. 4.20** - от com.rosen.homecontrolapp.activity - DeviceActivity

- Създават се бутона обектите, отговарящи на тези от layout-а на страницата. Задавам onClickListener-и, за функционалността им.

```

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)

```

**Фиг. 4.21** - от com.rosen.homecontrolapp.activity - DeviceActivity

За **заявките** към устройството се използва библиотеката Retrofit2 [4].

```

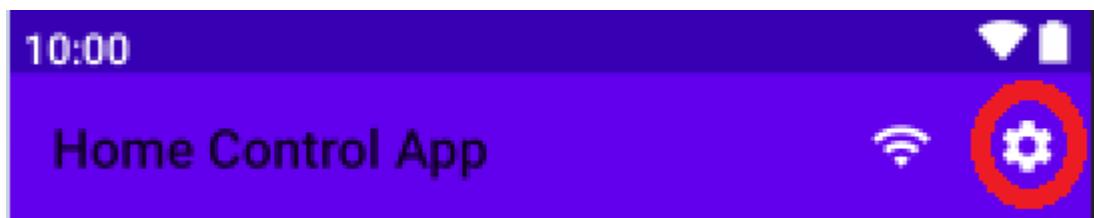
@GET( value: "/led")
fun ledState(@Query( value: "state") ledState: String): Call<Void>
@GET( value: "/switch")
fun switchMode(@Query( value: "mode") newMode: String): Call<Void>
@GET( value: "/config")
fun switchConfig(@Query( value: "ssid_ap") ssidAp: String?,
                 @Query( value: "password_ap") passwordAp: String?,
                 @Query( value: "port_ap") portAp: String?,
                 @Query( value: "ssid_sto") ssidSto: String?,
                 @Query( value: "password_sto") passwordSto: String?,
                 @Query( value: "port_sto") portSto: String?): Call<Void>

```

**Фиг. 4.22** - от com.rosen.homecontrolapp.service - EspApiService

#### **4.4 Промяна на настройките на устройството.**

Промяната на настройките на устройството се случва по сходен начин - чрез заявки към устройството. 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. editTextPreferences полетата в него са директно свързани със съответстващите на тях записи в SharedPreferences. Настройките на активното устройство вече са заредени в SharedPreferences още от миналата част чрез Preferences.loadDevice метода. Това се прави за да се използва една preferences.xml страница за множество устройства и техните различни настройки.

```
<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" />
```

**Фиг. 4.24** от *res/xml preferences.xml*

При **промяна** на текста в тези полета, се вика `onSharedPreferenceChanged` 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`.

```
override fun onSharedPreferenceChanged(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)!!
        }
    }
}
```

**Фиг. 4.25 - com.rosen.homecontrolapp.activity - ConfigurationActivity**

## 4.5 Записване и визуализация на изпратените команди в timeline

```
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"  
    }  
}
```

**Фиг. 4.26 - от com.rosen.homecontrolapp.model - CommandLog**

**deviceName** - името на устройството асоциирано с командата

**command** - вида на командата

**timestamp** - времето на изпращане на командата

**get\_info()** - улеснява визуализация

**Визуализация:** Запазените в SharedPreferences команди се зареждат чрез метода getLogs, подобно на част две на тази глава. Информацията от командите се зареждат в ListView-то на страницата.

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

**Фиг. 4.27 - от com.rosen.homecontrolapp.activity - CommandActivity**

**Записване:** Всеки път когато се праща команда към устройството, тя се записва в SharedPreferences чрез метода saveLog.

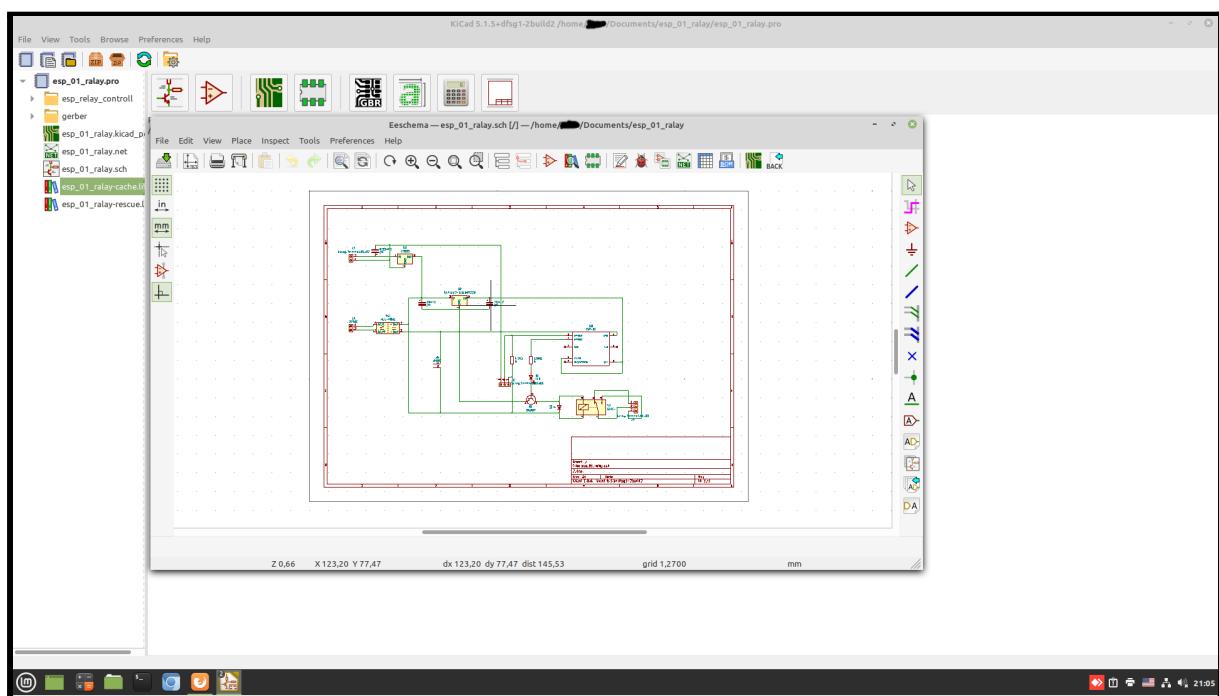
```
btnON.setOnClickListener { it: View!  
    espConnector.sendLedRequest(state = "on")  
    device!!.relayOn = true  
    relayStatusTextView.setText("ON")  
    preferences.saveDevice(device)  
    preferences.addLog(CommandLog(deviceName = device!!.name, command = "Relay ON"))  
}
```

**Фиг. 4.28 - от com.rosen.homecontrolapp.activity - DeviceActivity**

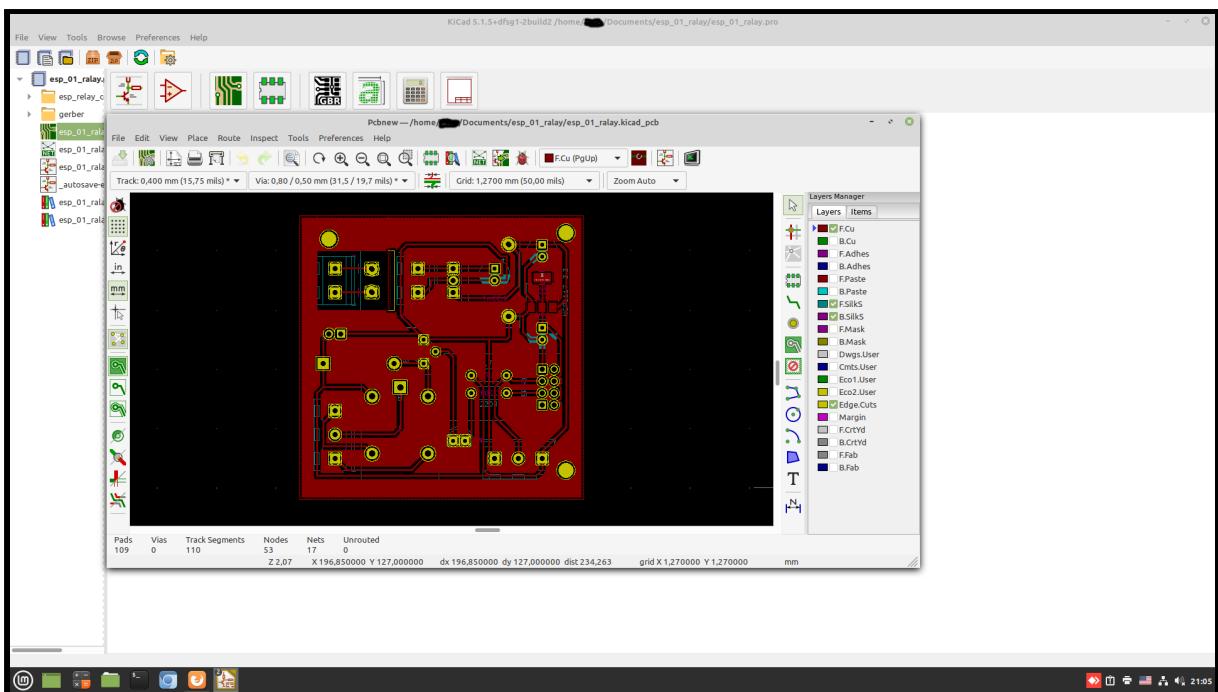
## ПЕТА ГЛАВА

### 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 бесплатен пълен лиценз за комерсиална и некоментриална работа. Софтуера е напълно стандартен и има всички нужни системи за създаването на схемата по която е направена печатната платка.



**Фиг 5.1 - проектираната схема в KiCad**



**Фиг 5.2 - проектираната схема в KiCad**

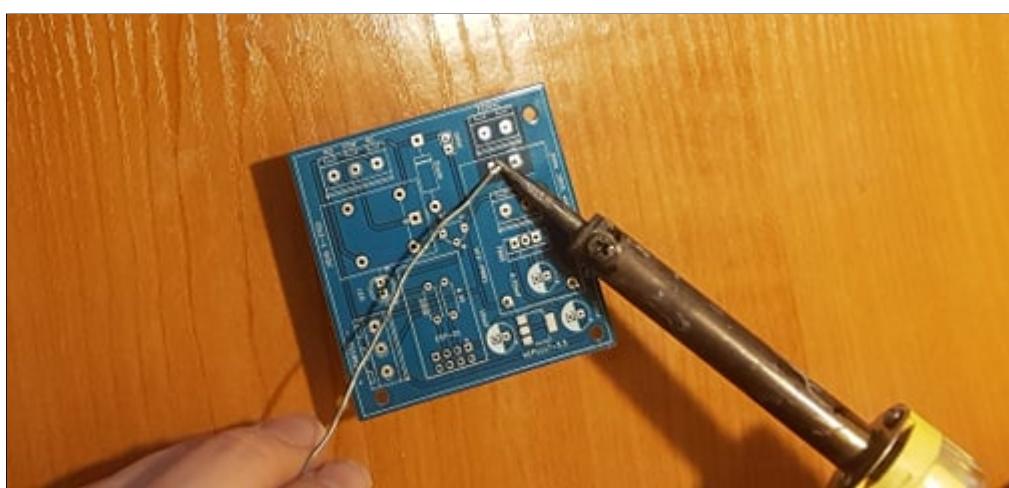
## ШЕСТА ГЛАВА

### 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)
- регулятор на напрежение - вход 5V DC, изход 3.3V DC
- 2 кондензатора 10 $\mu$ 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. Също така схемата може да се захрани и с ниско напрежение 12-24VDC.



**Фиг. 6.1** - печатната платка



**Фиг. 6.2** - устройството в действие



**Фиг. 6.3** - устройството в действие

## **СЕДМА ГЛАВА**

### **РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ**

#### **7.1 Инсталация на софтуера за управление на устройството. Софтуерни изисквания.**

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

Изискванията към устройството са единствено то да е на Android OS като е препоръчително той да е над версия 4.0, което включва почти всички устройства на направени след 2012. Но приложението работи на по-стари версии на Android.

#### **7.2 Добавяне на устройства**

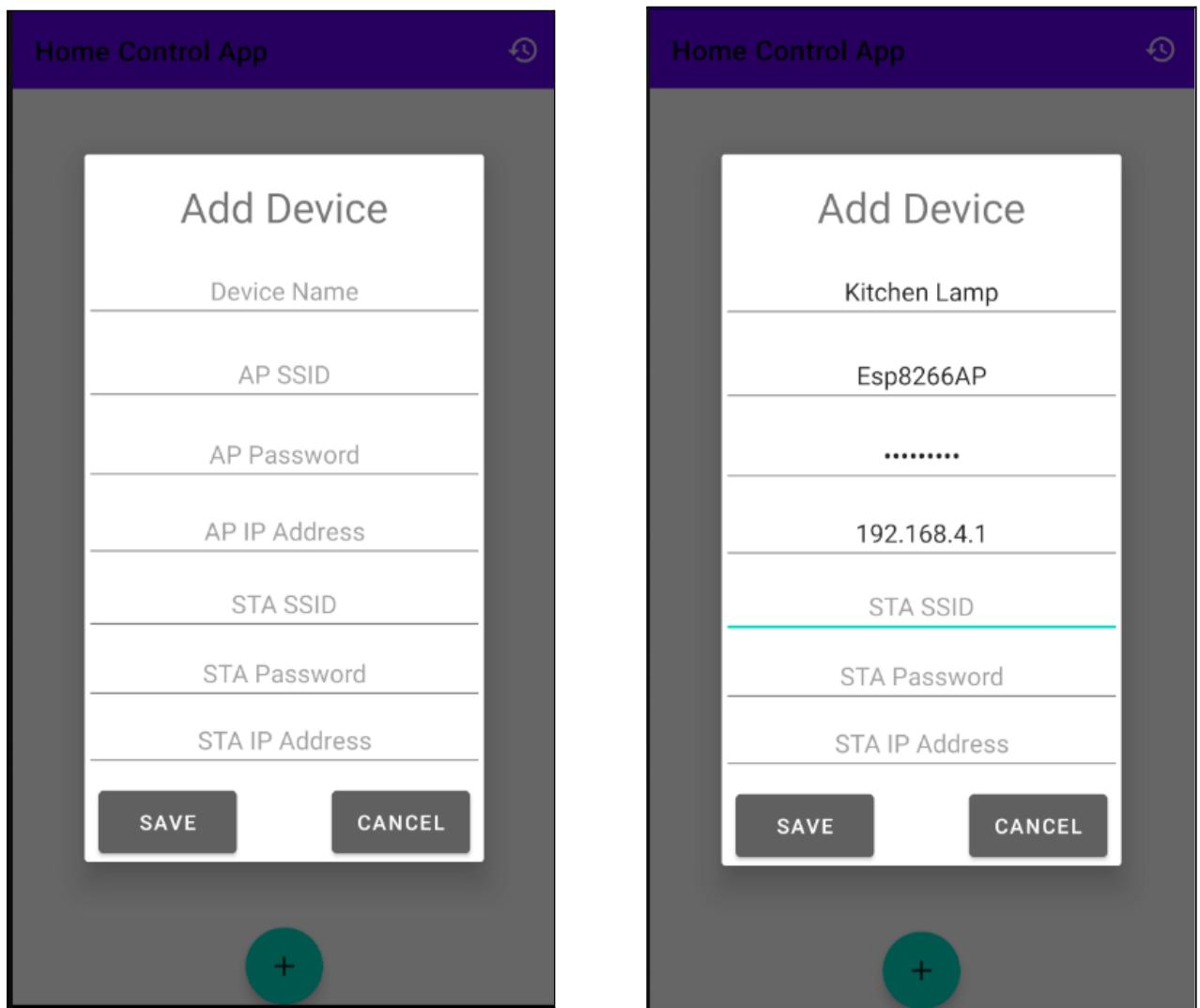
След като инсталиранието на приложението, трябва да добавим устройството си. При пускане на приложението за пръв път, началният екран ще изглежда по следния начин:



**Фиг 7.1 - главно меню**

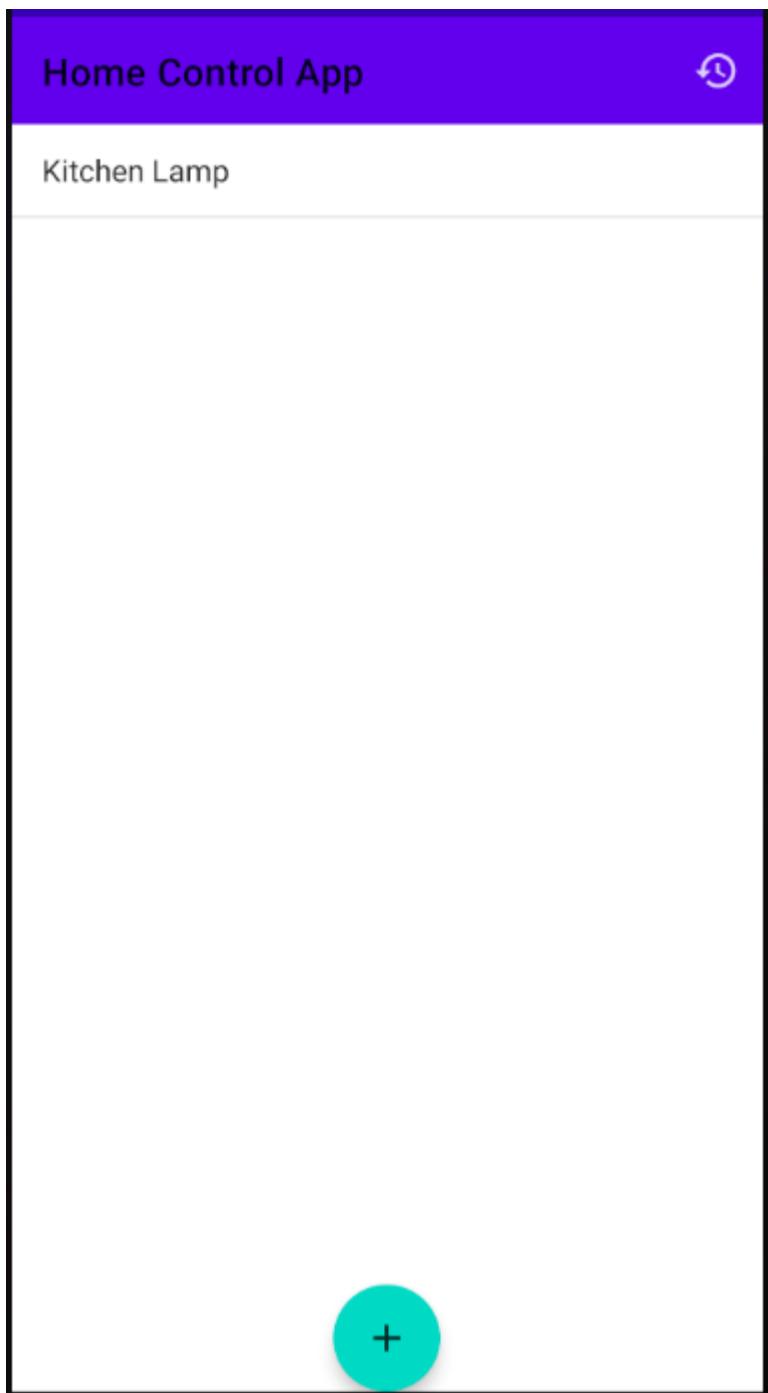
За добавяне на устройство, се натиска зеления бутон, което отваря диалога за добавяне на устройство, който има полета за попълване. Не е задължително всички полета да се попълнят. Това може да се случи и след добавянето на устройството чрез настройките му. Зададените начални настройки на устройството 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 hardcoded-нато.



**Фиг. 7.2 и 7.3** - меню за добавяне на устройство

След натискане на Save, устройството вече е записано.



**Фиг. 7.4** - главното меню след добавяне на устройство

### 7.3 Контролиране на записаните устройства

След като вече сме записване на устройството, към него вече могат да започнат да се изпращат команди. Първо, обаче трябва да се мобилното устройство да е свързано с мрежата - фиг 7.6. 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

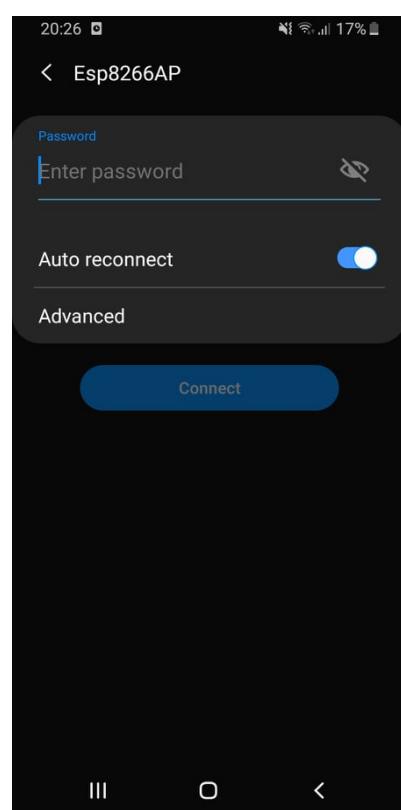
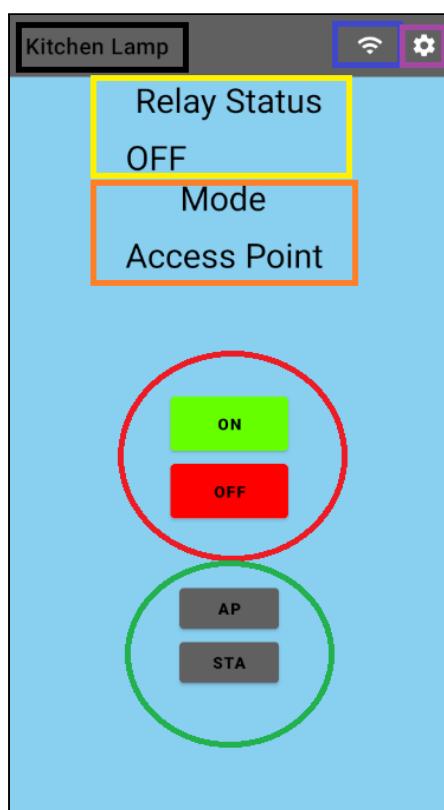
**Бутоните за изпращане на заявки към микроконтролера за релето.**

**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** - меню на устройството

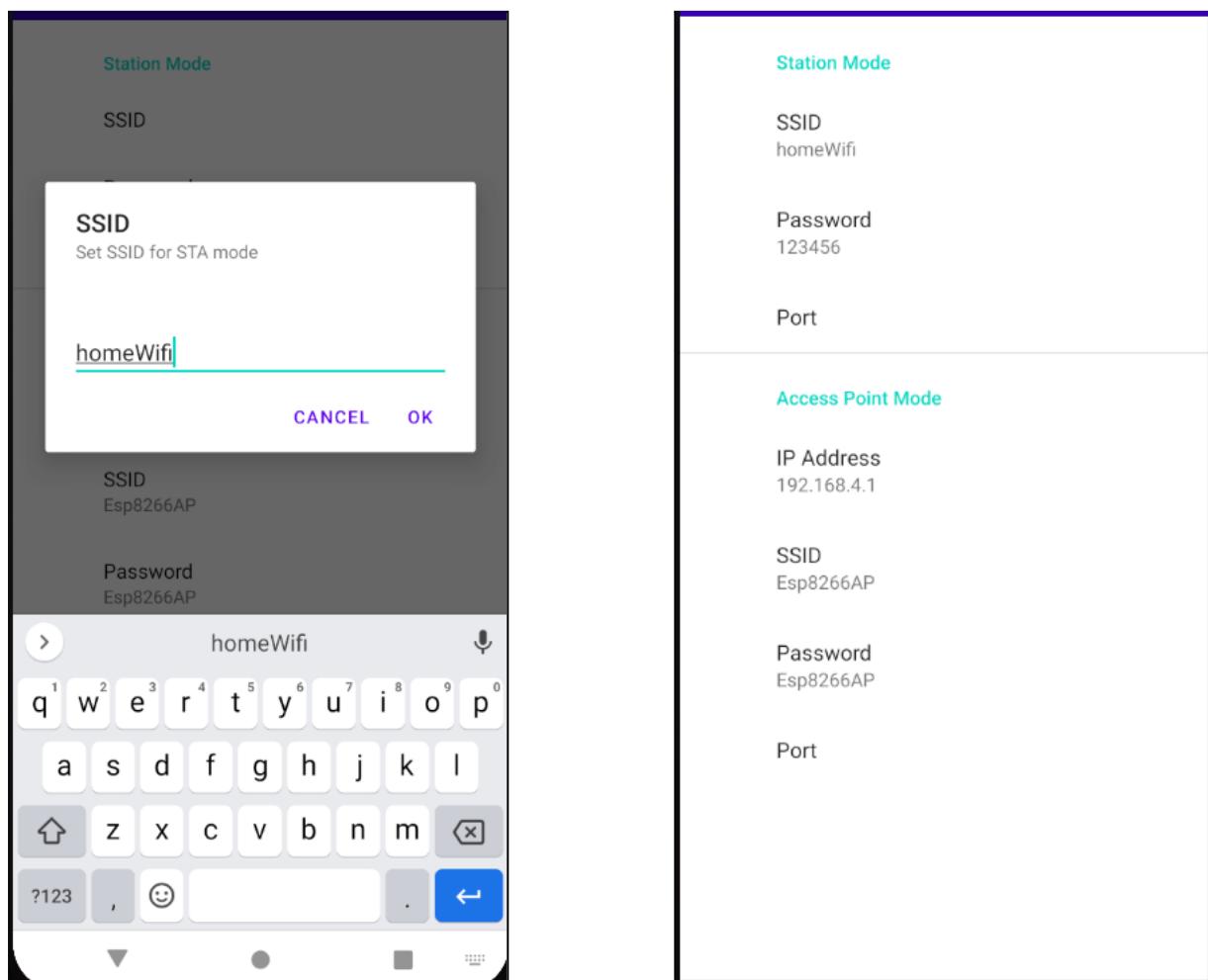
**Фиг. 7.6** - Android WiFi menu.

## 7.4 Промяна на настройките на устройство

За да променят настройките на устройството, се влиза неговата страница и се натиска бутона за настройки (в лилаво на фиг. 7.5).

Това отваря следната страница за настройки. Следната демонстрация показва добавяне на настройки за работа в мрежов режим

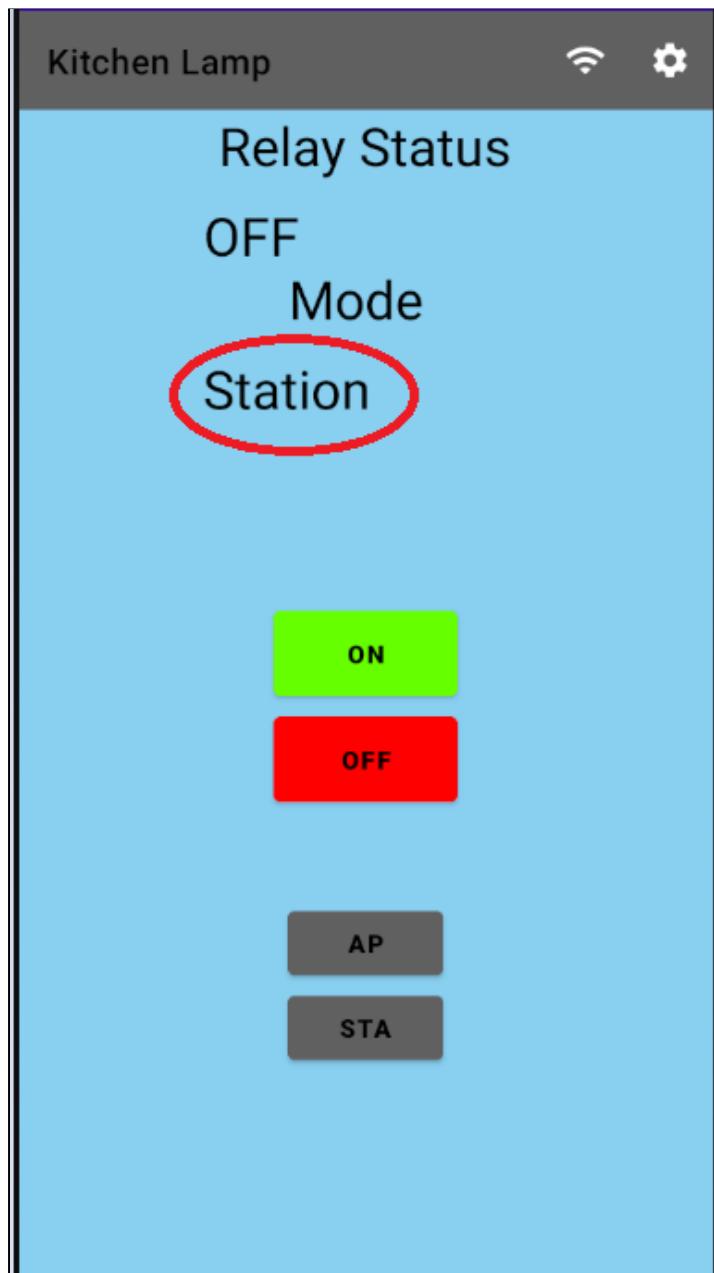
**Забележка:** Настройката за port, не се използва в сегашната версия на приложението



**Фиг. 7.7 и 7.8 - настройки на устройството**

След това е възможно променянето на режима на устройството чрез изпращане на заявка чрез бутона "STA" (в зелено на фиг 7.5)

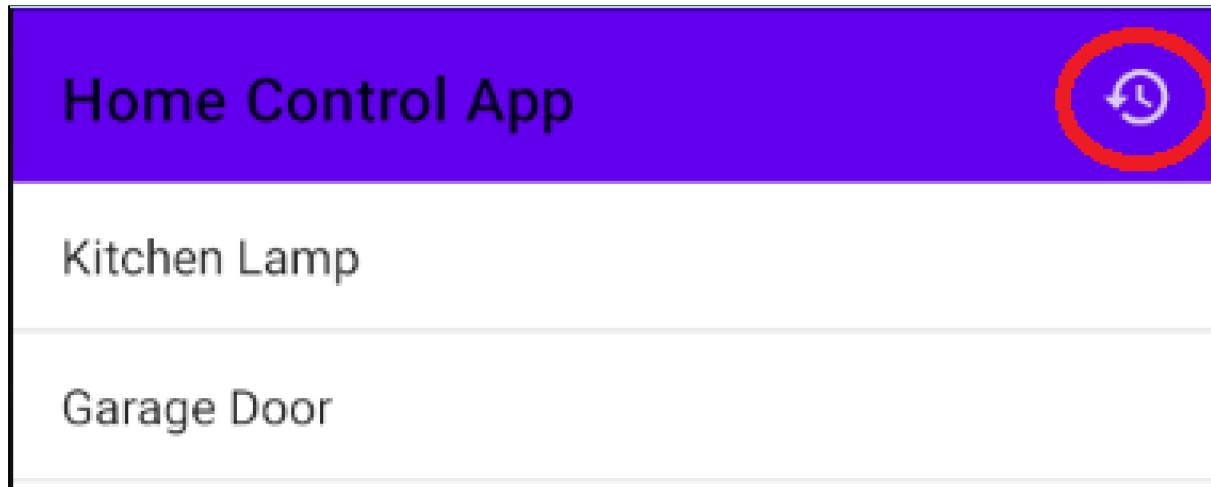
Извършването на това действие променя индикатора "mode" (в оранжево на фиг. 7.5) от "Access Point" на "Station". След сменяне на режима, мобилното устройство трябва ръчно да се премести към новоизбраната мрежи.



**Фиг. 7.9** - меню на устройство

## 7.5 Timeline на изпратените команди

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



**фиг 7.10** - главно меню

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

## **Име на устройството | Вид Команда | Дата и време**

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

**Фиг. 7.11** - timeline на командите

## **ЗАКЛЮЧЕНИЕ**

Дипломния проект постигна поставените към него изисквания. Разбира се разработката е в доста ранен етап и може да се усъвършенства. Работата по него ще продължава и в бъдеще. Някои от належащите поставени цели към него са:

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.
7. Да се направи web interface за управление като алтернатива на android приложението

## **ИЗПОЛЗВАНА ЛИТЕРАТУРА**

1. Документация на Kotlin - <https://kotlinlang.org/docs/home.html>
2. Урок за въведение в Kotlin - <https://bit.ly/3bJ0qa3>
3. Урок за създаване на страница за настройки -  
<https://bit.ly/30CgkwK>
4. Документация на Retrofit2 -  
<https://square.github.io/retrofit/2.x/retrofit/>
5. Документация на Esp8266 -  
<https://arduino-esp8266.readthedocs.io/en/latest/>
6. Сравнение между Java и Kotlin -  
<https://kotlinlang.org/docs/comparison-to-java.html>
7. Документация на Android Studio -  
<https://developer.android.com/docs>
8. Урок за инсталация на Esp8266 в Arduino Ide. -  
<https://randomnerdtutorials.com/how-to-install-esp8266-board-arduino-ide/>
9. Някои от използваните за решаване на проблеми Stack Overflow линкове - <https://pastebin.com/UzJK31Yr>

## **СЪДЪРЖАНИЕ**

Увод.....	4
Методи, средства и технологии за дистанционно контролиране на устройства от дома и разработка на android приложение.....	5
Проектиране на блоковата схема на система за дистанционно контролиране на устройства от дома.....	14
Реализация на система за дистанционно контролиране на устройства от дома.....	20
Програмна реализация на android приложение – управляващ софтуер.....	28
Проектиране на графични оригинали на печатни платки.....	42
за дистанционно контролиране на устройства от дома.....	44
Ръководство за потребителя.....	46
Заключение.....	54
Използвана литература.....	55