|  |  |  |
| --- | --- | --- |
|  | **УНИВЕРЗИТЕТ “Св. КИРИЛ И МЕТОДИЈ” - СКОПЈЕ**  **ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ** |  |

- **ДИПЛОМСКА РАБОТА** -

по предметот

**КЛАСТЕР СИСТЕМИ**

**Тема**

**БЕЗБЕДНО ДАЛЕЧИНСКО УПРАВУВАЊЕ СО УРЕДИ ВО ПАМЕТЕН ДОМ**

|  |  |
| --- | --- |
| Ментор: | Изработил: |
| Вон. проф.  Д-р Марија Календар | Борјан Калиноски, индекс бр. 237/2016  e-mail:borjankalinoski@hotmail.com |

*Скопје, декември 2020*

**Содржина**

[Апстракт 3](#_Toc222636336)

[1 Вовед 4](#_Toc222636337)

[2 Структура на дипломската 24](#_Toc222636338)

[3 Креирање на содржина 25](#_Toc222636339)

[4 Креирање на листа на слики и табели 25](#_Toc222636340)

[5 Тело на дипломската 26](#_Toc222636341)

[5.1 Уредување на дипломската 26](#_Toc222636342)

[5.2 Наслови 26](#_Toc222636343)

[5.3 Главниот текст 26](#_Toc222636344)

[5.4 Слики и табели 27](#_Toc222636345)

[5.5 Формули 29](#_Toc222636346)

[5.6 Фусноти (забелешки) 29](#_Toc222636347)

[6 Референци 29](#_Toc222636348)

[7 Проверка на граматичката исправност 30](#_Toc222636349)

[8 Предавање на дипломската работа 31](#_Toc222636350)

[9 Презентација 31](#_Toc222636351)

[10 Оценување 31](#_Toc222636352)

[11 Плагијат и како да се избегне 31](#_Toc222636353)

[12 Листа за проверка 31](#_Toc222636354)

**Листа на слики**

[Слика 1 Дијалог за креирање на листа на табели/слики 25](#_Toc222636355)

[Слика 2 Дијалог за нагодување на стилови 26](#_Toc222636356)

[Слика 3 Дијалог за вметнување на наслов на слика или табела 28](#_Toc222636357)

[Слика 4 Дијалог за нагодување на нумерација 28](#_Toc222636358)

[Слика 5 Лого на ФЕИТ 28](#_Toc222636359)

[Слика 6 Начин на вметнување на референца 30](#_Toc222636360)

**Листа на табели**

[Табела 1 Карактеристики на батерии кои може да се полнат 27](#_Toc222636361)

# Апстракт

*Во овој дипломски труд е дизајниран и имплементиран систем за безбедно далечинско набљудување и управување на електронски уреди во едно домаќниство. Како концепт за системот разгледани се електронските уреди: систем за разладување (клима) и поштенско сандаче. Системот вклучува хардверски и софтверски дел. Хардверскиот дел се состои од микроконтролери, сензори и актуатори кои се поставени во домаќинството и имаат за задача да ги направат електронските уреди „попаметни“. Софтверскиот дел претставува мобилна апликација преку која се набљудуваат и управуваат уредите во домаќинството. Мобилната апликација е изработена во технологијата React Native. За управување со клима уредот се искористени: микроконтролерот ESP8266, сензор за мерење на температура и влажност на воздух SI7021 и инфрацрвена диода. За поштенското сандаче се искористени: микроконтролерот ESP32 и два ултразвучни сензори за мерење на растојание HC-SR04. За комуникацијата помеѓу микроконтролерите и мобилната апликација, како и за автентикација на корисниците се користат неколку Cloud сервиси од Firebase. Системот ни овозможува преку една конторлна точка (мобилната апликација) да имаме целосен преглед и контрола врз нашиот дом само со пристап до интернет. Како што се развива технологијата, така концептот за паметни домови е се поактуелен бидејќи значително можат да ни го подобрат и олеснат секојдневниот живот. Затоа, системот е дизајниран на таков начин што ни дава можност во иднина да поврземе и други уреди кои би ни го направиле нашиот дом попаметен, а нашиот живот полесен.*

# Клучни зборови: Internet of Things, Smart Homes, Arduino IDE, ESP8266, ESP32, React Native, FirebaseВовед

Интернет на Нештата (англиски: Internet of Things (IoT)) се однесува на големиот број на физички уреди ширум светот кои се поврзани на интернет. Сите тие уреди собираат и разменуваат податоци, при што може да превземат соодветни акции во зависност од условите во кои се наоѓаат. Речиси секој физички објект може да се претвори во IoT уред доколку се поврзе на интернет. На пример сијалица која може да се управува од мобилна апликација се смета за IoT уред. Терминот IoT најчесто се користи за уреди кои обично не би се очекувало да бидат поврзани на интернет. Затоа, персоналниот компјутер и паметните телефони не се сметаат за IoT уреди, а илјадниците сензори и актуатори кои се наоѓаат во моторот на еден авион се сметаат за IoT уреди кои ја набљудуваат состојбата на авионот.

Првиот ваков уред е изграден од страна на студенти на универзитетот Carnegie Mellon во 1980-тите години. Студентите успеале да поврзат автомат за пијалоци на интернет, при што апаратот им кажувал дали пијалоците во него се доволно ладни. Технологијата во тоа време споро се развивала бидејќи хардверските компоненти биле скапи и обемни и немало начин за тие ефикасно да комуницраат меѓусебе.

Денеска, во 2020 година на интернет се поврзани околу 9.5 билиони IoT уреди. Претпоставките на Интернационалната Корпорација на Податоци (англиски: International Data Corporation) проценуваат дека во 2025 година на интернет ќе бидат поврзани околу 41.6 билиони IoT уреди, генерирајќи повеќе од 79.4 ZB (zettabytes, 10^21 Bytes) податоци. Развојот на 5Г технологијата и машинското учење се само дел од технологиите кои го забрзуваат трендот на IoT уредите. Ова овозможува голем број на IoT уреди да разменуваат и да собираат податоци со поголема брзина, при што пристапот до тие податоци нудат предности за кои сеуште не сме ни свесни. Големиот број на податоци веќе се користат за креирање на различни модели за предвидување. IoT уредите овозможуваат следење на состојбата на системите во реално време, што придонесува до полесно одржување и оптимизација на истите. Тие наоѓаат се поголема примена медицината, земјоделството, индустријата, а концептите како што се паметни градови и паметни домаќниства веќе стануваат реалност.

Од паметни часовници и телевизори, се до паметни клими и фрижидери, на пазарот се појавуваат се повеќе уреди кои се паметни. Бидејќи паметните уреди се релативно нова технологија, може да се примети дека постои разлика во цената на еден обичен уред и паметен уред. За едно домаќинство да се направи „паметно“, потребно е да се заменат веќе постоечките електронски уреди со нови и паметни уреди. Меѓутоа, цената да се заменуваат функционални електронски уреди со паметни уреди е многу висока. Наместо да се „фрлат“ постоечките електронски уреди, многу подобро би било доколку постои начин истите да се направат паметни, притоа цената за да се изведе тоа да биде исплатлива.

Во овој дипломски труд е разгледано како да се направат електронските уреди во едно домаќинство „паметни“ со помош на ефтини хардверски компоненти, при што е изработена мобилна апликација преку која може да се набљудуваат и управуваат сите паметни уреди во едно домаќинство. Електронските уреди кои се разгледани во дипломскиот труд се: систем за разладување (клима) и поштенско сандаче.

# Избор на технологија за мобилна апликација

## Споредба помеѓу Native и Cross-Platform развивање на мобилни апликации

//Бидејќи изборот на технологии за развивање на еден проект директно влијае врз ресурсите кои се трошат на проектот, како и на перформансите и функционалностите што ги нуди истиот, најпрво се разгледани начините на кои може да се развие мобилна апликација.

Постојат два начини за развивање на мобилни апликации и тоа: развивање на матични (англиски: Native) апликации и развивање на апликации за повеќе платформи (англиски: Cross-Platform).

Развој на Native апликации значи развивање на апликации со алатки и програмски јазици специфични за една платформа. На пример може да се развие Native апликација за Android во програмскиот јазик Java или да се развие Native апликација за iOS во програмскиот јазик Swift. При развој на Native апликации постои директната интеракција помеѓу кодот и хардверските ресурси на платформата на која се работи. Тоа овозможува високи перформанси на апликацијата, а високите перформанси резултираат во подобро корисничко искуство. Недостаток при развој на Native апликации е тоа што доколку треба да се развие апликација за платформите Android и iOS, ќе биде потребно два тима да развиваат две апликации за двете платформи.

Развој на Cross-Platform апликации значи развивање на апликации кои работат на повеќе платформи. Нивната најголема предност предност е во тоа што со пишување на еден код може да се развијат апликации повеќе платформи како што се Android и iOS. Во овој случај на апликацијата и е потребен дополнителен слој за апстракција за да пристапи до ресурсите на платформата, што ќе резултира во пониски перформанси на апликацијата. На програмерите ќе им биде потешко да пристапат до low-level функционалностите на телефонот како што се микрофон, камера, локација и друго. Работи кои треба да се разгледаат при избор на технологија за развој на мобилни апликации се:

* Комплексност на апликацијата
* Време за развој
* Цена

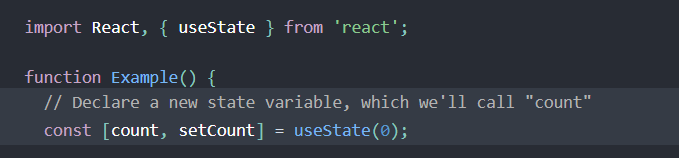
Доколку е потребно да се развие апликација во која треба да се пристапува до low-level функционалностите на телефонот и апликација во која има тешка обработка на податоци, подобар избор е да се развие Native апликација. Доколку апликацијата е поедноставна и треба само да се прикажат некои податоци кои се преземени од интернет, подобар избор е да се развие Cross-Platform апликација. Времето за развој на апликацијата и ресурсите кои треба да се одвојат при развој на Native апликации се значително поголеми и не треба да се занемарат при избор на технологија за развој на мобилни апликации.

Во овој дипломски труд е изработена мобилна апликација во која корисниците можат да ги набљудуваат и управуваат паметните уреди во нивното домаќинство, притоа корисниците можат да имаат уреди од тип на клима уред и поштенско сандаче. За овозможување на безбедносна комуникација помеѓу корисникот и паметните уреди, како и за обезбедување на тајноста и интегритетот на податоците што се разменуваат, корисникот најпрво мора да биде автентикуван пред да може да ги управува и набљудува паметните уреди во неговиот дом. Изработени се три екрани (англиски: Screens) од кои: екран за автентикација, екран за преглед на уредите и екран за управување со клима уред. Поради едноставноста на мобилната апликација и поради потребата од пократко време за развој на апликација за повеќе платформи, мобилната апликација е изработена со JavaScript рамката (англиски: Framework) React Native, која преставува Cross-Platform технологија и е базирана на библиотеката библиотеката React.

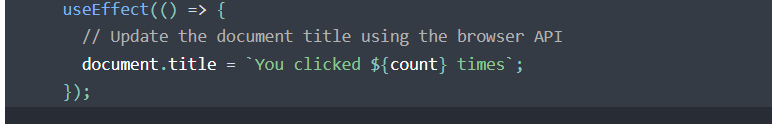
## React

React претставува JavaScript библиотека за развивање на кориснички интерфејси. Таа може да се извршува на било која платформа која има можност да извршува JavaScript код, така што со развивање на една React апликација ефективно развиваме апликација за повеќе платформи и тоа: веб, Android и iOS. Корисничкиот интерфејс на една React апликација е составен од повеќе мали и изолирани парчиња код наречени компоненти, кои се организирани во хиерархиска структура. React компонентите на влез добиваат податоци наречени сопствености (Properties или props), кои ги добиваат од родител компонентите, а на излез, преку методот render, го рендерираат (враќаат) изгледот што треба да се прикаже на екран. Покрај влезните податоци, компонентите можат да имаат внатрешна состојба (англиски: State), при што компонентата се ре-рендерира со секоја промена на состојбата. Внатрешната состојба на компонентите се состои од променливи чија структура е дефинирана од страна на програмерот.

Во верзијата 16.8 од React, воведен е нов начин за развивање на React апликации со користење на React Hooks. Тие претставуваат специјални функции кои ни овозможуваат да пристапиме до одредени одлики на React. Кодот при развивање на React апликации со користење на React Hooks е повеќе читлив и одржлив во споредба со кодот при традиционалниот начин за развивање на React апликации (со Class компоненти). Покрај тоа, подобрени се перформансите на React апликациите, подобрено е развојното искуство на програмерот (англиски: Developer Experience, DX), а со тоа е подобрено и корисничкото искуство.

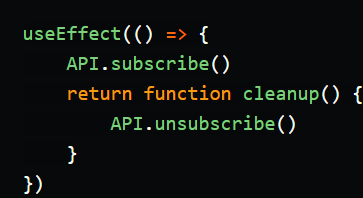
Секоја компонента има внатрешна состојба која е дефинирана од состојбени променливи. За декларирање на состојбени променливи, се користи hook-от useState. На Слика 3 е прикажан пример за декларирање на состојбена променлива со име count (бројач). Променливата се нарекува count, меѓутоа може да биде наречена било како. Вредностите на обичните променливи исчезнуваат кога компонентата ќе заврши со рендерирање, а состојбените променливи се зачувуваат од страна на React, така што тие ја задржуваат својата вредност при секое рендерирање на компонентата. Единствениот аргумент кој се доделува на useState hook-от е почетната вредност на состојбената променлива. Во примерот, како почетна вредност на бројачот е зададена вредноста 0. useState hook-от враќа пар вредности и тоа: моменталната вредност на променливата и функција за ажурирање на вредноста на променливата. Во случај кога во компонентата има комплексна состојбена логика, како алтернатива на useState се користи useReducer hook-от, кој може да ракува со повеќе состојбени променливи одеднаш.

Слика 1. Пример за користење на useState hook-от

useEffect hook-от ни овозможува да извршуваме несакани ефекти (англиски: Side Effects) во компонентите. Преземање на податоци од интернет, претплаќање на надворешен извор на податоци и рачно менување на променливите во React компонентите се само неколку примери за несакани ефекти. useEffect hook-от се извршува при секое рендерирање на компонентата. На Слика 4 е прикажан пример за користење на useEffect hook-от, при што е надополнет претходниот пример со бројачот. Бидејќи секоја промена на некоја состојбена променлива предизвикува ре-рендерирање на React компонентата, со промена на вредноста на променливата count, ќе се ре-рендерира самата компонента, така што ќе се повика ефектот кој ќе го менува насловот на веб прелистувачот соодветно.

Слика 2. Пример за користење на useEffect hook-ot

Но што ако во компонентата има повеќе состојбени променливи? Тоа значи дека ефектот ќе се извршува при промена на било која од вредностите на состојбените променливи. За да се избегнат непотребните извршувања на ефектот, односно ефектот да се извршува само кога ќе се промени вредноста на променливата count, како втор аргумент на useEffect hook-от може да се додели низа од променливи од кои зависи ефектот, што овој случај би било низа со само елементот count. При секое рендерирање, React ги споредува вредностите на променливите од низата со вредностите што тие ги имале при претходното рендерирање, така што ефектот ќе се изврши само доколку постои разлика во истите.

Кога се користат ефекти во кои има претплаќање на некој надворешен извор на податоци, потребно е ефектот да се исчисти (англиски: Cleanup), односно да се отпишеме од претплатата со цел да се избегне претекување на меморија и паѓање на програмата. За тоа, во ефектот се пишува cleanup функција, која се извршува секогаш кога React компонентата ќе се одмонтира (англиски: unmounts) од изгледот на екранот.

Слика 3. Пример за користење на useEffect hook-от со cleanup функција

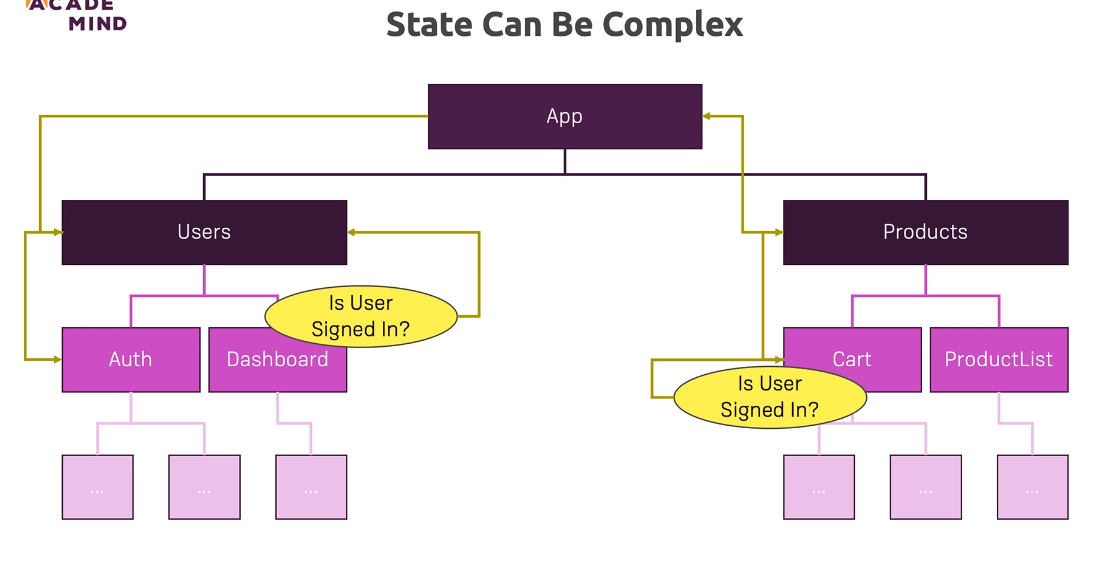
useCallback хоок-от се користи за оптимизација на функциите во React компонентите. Секоја функција е „обвиткана“ од овој hook, со цел таа да не се ре-рендерира при промена на состојбата во компонентата. Првиот аргумент на овој hook е функцијата која треба да се изврши, а вториот аргумент е низа од променливи од кои зависи извршувањето на функцијата. Во примерот прикажан на Слика 6, функцијата зависи од променливите a и b, така што функцијата ќе остане меморирана од страна на React се додека променливите a и b останат непроменети. При секое рендерирање на компонентата, функцијата нема одново да се креира, туку таа ќе ја задржи истата вредност сè додека некоја од променливите a или b не добие нова вредност.

Слика 4. Пример за користење на useCallback хоок-ot

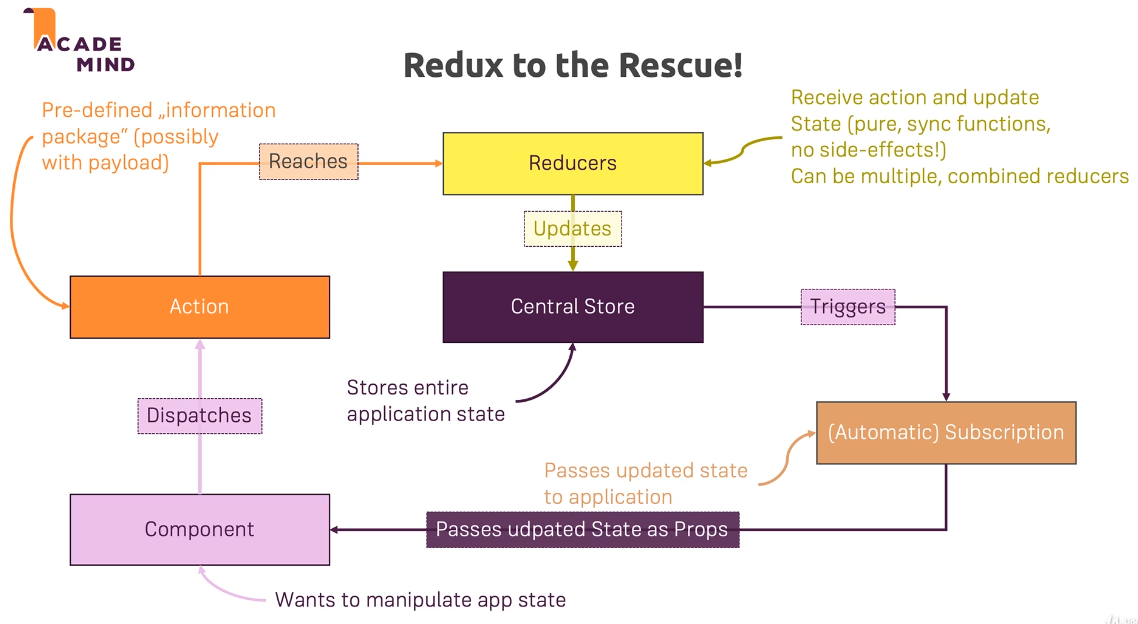
* useReducer?
* Povekje za redux?

## Redux

За еден податок да пристигне од една компонента до друга, потребно е тој да се искачи по хиерархиското дрво до првиот заеднички родител на компонентите, а потоа да се спушти низ хиерархијата до компонентата до која што треба да пристигне. Со зголемување на бројот на компонентите во една React апликација, така се зголемува комплексноста на разменувањето и споделувањето на податоци помеѓу компонентите (Слика 10). Ова претставува проблем кај сите React апликации, а се разрешува со користење на надворешни библиотеки кои се задолжени за менаџирање на глобална состојба на апликацијата.



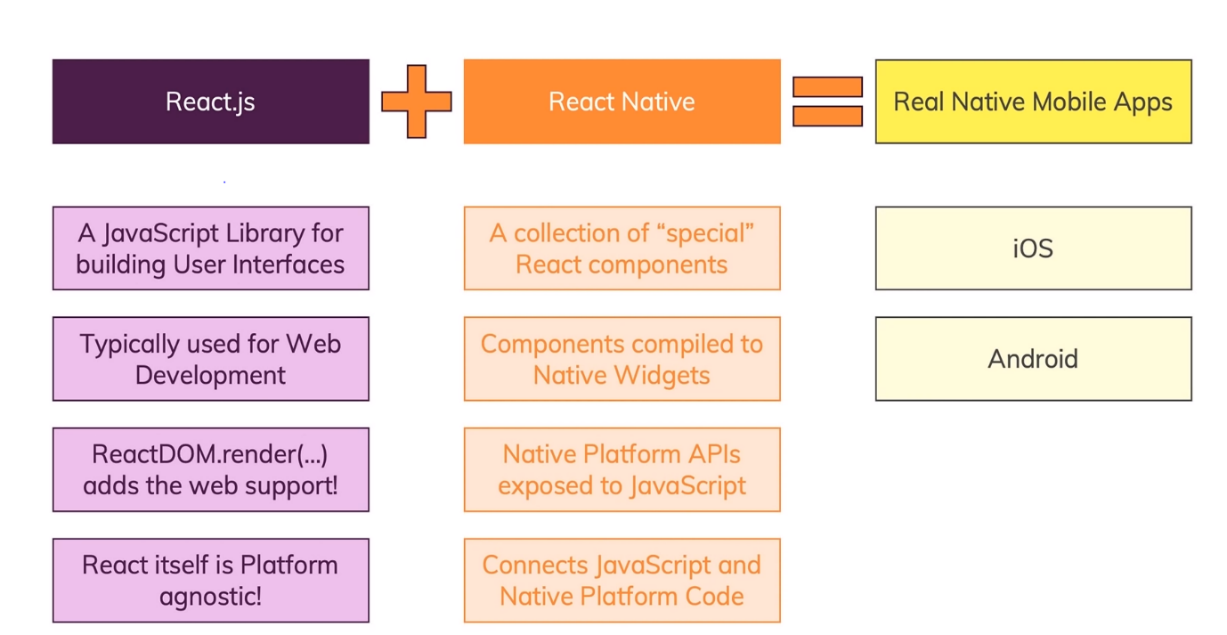
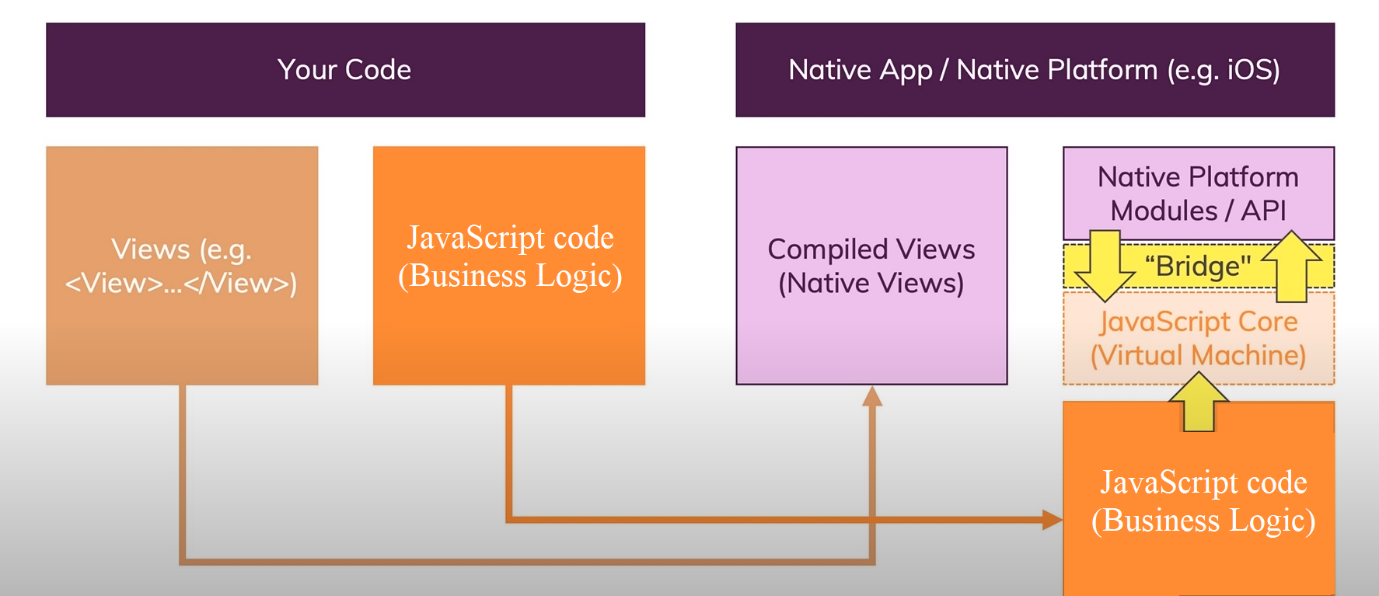
Слика 5.

За разрешување на овој проблем, искористена е JavaScript библиотеката Redux, која е најпозната и најкористена во React апликациите. За разлика од локалната состојбата на компонентите, со Redux се дефинира глобална состојба на целата апликација која се чува во централно складиште (англиски: Store). Централното складиште е споделено помеѓу сите компоненти, при што компонентите можат да ги читаат и да ги менуваат податоците во него. За менување на состојбата во складиштето, компонентата треба да испрати (англиски: dispatch) акција, која се проследува до редуктор (англиски: reducer) кој е задолжен за ажурирање на состојбата во централното складиште. Другите компоненти може да ги слушаат ажурирањата на состојбата во централното складиште. На овој начин е овозможено споделување и разменување на податоци помеѓу компонентите во една React апликација. Структурата на состојбата во складиштето, акциите и редукторите се дефинираат од страна на програмерот, додека пак Redux е задолжен за споделувањето на состојбата низ целата апликација.

Слика 6.

## React Native

React Native претставува JavaScript рамка за развивање на мобилни апликации и е целосно базиран на библиотеката React. Тој нуди колекција од React компоненти кои потоа се компајлираат во Native код специфични за платформите Android и iOS соодветно. Low-level функционалностите на телефонот како што се пристап до микрофон, камера и друго се изложени преку API од Native платформата, кое може да се пристапи со JavaScript код.

На Слика 8 е прикажан детален начин за тоа како се компајлира кодот на една React Native апликација. По компајлирање, кодот напишан во React Native се дели на два дела и тоа: изгледот на апликацијата (Views) и бизнис логиката на апликацијата. Изгледот на апликацијата се всушност компонентите што ги нуди React Native и тие целосно се компајлираат во Native код специфични за самата платформа. Од кориснички аспект тука спаѓаат сите работи што корисниците ги гледаат на екранот. Кодот задолжен за бизнис логиката не се компајлира во Native код, тој се извршува како JavaScript код на посебна виртуелна машина која се наоѓа во самата апликација. JavaScript кодот кој се наоѓа на виртуелната машина комуницира со API-то на Native платформата преку мост. Програмерот не треба да се грижи за тоа на кој начин се одвива комуникацијата помеѓу JavaScript кодот и API-то на Native платформата, меѓутоа, тој треба да биде свесен дека постои дополнително ниво на апстракција при комуникацијата меѓу кодот и API-то на Native платформата. Размена на податоци помеѓу апликацијата и сервер, менаџирање на локална состојба на React компонентите и менаџирање на глобална состојба на апликацијата со Redux се само дел од работите кои спаѓаат во бизнис логиката на една React апликација.

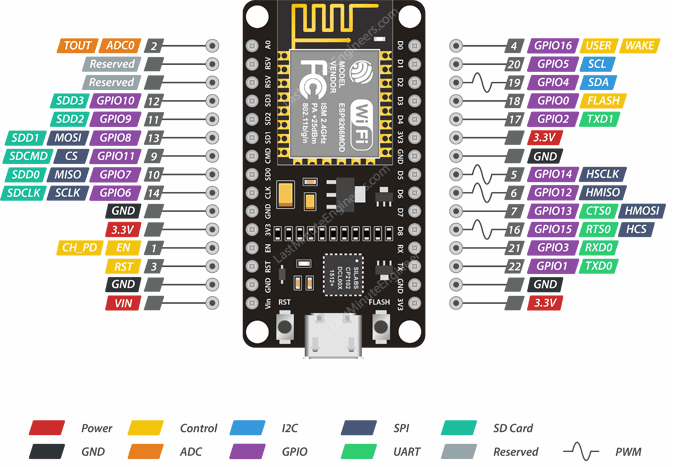
Слика 7. Компајлирање на кодот на една React Native апликација

Слика 8.

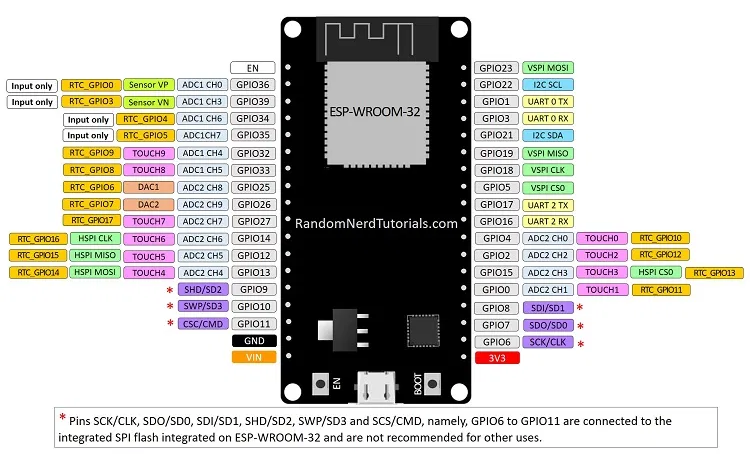
# Избор на микроконтролери – ESP8266 и ESP32

За електронските уреди во едно домаќинство да се претворат во IoT уреди, потребен е исплатлив и едноставен начин за тие да можат да се контролираат и набљудуваат преку интернет. За таа цел се искористени нискобуџетните микроконтролери ESP8266 и ESP32, кои претставуваат системи на чип (англиски: System on a Chip, SoC) произведени од кинеската компанија Espressif Systems. Поради вградениот WiFi примопредавател тие највеќе се користат за IoT базирани апликации. Во споредба со други микроконтролери со вграден WiFi примопредаватаел, ESP микроконтролерите се најевтини на пазарот, а имаат доволно добри перформанси за да ги задоволат поголем дел од IoT апликациите. Тие можат да се поврзат на WiFi мрежа, да хостираат WiFi мрежа, да се користат како веб сервери и друго. Опсегот на работната температура им се движи од -40°C – 125°C. Различни компании ги користат ESP микроконтролерите со цел да изградат различни модули и развојни плочи кои се разликуваат по бројот на пинови, WiFi антената, флеш меморијата и друго. За клима уредот е искористена развојната плоча NodeMCU ЕSP8266, а за поштенското сандаче е искористена развојната плоча NodeMCU ESP32. Во случајот со климата, микроконтролерот користи акутатор кој може да ја контролира климата преку мобилната апликација, а во случајот со поштенското сандаче поставен е сензор кој детектира дали има пошта во сандачето, а потоа микроконтролерот ја испраќа информацијата до мобилната апликација. Развојните плочи изградени од NodeMCU имаат вграден регулатор на напон и micro USB порта, што ги прави лесни за програмирање и се идеални за почетници. ESP32 е понов и помоќен од ESP8266 со тоа што има вграден Bluetooth модул, има поголема процесирачка моќ и има повеќе вградени пинови. Бидејќи хардверскиот дел во овој труд е прилично едноставен и се искористени се само неколку пинови и WiFi примопредавателот, може да се искористат само два ESP8266 микроконтролери, меѓутоа бидејќи на располагање имав само еден ESP8266 микроконтролер, искористен е и микроконтролерот ESP32.

Развојната плоча NodeMCU ESP8266 се состои од 32-битен микропроцесор Tensilica Xtensa LX106 RISC кој работи со брзина од 80 MHz. На располагање има 64KB RAM и 4MB флеш меморија, а големината на меморијата му е доволна за да се справи со разменување на HTML, JSON, XML и други типови на податоци што се најчесто потребни во IoT апликации. На Слика 3 е прикажан NodeMCU ESP8266 и неговите 30 пинови групирани по функционалност. Пиновите за напојување служат за да се напојува микроконтролерот и за да се напојуваат други надворешни компоненти. Контролните пинови служат за да го ресетираат и да го будат микроконтролерот. Има 1 аналоген пин и 16 дигитални пинови на кои може да се поврзат сензори и акутатори за да комуницираат со околината во која се наоѓа. Поддржува UART, I2S, SPI и I2C типови на комуникација со надврорешни периферии.

 Развојната плоча NodeMCU ESP32 има вграден WiFi примопредавател и Bluetooth 4.2 модул. Тој се користи во IoT базирани апликации и во проекти кои имаат потреба од Bluetooth функционалност. Се состои од микропроцесорот Tensilica Xtensa 32-bit LX6 кој има две јадра, чија брзина е конфигурабилна и се движи од 80 - 240 MHz. Има 512KB RAM и 4MB флеш меморија, која може да биде дополнително надградена. Тоа го прави доста помоќен од неговиот претходник ESP8266, а може да се најде по цена од 3 $. Опсегот на работниот напон му се движи од 2.3 – 3.6 V. Развојната плоча NodeMCU ESP8266 може да биде напојувана преку micro USB приклучок, преку VIN пинот кој очекува напон од 7-12 V или преку пинот за 3.3 V. На располагање има 36 пинови за генерална намена од кои 6 се резервирани и не се препорачува да се користат. Има 16 аналогни пинови и сите GPIO пинови освен пиновите од GPIO 6 до GPIO 11 можат да се искористат како дигитални пинови. Има 10 капацитивни сензори кои можат да чувствуваат промена во се што држи електричен полнеж, како на пример човечката кожа. Тие можат да се искористат за различни апликации базирани на допир. Поддржува SPI, I2C, I2S и UART типови на комуникација со надворешни периферии.

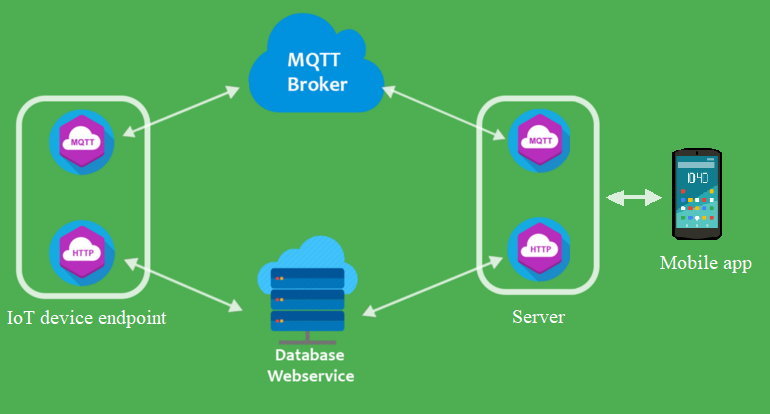
Слика 9. Пинови на NodeMCU ESP8266



Слика 10 Пинови на NodeMCU ESP32

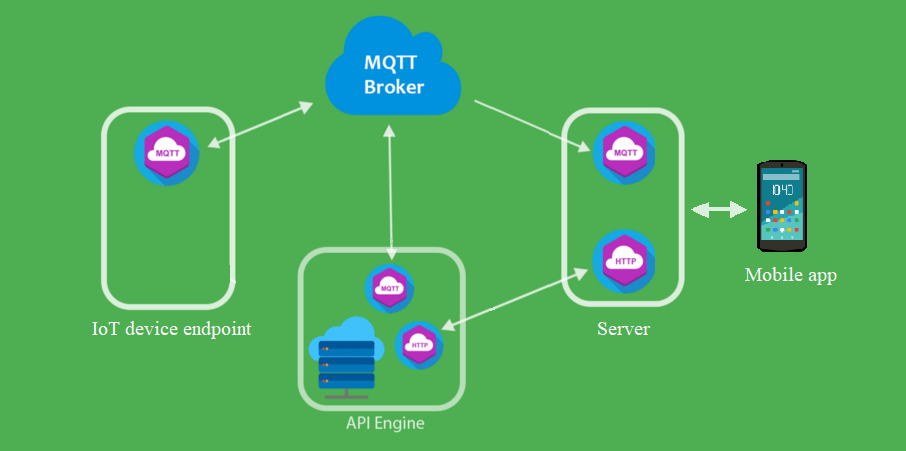
# Дизајн на системот - Firebase

Генерално сите IoT проекти имаат потреба од начин на комуникација помеѓу различни крајни точки (англиски: endpoints) кои можат да бидат IoT уреди, сервиси или апликации при што податоците треба да бидат зачувани во база на податоци. Еден од начините да се направи тоа е со помош на MQTT посредник, кој ќе ја овозможи комуникацијата помеѓу IoT уредите и другите уреди со тоа што ќе ги пренасочува дојдовните пораки од IoT уредите кон уредите кои имаат потреба од тие податоци. На Слика 5 е прикажана архитектурата на системот доколку се користи MQTT посредник. IoT уредот прво го испраќа податокот до MQTT посредникот, кој потоа преку серверот се препраќа до мобилната апликацијата каде се прикажува во реално време, меѓутоа за податоците да се складираат во база на податоци, потребно е IoT уредот истиот податок да го испрати и до базата на податоци, а потоа мобилната апликација од таму може да ги превзема податоците, повторно преку серверот. Користејќи ја оваа архитектура, IoT уредот прво го испраќа податокот до MQTT посредникот, а потоа го испраќа истиот податок до базата на податоци во вид на HTTP порака.



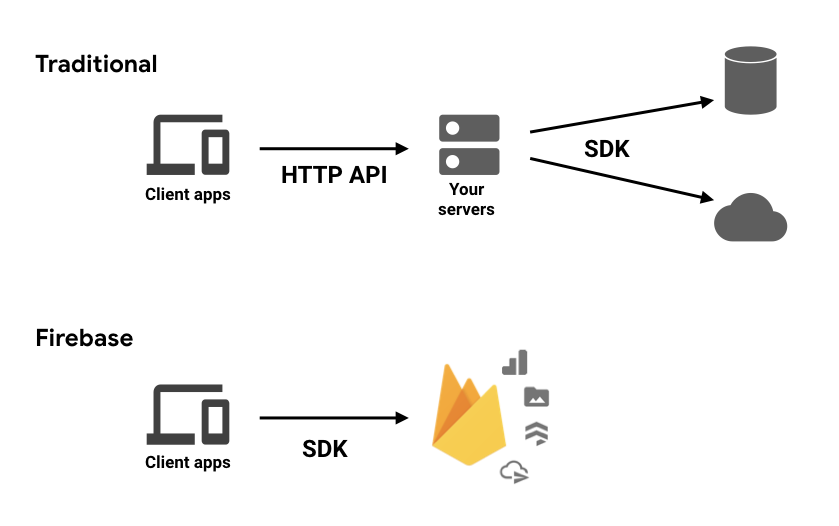
Слика 11. Архитектура на системот користејќи MQTT посредник

На Слика 6 е прикажан друг начин како да се постигне истото, со тоа што IoT уредот не го испраќа истиот податок два пати, туку го испраќа само еднаш до MQTT посредникот кој потоа преку серверот се препраќа до мобилната апликација и до API Engine кој го запишува податокот во базата на податоци. Овој начин е подобар со тоа што програмата на IoT уредот е многу помала, што е многу битно при развивање на IoT уреди каде ресуристе како меморија и процесирачка моќ се ограничени. Овие начини на реализација имаат дополнителна работа бидејќи имаат потреба од развивање на сервер кој се грижи за безбедноста на системот и автентикација на податоците.

За да се избегне развивањето на сервер ис4користена е платформата Firebase. Firebase претставува платформа за развој на апликации изградена од Google и нуди група на алатки кои можеме да ги искористиме за да ја градиме, подобруваме и да ја развиваме нашата апликација. Алатките опфаќаат голем дел од сервисите кои програмерите вообичаено би требало сами да ги градат. Овие сервиси се основа на секоја модерна апликација и вклучуваат: анализа на податоци, автентикација на корисници, бази на податоци и други. Дел од овие сервиси се всушност продукти на Google Cloud кои се интегрирани во Firebase. Сервисите се хостирани на Cloud и имаат можност за скалирање со никаков или минимален напор на програмерот.

Слика 12. Архитектура на системот користејќи MQTT посредник и API Engine

Комплетите за развој на софтвер (англиски: Software Development Kits, SDKs) кој ги нуди Firebase комуницираат директно со серверите на Google. Поради тоа тие можат директно да се користат во клиентската апликација. На Слика 7 е прикажана споредба помеѓу традиционалниот начин на развивање на апликации и развивање на апликации со помош на Firebase. Традиционалнот начин на развивање на апликации вклучува развивање на frontend и backend софтвер, каде frontend-от испраќа барања до backend-от, а потоа backend-от ја извршува целата работа. Користејќи ги Firebase сервисите, backend делот целосно се заобиколува и ни овозможува целосно да се фокусираме на развивање апликацијата која ја користат корисниците. Сервисите на Firebase нудат бесплатен план со одредени ограничувања, кои поради нивната едноставност и добра документација може лесно да се интегрираат во било која апликација.



Слика 13. Споредба помеѓу традиционален начин на развивање на апликации и развивање на апликации со Firebase

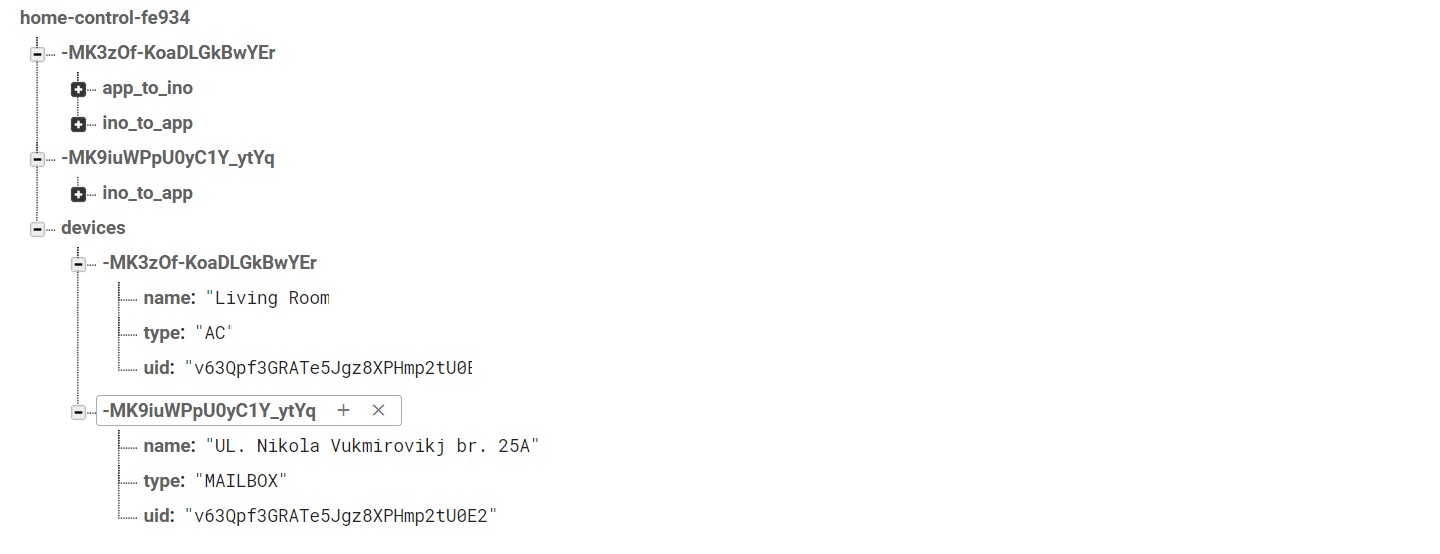
Дел од сервисите на Firebase се:

* **Firebase Authentication –** идентификација и најава на корисници
* **Realtime Database –** нерелациона база на податоци во реално време
* **Cloud Firestore –** нерелациона база на податоци во реално време
* **Cloud Functions –** „serverless“ backend функции
* **Firebase Hosting –** хостирање на апликации
* **ML Kit –** извршување на чести Machine Learning задачи
* **Cloud Messaging –** сервис за испраќање на пораки до корисници

Мобилната апликација има потреба да го знае идентитетот на корисникот, со цел тој да има пристап до уредите поврзани во неговиот дом и безбедно да комуницира со нив. За таа цел е искористен сервисот **Firebase Authentication** кој се грижи за најава и идентификација на корисниците. Најпрво се превземаат ингеренциите за автентикација на корисникот, како што се email и лозинка, и со неколку линии код се предаваат до Firebase SDK. Потоа, серверите на Firebase ги верификуваат ингеренциите на корисникот и враќаат соодветен одговор. По успешното најавување, корисникот има пристап само до уредите поврзани во неговиот дом, така што тој може да ги набљудува и управува само нив.

За да се овозможи комуникацијата помеѓу мобилната апликација и IoT уредите може да се искористи еден од сервисите за бази на податоци: **Cloud Firestore** или **Realtime Database**. Тие преставуваат Cloud базирани, нерелациони бази на податоци во реално време. Во споредба со Realtime Database, Cloud Firestore претставува поново и пософистицирано решение кое овозможува извршување на побрзи и покомплексни пребарувања (англиски: queries), меѓутоа бидејќи ESP микроконтролерите немаат поддршка за Cloud Firestore избран е сервисот Realtime Database. Податоците во Realtime Database се зачувани во вид на JSON дрво и се синхронизираат во реално време со сите поврзани клиенти. Клиентите можат да се претплатат на одредени гранки во JSON дрвото и можат да ги ажурираат гранките, при што секое ажурирање на одредена гранка е проследено до сите клиенти кои се претплатени на таа гранка. Помеѓу базата на податоци и клиентите е воспоставена двонасочна врска преку веб сокети (англиски: web socket) и на тој начин е овозможена синхронизација на податоците во реално време. Лесно се интегрира со Firebase Authentication, каде во овој случај се дефинирани правила кои кажуваат дека само автентикуваните корисници имаат пристап до базата на податоци.

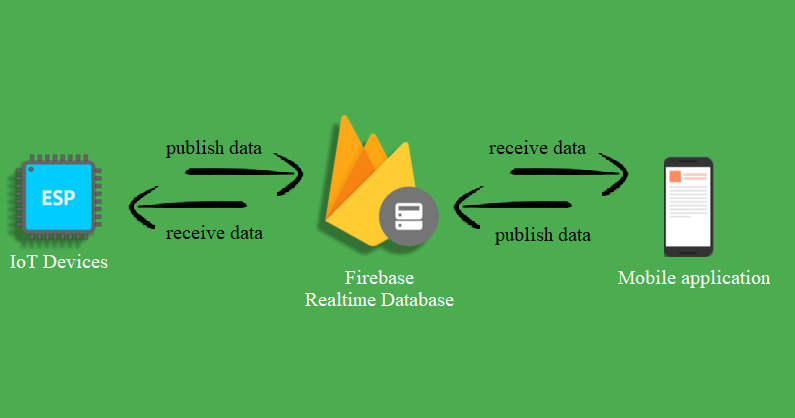
На Слика 8 е прикажанa структурата на JSON дрвото во Realtime Database за нашиот систем. Сите уреди се зачувани во патеката /devices/ како клуч-вредност парови, каде клучот е уникатен идентификатор за секој уред генериран од Firebase, а како вредност се чуваат името на уредот, типот на уредот и уникатен идентификатор на корисникот, кој е добиен од сервисот Firebase Authentication.



Слика 14. Структура на JSON дрвото во Firebase Realtime Database

На Слика 9 е прикажан начинот на комуникација помеѓу мобилната апликација и IoT уредите. Микроконтролерите запишуваат податоци на одредена гранка или патека во Firebase Realtime Database, а од другата страна мобилната апликација ги слуша сите промени што се случуваат на истата патека во базата на податоци и обратно. Испраќањето на податоци од страна на микроконтролерите до мобилната апликација е овозможен на тој начин што уредите запишуваат податоци во Realtime Database на патеките /{idNaUred}/ino\_to\_app, а од другата страна мобилната апликација се претплаќа на истите патеки, при што ги слуша сите ажурирања на тие патеки. Корисникот испраќа податоци од мобилната апликација до уредите со тоа што испраќа податоци на патеките /{idNaUred}/app\_to\_ino, а уредите слушаат на ажурирањата на податоците на истата патека и превземаат соодветна акција. Комуникацијата помеѓу микроконтролерот во поштенското сандаче и мобилната апликација е еднонасочна со тоа што микроконтролерот испраќа податок до мобилната апликација за тоа дали има пошта во сандачето. Во случајот со климата, комуникацијата е двонасочна бидејќи се испраќа податок од мобилната апликација до микроконтролерот задолжен за управување со климата, а назад се добиваат податоци од сензор за температура и влажност на воздух во собата, како и потврдна информација за тоа дали успешно сме управувале со климата. Со користење на Firebase целосно можеме да се фокусираме во развивање на IoT уредите и софтверот за мобилната апликација, додека Firebase сервисите се грижат за безбедната комуникација помеѓу различните endpoints во системот, како и за автентикација на корисниците, при што постои можност за понатамошно развивање на системот со интегрирање на други сервиси од Firebase.

Слика 15. Начин на комуникација помеѓу IoT уредите и мобилната апликација



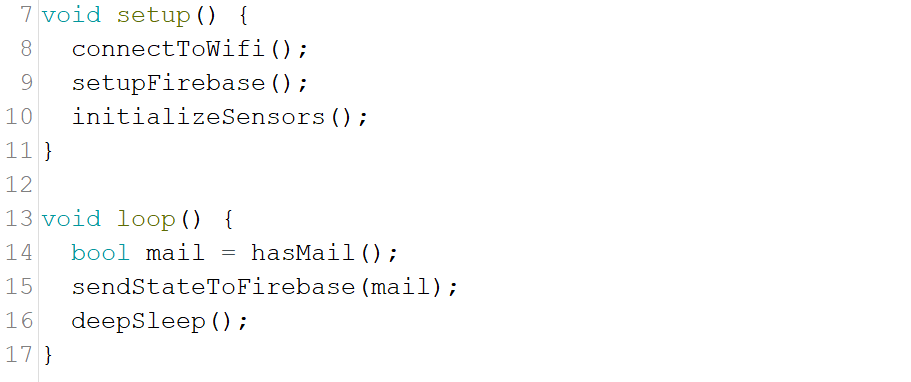
# Паметно поштенско сандаче

Поштенско сандаче претставува обична кутија во која се доставува пошта. Многу често се случува да го проверуваме поштенското сандаче за да видиме дали во него има пошта или не. Знаејќи кога точно имаме пошта во сандачето, ќе ги избегнеме сите безполезни проверувања. Наместо на секое доаѓање или излегување од дома да го проверуваме нашето сандаче, со проверување на мобилната апликација, паметното поштенско сандаче ќе не информира дали има пошта во него. За реализација се искористени електронски компоненти кои ќе детектираат дали има пристигнато пошта во поштенското сандаче и ќе ја испратат информацијата до Firebase Realtime Database, односно до мобилната апликација. Во поштенското сандаче се поставени два ултразвучни сензори HC-SR04 кои го мерат растојанието од почетокот до крајот на сандачето, а според измереното растојание, микроконтролерот ESP32 проверува дали има пошта во сандачето и ја испраќа информацијата до мобилната апликација.

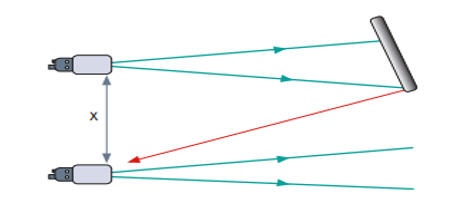
Ултразвучниот сензор HC-SR04, кој е прикажан на Слика 10, претставува сензор за мерење на растојание, и притоа може да мери растојанија од 2cm – 400cm со точност до 3mm. Има VCC пин, Trigger пин (Output), Echo пин (Input), и GND пин. Работи на следниов начин: по испраќање на високо ниво на Trigger пинот во времетраење од 10us, сензорот испраќа осум 40KHz звучни бранови од предавателот, а приемникот чека истите да се вратат. Echo пинот преминува во високо ниво се додека испратените сигнали не се рефлектираат од некој објект. Времето за кое Echo пинот е на високо ниво, односно времето што е потребно за звучните бранови да бидат испратени од предавателот и да бидат регистрирани од приемникот се користи за да се измери растојанието помеѓу сензорот и објектот пред него. Искористени се два ултразвучни сензори со цел да се опфати целото поштенско сандаче. Вредностите кои се добиени од сензорите можат да варираат подари разлика во температурата или притисокот на воздухот, затоа, најпрво се направени неколку мерења за растојанието од почетокот до крајот на сандачето, кои се искористени како готови во кодот. Сензорите потоа периодично го мерат растојанието од почеток до крајот на сандачето, при што доколку измереното растојание е надвор од првичниот измерен опсег на растојанија, тоа значи дека има пошта во сандачето, а микроконтролерот ја испраќа информацијата до Firebase Realtime Database, односно до мобилната апликација.

Слика 16. Ултразвучен сензор за мерење на растојанија HC-SR04

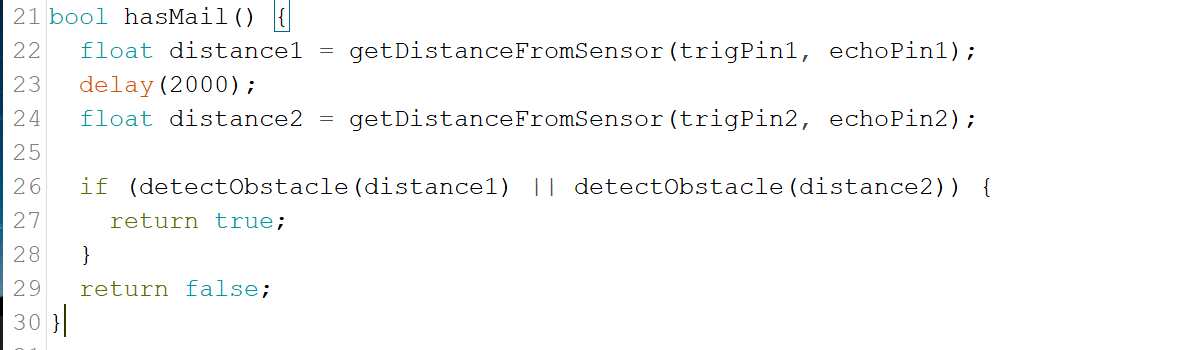
Главната програма која ја извршува микроконтролерот е прикажана на Слика 11. Функцијата setup() се извршува само еднаш и тоа при подигање на микроконтролерот. Микроконтролерот најпрво се поврзува на WiFi мрежа користејќи ја функцијата connectToWifi(). Потоа се поврзува на Firebase Realtime Database со повикување на функцијата setupFirebase(). Функцијата initializeSensors() се користи за да се иницијализираат Trigger и Echo пиновите од сензорите како Input и Output пинови соодветно. За микроконтролерот да може да се поврзе и да ажурира податоци во Firebase Realtime Database, искористена е Arduino библиотеката FirebaseESP32. Откатко ќе заврши со извршување фунцкијата setup(), почнува со извршување фунцкијата loop() која се извршува постојано.

Во loop() функцијата најпрво се повикува функцијата hasMail() која е прикажана на Слика 13, и проверува дали има пошта во сандачето. Во неа, се повикува функцијата getDistanceFromSensor() за двата сензори соодветно. При тестирање, забележав дека едниот сензор постојано дава грешни мерења. Всушност доколку пробаме во скоро исто време да мериме растојанија со двата сензора, вториот сензор ги регистрира одбиените сигнали од првиот сензор и дава грешна информација за растојанието. За да се избегне интерференција на сигналите помеѓу сензорите, меѓу двете мерења е вклучено одложување (англиски: delay) од 2000ms.

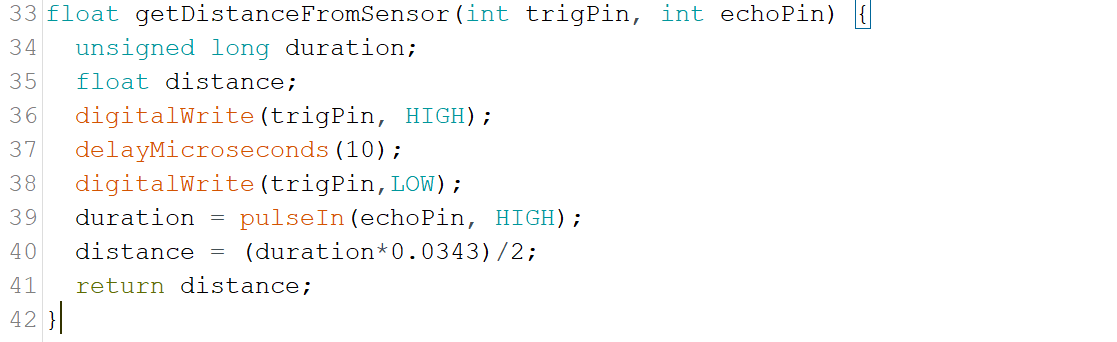
Слика 17. Главна програма која ја извршува микроконтролерот во поштенското сандаче



Слика 18. Интерференција помеѓу два ултразвучни сензори

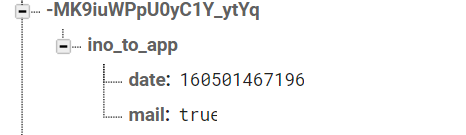
Функцијата getDistanceFromSensor() е прикажана на Слика 14. Таа како аргументи ги добива Trigger и Echo пиновите на сензорот кој треба да го мери растојанието. Според начинот на работа на сензорите, најпрво се испраќа високо ниво на Trigger пинот во времетраење од 10us, а Echo пинот преминува во високо ниво се додека приемникот не ги детектира сигналите испратени од предавателот. Времетраењето за кое Echo пинот се наоѓа во високо ниво, односно времетраењето на звучните бранови да бидат испратени од предавателот и регистрирани од приемникот се добива со функцијата pulseIn. Растојанието се добива со множење на времетраењето на патувањето на звучните бранови (добиено со функцијата pulseIn) и брзината на движење на звучните бранови, која изнесува 343 m/s, или 0.0343 cm/us и притоа резултатот се дели со два. Делењето со два е поради тоа што звучните бранови патуваат од сензорот до објектот и назад до сензорот. Добиените растојанија од двата сензори се предаваат на функцијата detectObstacle(), која проверува дали некој од сензорите измерил растојание надвор од првичниот измерен опсег на растојанија за должината на сандачето (7.5cm – 8.5cm), што би значело дека има пошта во сандачето.

Слика 19. Функцијата hasMail која проверува дали има пошта во сандачето

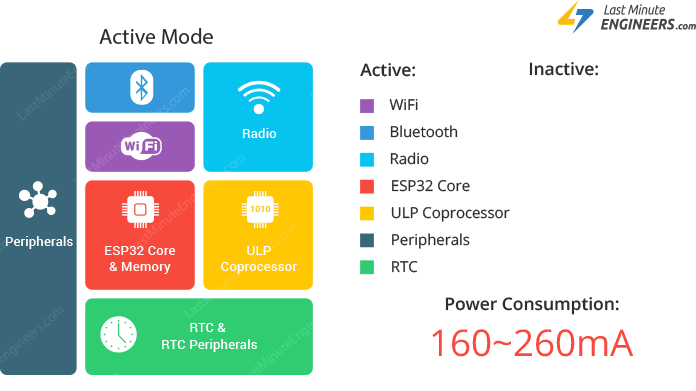


Слика 20. Функција за мерење на растојание со HC-SR04 сензор

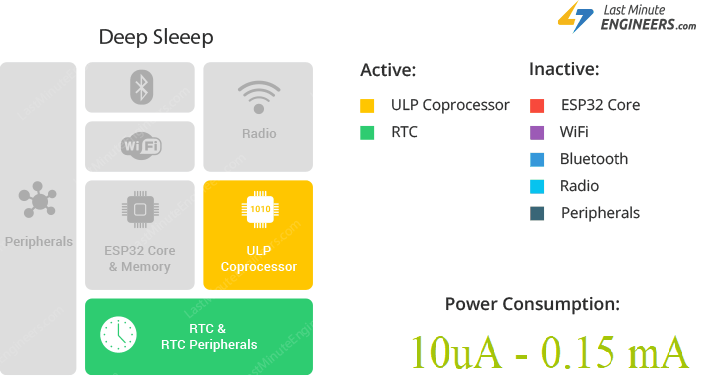
Откатко сензорите ќе проверат дали има пошта во сандачето, микроконтролерот ја запишува информацијата во Firebase Realtime Database на патеката /{idNaUred}/ino\_to\_app, каде се запишува информацијата за тоа дали има пошта во сандачето и времето во UNIX формат за тоа кога е добиена таа информација. Резултатот се зачувува во JSON формат и е прикажан на Слика 15.



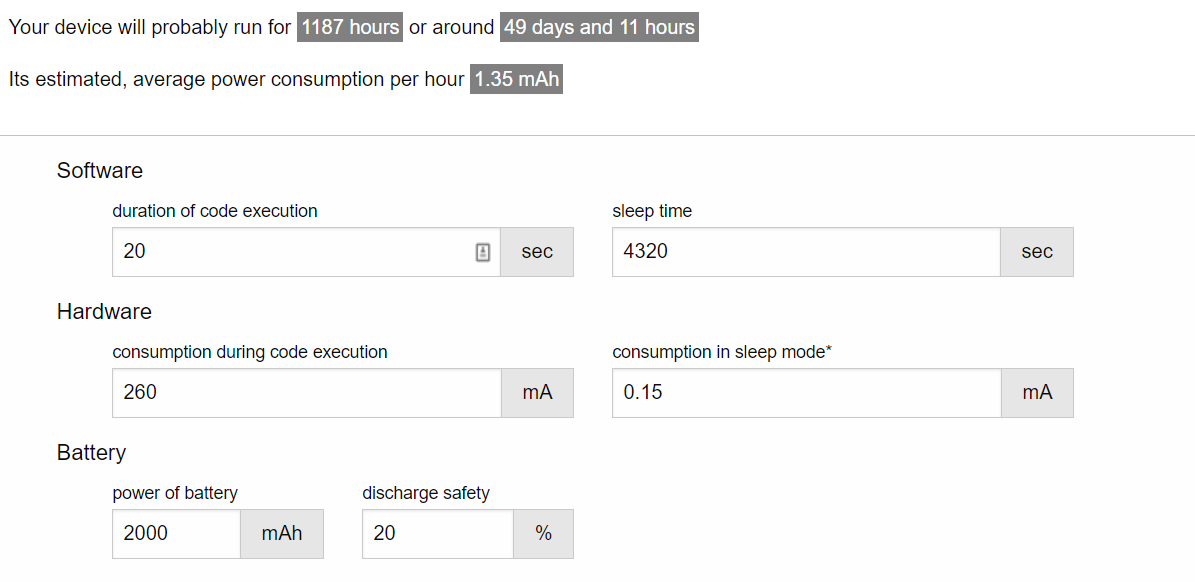
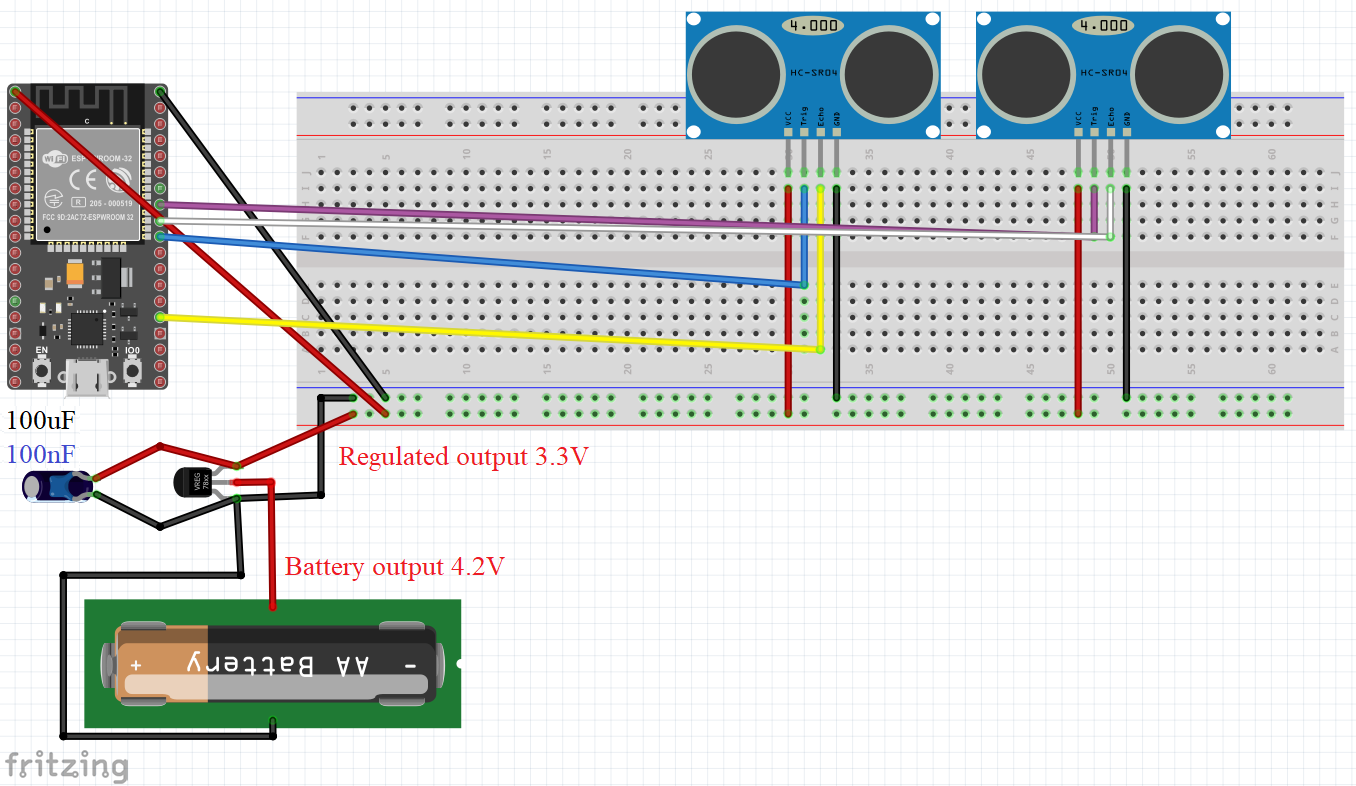
Слика 21. Информација запишана во Firebase Realtime Database

Бидејќи поштенското сандаче би било поставено на место каде што нема директен пристап на електрична енергија, потребно е хардверските компоненти да бидат напојувани преку батерија. При нормален или активен режим на работа на микроконтролерот, вклучени се сите електронски компоненти (Слика 16), каде микроконтролерот може да троши струја до 260mA. Доколку за напојување на хардверските компоненти се искористи батерија со капацитет од 2000mAh, батеријата ќе се истроши за помалце од 8 часа. Затоа, за да се заштеди на енергија, по запишувањето на податоците во Firebase Realtime Database, микроконтролерот преминува во режим на спиење во период од 72 минути, кое претсавува максималното време за спиење на микроконтролерот. Постојат три режими на спиење и тоа: Modem Sleep Mode, Light Sleep Mode и Deep Sleep Mode. Режимот на спиење во кој се исклучени најголем дел од електронските компоненти и со кој се заштедува највеќе енергија е Deep Sleep режимот на спиење.

Слика 22. Нормален режим на работа на микроконтролерот ESP32

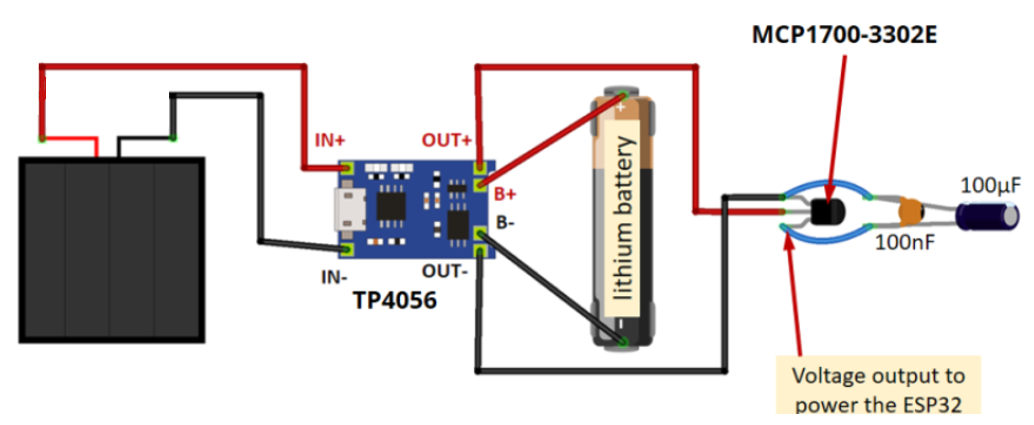
 Во Deep Sleep режимот на спиење се исклучуваат: процесорот, РАМ меморијата, Bluetooth модулот, WiFi примопредавателот и поврзаните периферни уреди, а единствените делови што остануваат вклучени се: RTC (Real Time Clock) меморијата, RTC периферните уреди и ULP копроцесорот. Потрошувачката на струја во овој режим на спиење се движи од 10uA – 0.15mA.

Слика 23. Deep Sleep режим на спиење на микроконтролерот ESP32

За пресметување на времетраењето на батеријата е искористен онлајн калкулатор за проценка на времетраење на батерии кај IoT апликации [1], а резултатот од пресметката е прикажан на Слика 18. Како параметри се зададени времетраењето на извршување на кодот во нормален режим, кој по одредени тестирања, во најлош случај изнесуваше 20 секунди, времетраењето на режимот на спиење кој изнесува 72 минути или 4320 секунди, и најлошите сценарија за потрошувачката на струја при двата режими на работа. Како пример е земена литиумска батерија со капацитет од 2000mAh и излезен напон од 3.7V. Излезниот напон на батеријата кога е целосно наполнета изнесува 4.2V, кој е многу висок и може да го оштети микроконтролерот, затоа во електричното коло е вметнат регулатор на напон (MCP1700-3302E), кој му обезбедува стабилен напон од 3.3V на микроконтролерот. За да се избегнат нагли промени во напонот, на влезот од регулаторот на напон паралелно се поврзани два кондензатори, од кои еден керамички со капацитет од 100nF и еден електролитски со капацитет од 100uF. Поради конверзија на напонот ефективниот капацитет на батеријата се намалува за 20%. Резултатот од пресметката за времетраењето на батеријата изнесува 49 дена и 11 часа, што претставува значително подобрување од првичните добиени 8 часа. Електричната шема за паметното поштенското сандаче е прикажана на Слика 19.

Слика 24. Пресметување на времетраење на батерија

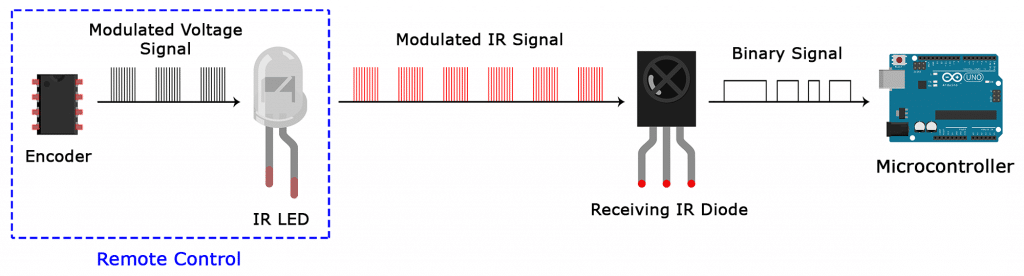
Слика 25. Електрична шема на поштенското сандаче

Како подобрување на системот, за дополнително зголемување на времетраењето на батеријата, може да се искористи батерија со поголем капацитет, така што ако се искористи батерија со капацитет од 10000mAh, времетраењето на батеријата ќе изнесува околу 250 дена. За системот да стане целосно автономен, на ова коло дополнително може да се прикачат мал соларен панел и модул за полнење на литиумски батерии (Слика 20). Излезот од соларниот панел се поврзува со влезот од модулот за полнење на батерии, а излезот од модулот за полнење на батерии се поврзува со литиумската батерија. На овој начин соларниот панел ќе ја полни батеријата секогаш кога ќе има сонце, а времетраењето на батеријата значително ќе се зголеми. При напојување на хардверските компоненти преку батерија, корисно е да го знаеме преостанатиот капацитет на батеријата. Тоа може да се постигне со отчитување на излезниот напон од батеријата, кој се намалува како што се троши батеријата, а за реализација потребно е да се искористи еден аналоген пин од микроконтролерот кој ќе ја отчитува вредноста на излезниот напон од батеријата, а отчитаната вредност потоа ќе се мапира во вредности од 0% - 100% соодветно. Исто така потребно е електронските компоненти да се заштитат од надворешните временски услови, највеќе од вода, а тоа може да се постигне со добра изолација на компонентите или со заменување на постоечките компоненти со водоотпорни компоненти. Друг проблем кој може да се јави е микроконтролерот да не може да се поврзе на WiFi мрежа поради недоволна јачина на WiFi сигналот. Тоа може да се реши со прикачување на надворешна антена на микроконтролерот, која ќе го засили сигналот на WiFi примопредавателот. За подобрување на вредностите добиени од HC-SR04 сензорите, може да се прикачи сензор за мерење на температурата и притисокот на воздухот, кои ќе се искористат во кодот за да се пресмета точната брзина на звучните бранови.

Слика 26. Прикачување на соларен панел и модул за полнење на батерија на колото

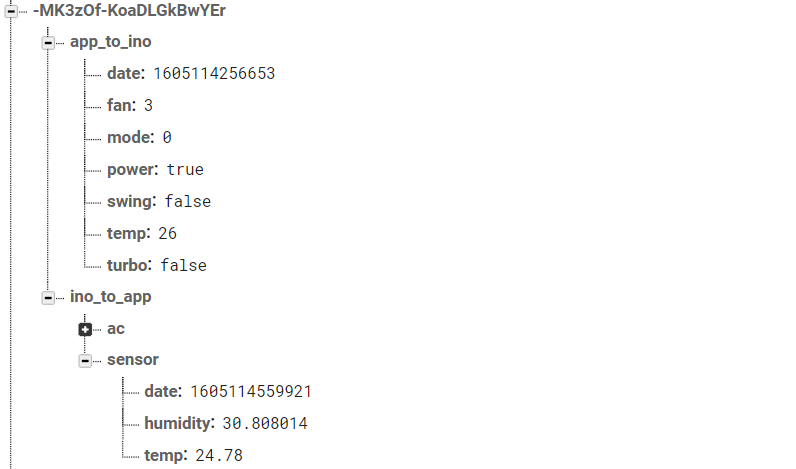
# Паметна клима

Клима уредите комуницираат користејќи ја инфрацрвената технологија. Еден инфрацрвен систем за комуникација се состои од два дела: инфрацрвен предавател и инфрацрвен приемник. Предавателот претставува инфрацрвена LED диода која емитува инфрацрвени сигнали, а приемникот претставува фотодиода која ги регистрира инфрацрвените сигнали и ги конвертира во електрични сигнали. Далечинските управувачи користат инфрацрвени LED диоди за пренос на сигналот од далечинскиот управувач до инфрацрвениот приемник. Меѓутоа, насекаде околу нас постои инфрацрвен шум, предизвикан од телата кои емитуваат топлина (човекот, сијалиците, сонцето и други), кој може да се меша со инфрацрвениот сигнал што се испраќа. За да се избегне интерференција на сигналот, се користи техника наречена модулација на сигналите (Слика 21).

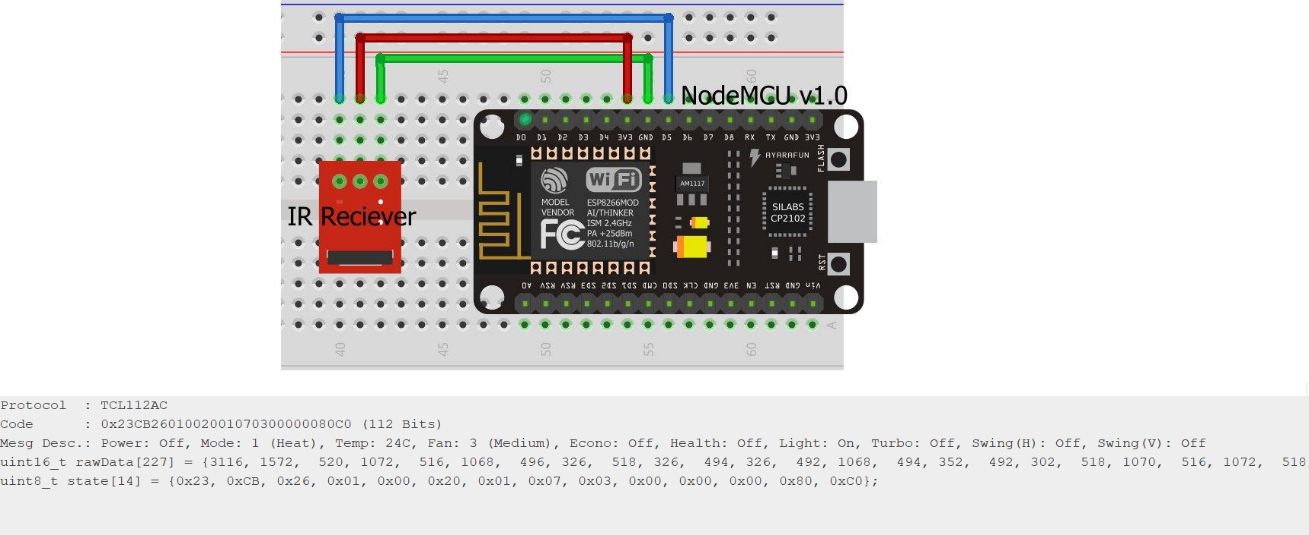
При модулација на инфрацрвените сигнали, енкодерот во далечинскиот управувач го конвертира бинарниот сигнал (добиен при клик на копче на далечинскиот управувач) во модулиран електричен сигнал. Потоа, овој електричен сигнал се испраќа до предавателната инфрацрвена LED диода, која го конвертира електричниот сигнал во модулиран инфрацрвен сигнал. Модулираниот сигнал се испраќа до инфрацрвениот приемник, кој потоа го демодулира сигналот и го конвертира назад во бинарен сигнал, кој се испраќа до микроконтролерот. Начинот на кој еден модулиран сигнал се конвертира во бинарен сигнал е дефиниран со протокол за пренос. Sony, RC5 и NEC се едни од најчестите протоколи за пренос на инфрацрвени сигнали. Различни далечински управувачи испраќаат различни сигнали, затоа е неопходно микроконтролерот да знае како да ги декодира кодовите кои се регистрирани од страна на приемникот.

Слика 27. Модулација на инфрацрвени сигнали

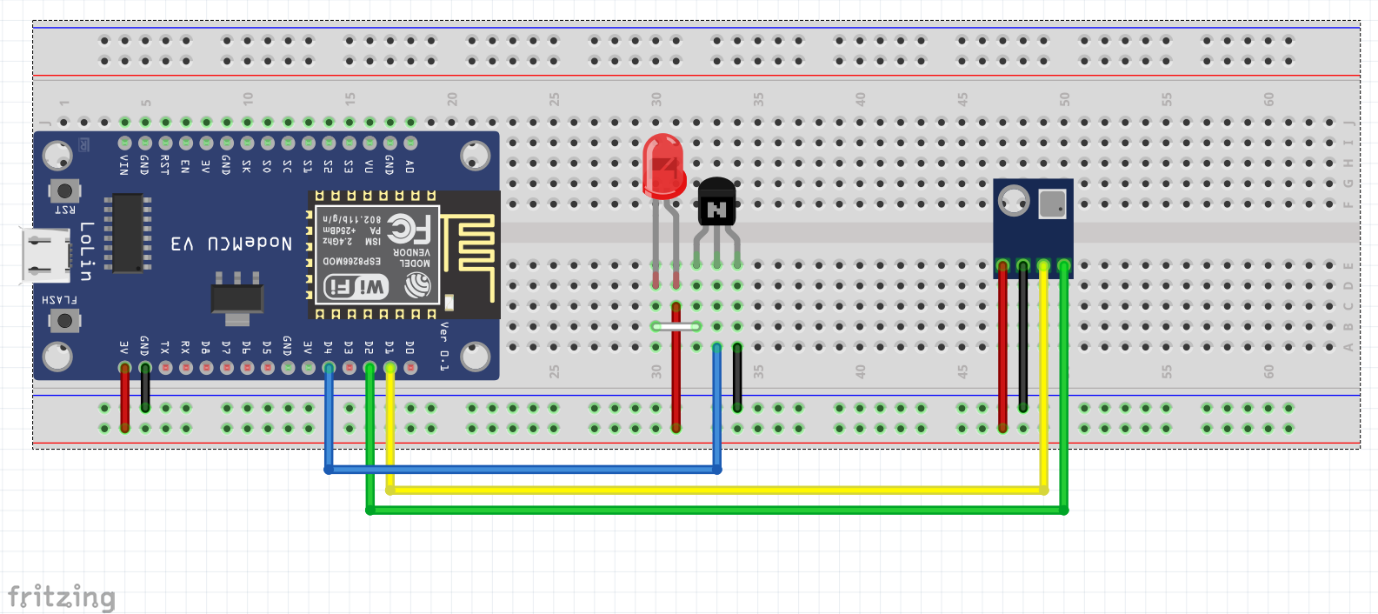
Со познавање на начинот на кои се управуваат клима уредите, искористени се хардверски и софтверски компоненти со кои е изработен далечински управувач кој може да се управува преку мобилната апликација. Хардверските за управување со климата се состојат од: микроконтролерот ESP8266, инфрацрвена LED диода, инфрацрвен приемник TSOP1738, а за набљудување на температурата и влажноста на воздухот во собата кај што се наоѓа клима уредот е искористен сензорот SI7021. За детекција на протоколот за пренос кој го користи климата што сакаме да ја управуваме искористена е Arduino библиотеката ESP8266IRRemote, а за овозможување на комуникацијата помеѓу микроконтролерот и мобилната апликација искористена е Arduino библиотеката FirebaseESP8266.

Принципот на работа е следниов: мобилната апликација запишува податок за управување со климата во Firebase Realtime Database на патеката /{idNaUred}/app\_to\_ino, а микроконтролерот ги отчитува податоците од истата патека и ја управува климата во зависност од добиените податоци. Податоците што се запишуваат во Firebase Realtime Database се прикажани на Слика 22, каде се запишуваат податоци кои кореспондираат со податоци од библиотеката IRRemoteESP8266, и служат за подесување на параметри за работа на клима уредот како што се: режимот на работа на клима уредот (греење, ладење и слично), посакуваната температура, турбо режим на работа, swing режим на работа, подесување на брзината на вентилаторот, како и датумот во UNIX формат за тоа кога е запишан податокот. Микроконтролерот ги отчитува податоците од истата патека, го споредува отчитаниот датум со датумот на претходната добиена инструкција (доколку има таква), и доколку постои разлика во датумите, тој управува со климата користејќи фунцкии од библиотеката ESP8266IRRemote. По успешното управување со клима уредот, микроконтролерот запишува потврдна информација во Firebase Realtime Database на патеката /{idNaUred}/ino\_to\_app/ac, каде го запишува истиот податок кој го добил од мобилната апликација, со цел да ја потврди последната успешно извршена инструкција која ја добил од мобилната апликација. Дополнително, микроконтролерот на секој час запишува податоци за температурата и влажноста на воздухот во Firebase Realtime Database на патеката /{idNaUred}/ino\_to\_app/sensor, добиени од сензорот SI7021. Мобилната апликација ги отчитува сите податоци од Firebase Realtime Database кои се запишувани од страна на микроконтролерот, при што се прикажува на корисниците во реално време.

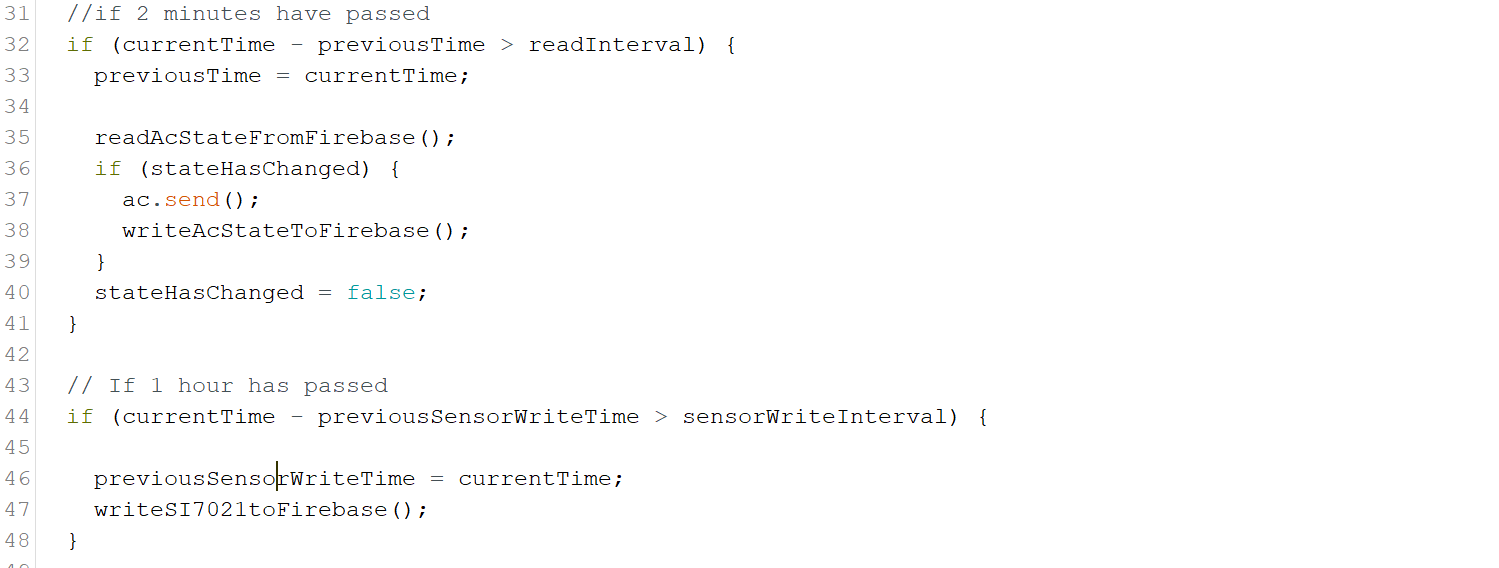
Слика 28. Начин на комуникација меѓу мобилната апликација и микроконтролерот

Најпрво, за детекција на протоколот за пренос кој го користи клима уредот што сакаме да го управуваме, на микроконтролерот е прикачена шема (англиски: Sketch) од Arduino библиотеката IRRemoteESP8266. За користење на таа шема, микроконтролерот е поврзан со инфрацрвениот приемник TSOP1738 (Слика 23), каде дигиталниот пин D4 од микроконтролерот е поврзан со излезниот пин од инфрацрвениот приемник. Потоа, со насочено притискање на копчиња од далечинскиот управувач, кон инфрацрвениот приемник, на излез од серискиот монитор на Arduino IDE се добиени информации за протоколот за пренос на клима уредот што сакаме да го управуваме, кои се прикажани на Слика 23. Според добиените информации за протоколот за пренос, за понатамошно управување со климата, во кодот е вклучена библиотеката IR\_TCL, преку која се искористени објекти и функции за управување на клима уредот.

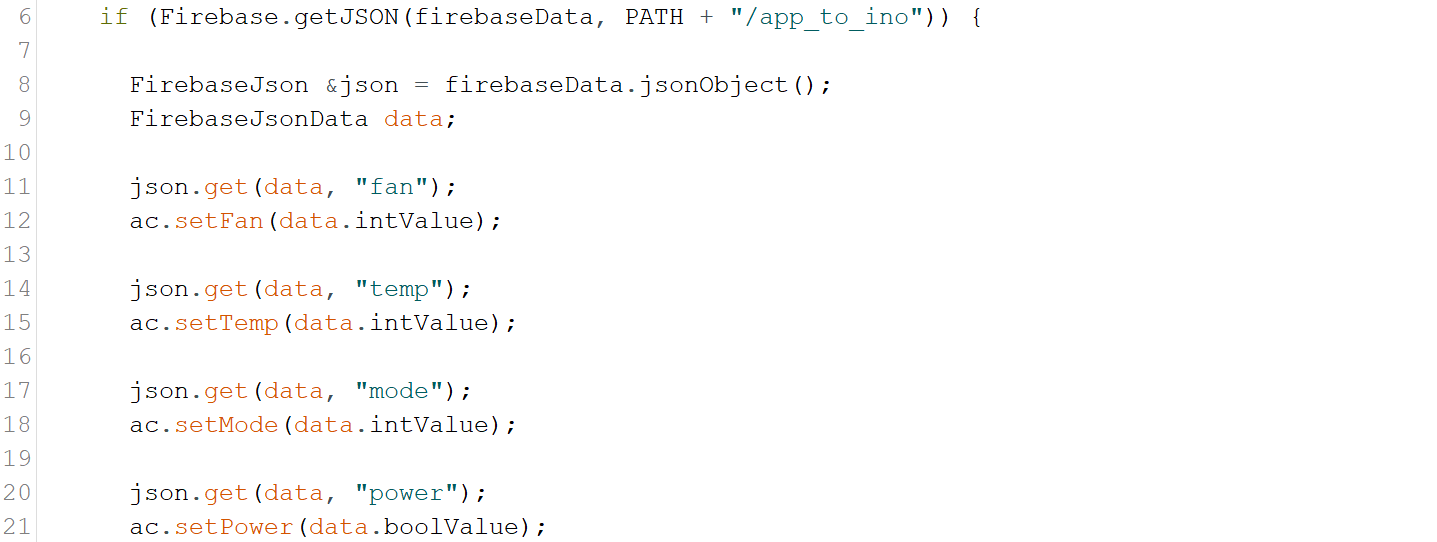
Слика 29. Електрична шема за детекција на протокол за пренос

Откако е добиен протоколот за пренос кој го користи клима уредот што сакаме да го управуваме, од електричното коло е отстранет инфрацрвениот приемник, а наместо него се прикачени NPN транзистор, инфрацрвена LED диода и сензор за мерење на температура и влажност на воздух SI7021. Електричната шема е прикажана на Слика 24. Инфрацрвената LED диода е поврзана со NPN транзисторот, со тоа што таа емитува инфрацрвени сигнали само кога ќе се доведе високо ниво на базата на транзисторот, а базата на транзисторот е поврзан со дигиталниот пин D4 од микроконтролерот. Инфрацрвената диода емитува инфрацрвени сигнали само кога ќе се доведе високо ниво на базата на транзисторот, а тоа е овозможено со повикување на функцијата send() од библиотеката ESP8266IRRemote, која ја повикуваме само кога сакаме да го управуваме клима уредот. SCL и SDA пиновите од сензорот SI7021 се поврзани со пиновите D1 и D2 од микроконтролерот соодветно, а за отчитување на вредностите за температурата и влажноста на воздухот од сензорот, искористена е Arduino библиотеката SI7021.

Слика 30

 Дел од главната програма која ја извршува микроконтролерот е прикажана на Слика 25. Микроконтролерот ги отчитува податоците добиени од мобилната апликација со повикување на функцијата readAcStateFromFirebase() (Слика 26). Отчитувањето се извршува на секои две минути поради тоа што ESP микроконтролерите моментално немаат поддршка за отчитување на податоци од Firebase Realtime Database во реално време. Ова времетраење е избрано со цел да не се премине границата на бесплатниот план од Firebase, која овозможува отчитување на 10GB податоци месечно.

Слика 31. Дел од главната програма која ја извршува микроконтролерот

Отчитаните вредности од Firebase Realtime Database за управување со клима уредот се доделуваат на објектот AC (Air Conditioner) од библиотеката ESP8266IRRemote. Дополнително, се споредуваат отчитаниот датум на инструкцијата со датумот на претходната добиена инструкција, при што глобалната променлива stateHasChanged добива вредност true доколку истите се разликуваат. Клима уредот се управува со повикување на функцијата ac.send(), која се извршува само доколку само доколку отчитаниот датум на инструкцијата и датумот на претходната инструкција се разликуваат (stateHasChanged има вредност true). По управување со клима уредот, микроконтролерот ја повикува функцијата writeStateToFirebase(), која запишува потврдна информација во Firebase Realtime Database дека успешно го управувал клима уредот. Со повикување на функцијата writeSI7021ToFirebase(), која периодично се повикува на секој час, микроконтролерот ги отчитува вредностите за температурата и влажноста на воздухот добиени од сензорот, а потоа ги запишува во Firebase Realtime Database.

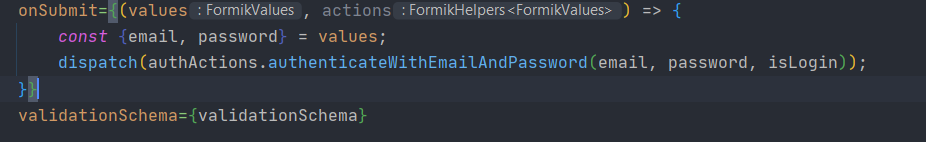
Слика 32. Мапирање на вредности добиени од мобилната апликација во AC објектот од библиотеката ESP8266IRRemote

Микроконтролерот е напојуван преку USB портата, така што струјата ја добива од ѕиден приклучок, а инфрацрвената LED диода е насочена кон клима уредот што сакаме да го управуваме. До неуспешно управување со клима уредот може да дојде доколку микроконтролерот го изгуби пристапот до интернет или доколку снема струја. Кодот е дизајниран на таков начин што при повторно приклучување на интернет, микроконтролерот ќе знае дали има нова инструкција што треба да ја изврши. Неуспешно управување со клима уредот може да дојде и доколку има некаква препрека пред инфрацрвената LED диода. На овој начин ние ќе мислиме дека успешно сме го управувале клима уредот, а всушност инфрацрвениот сигнал бил блокиран од некоја препрека. Затоа, пожелно е микроконтролерот да се постави на место каде што не би имало препреки помеѓу него и клима уредот што сакаме да ја управуваме. Како подобрување на системот, може да се додадат опции вклучување и исклучување на клима уредот во одредени временски интервали во денот или при постигнување на одредени температури добиени од сензорот. Понатаму, системот може да се интегрира со други сервиси од Firebase, како што е сервисот Firebase Machine Learning Kit. На овие начини може да се добие автоматизиран систем, кој ќе ни овозможи да бидеме енергетско ефикасни и еко-фриендлу.

# Изработка на мобилна апликација

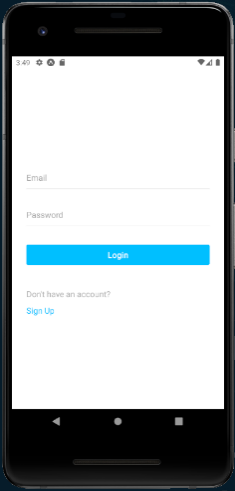
За мобилната апликација изработени се три екрани низ кои корисникот може да се движи и тоа: екран за автентикација, екран за преглед на уредите и екран за управување со клима уреди. Секој екран претставува посебна React компонента, при што во еден момент е прикажан само еден екран.

## Екран за автентикација

За корисникот да може да се најави или регистрира на мобилната апликација, изработен е екран за автентикација, кој се прикажува при првичното отворање на апликацијата. На екранот, за корисникот да може се најави или регистрира, креирана е форма во која тој треба да ги внесе неговите ингеренции за емаил и лозинка. Информацијата за тоа дали корисникот се најавува или регистрира се чува во состојбената променлива isLogin, креирана со hook-от useState, а во зависност од вредноста на состојбената променлива на екранот се прикажува текст соодветен за најава или регистрација. За креирање на формата е искористена библиотеката Formik, а за проверка на валидноста на внесените полиња е искористена библиотеката Yup. Со Yup е дефинирана шема на валидација, која е доделена на Formik, така што доколку корисникот внесе невалиден емаил или лозинка со помалку од 6 карактери, ќе се појави соодветна порака за грешка под невалидното поле, при што копчето за Login или Sign Up ќе биде оневозможено. При клик на Login или Sign Up, се повикува методот onSubmit од Formik, кој е прикажан на Слика 33, и во него се повикува акција за автентикација на корисникот во која се доделуваат внесените вредности за email и лозинка, како и вредноста на состојбената променлива isLogin. Во зависност од состојбената променлива isLogin, акцијата повикува функција во која се испраќа барање до Firebase Authentication сервисот за најава или регистрација на нов корисник. Firebase враќа потврден одговор за успешно автентикување на корисникот само доколку се испратени правилни и точни податоци. Во тој случај, reducer-от кој е задолжен за ракување со акциите за автентикација ја ажурира состојбата во централното складиште од Redux, при што во него ги запишува податоците за корисникот кои се добиени од Firebase. По успешната автентикација, корисникот се пренасочува на почетниот екран, односно на екранот за преглед на уредите. Дополнително, Firebase ги зачувува податоците за корисникот во меморијата од телефонот, така што корисникот ќе биде автоматски најавен при следните отворања на апликацијата. Во случај на неуспешна автентикација, на екранот се појавува соодветна грешка која се добива од серверите на Firebase.

Слика 33. onSubmit метод за автентикација на корисникот

Покрај автентикација со email и лозинка, Firebase Authentication сервисот поддржува повеќе типови на автентикација и тоа: автентикација со акаунти од Google, Facebook, Twitter, GitHub, Apple, Twitter и други, како и автентикација преку мобилен телефон кои понатаму можат да се интегрираат во мобилната апликација и тоа со пишување на една линија код. Покрај тоа, со Firebase Authentication сервисот може да се овозможат опциите за потврда на email, ресетирање на лозинка или промена на податоците од корисникот (email, слика и друго), исто така со пишување на една линија код. Крајниот изглед на екранот за автентикација е прикажан на Слика 34.

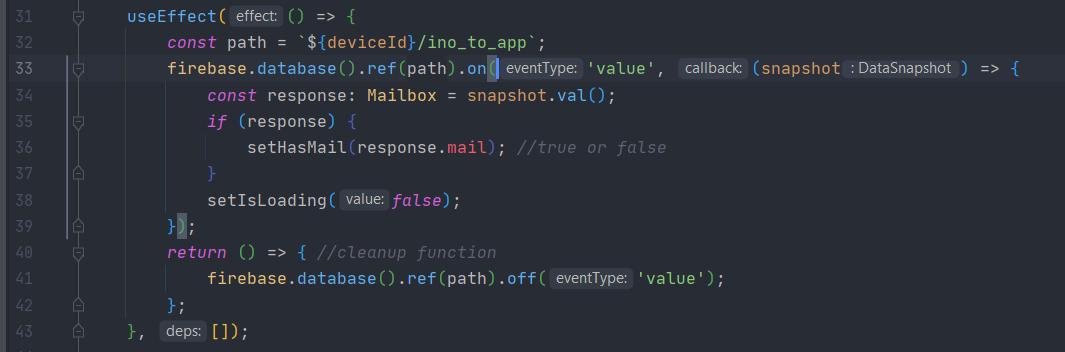


Слика 34. Екран за автентикација на корисници

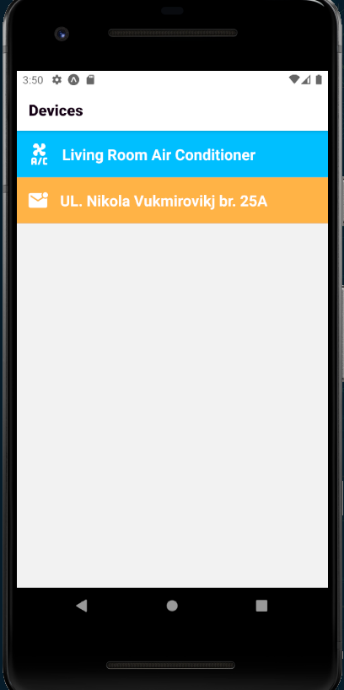
## Екран за прикажување на уредите

По успешната автентикација на корисникот, тој се пренасочува на екранот за прикажување на уредите. Во него, се прикажуваат сите паметни уреди кои се поврзани во домаќинството на корисникот. При иницијалното рендерирање на екранот, се извршува ефект кој испраќа акција за преземање на уредите од Firebase Realtime Database. Акцијата повикува функција која прави барање до Firebase, со кое се преземаат уредите кои се поврзани во домаќинството на корисникот кој ја користи мобилната апликација. Дополнително, корисникот може да направи ново барање за преземање на уредите со освежување (англиски: Refresh) на екранот. Преземените уреди се доделуваат на компонентата FlatList, која ги рендерира уредите на екранот од корисникот. Бидејќи корисникот може да има уреди од тип на поштенско сандаче или клима уред, FlatList компонентата, користејќи го методот renderItem, го проверува типот на секој од преземените уреди, и во зависност од него рендерира различна компонента која го прикажува соодветниот уред.

За прикажување на клима уредите, креирана е компонента со име AirConditionerListItem. Како сопствености на компонентата (Props) се испраќаат податоците за уникатниот идентификатор на клима уредот и името на клима уредот кои се претходно добиени од Firebase Realtime Database. Компонентата го прикажува името на клима уредот што е добиено како Props, а при клик на неа корисникот се пренасочува на екран за управување со клима уредот. Креирана е функција onPressHandler, која го регистрира кликот на корисникот и го пренасочува до соодветниот екран за управување со клима уредот. Во неа се повикува функцијата Navigate, на која се доделени податоците кои се добиени како Props, и служи за пренасочување на корисникот во екранот за управување со клима уредот. За да се избегне непотребно рендерирање на функцијата onPressHandler, искористен е useCallback хоок-от, во која како прв аргумент е доделена функцијата која го извршува пренасочувањето, а како втор аргумент, во низата од зависности е доделена функцијата Navigate.

За прикажување на поштенските сандачиња, креирана е компонента со име MailboxListItem, која на екран ги прикажува името на сандачето и информацијата за тоа дали има пошта во него. При иницијалното вчитување на компонентата, се извршува ефект (Слика 35) во кој се поставува слушател (англиски: Listener) кој ги слуша ажурирањата од страна на микроконтролерот на патеката /{idNaUred}/ino\_to\_app во Firebase Realtime Database. Вредноста на состојбената променлива hasMail, во која се чува податокот за тоа дали има пошта во сандачето, се ажурира според податоците кои периодично се добиваат од Firebase Realtime Database. Бидејќи во овој ефект е поставена претплата на надворешен извор од податоци, потребно е тој да се избрише, а тоа е направено со cleanup функцијата која се извршува кога компонентата ќе се одмонтира, односно кога веќе нема да се прикажува на екран. Во случај кога не би се исчистил слушателот, при повторно рендерирање на компонентата, ефектот ќе се изврши одново, при што ќе се креира нов слушател на истата патека, а тоа ќе доведе до претек на меморијата или паѓање на програмата. Бојата на позадината на компонентата зависи од вредноста на состојбената променлива hasMail, така што кога вредноста на променливата е true, односно кога има пошта во сандачето, бојата на позадината на компонентата е жолта, а во случај кога вредноста на променливата е false, бојата на позадината на компонентата е сина. На овој начин корисникот лесно може да воочи дали има пошта во сандачето. Крајниот изглед на екранот за преглед на уреди е прикажан на Слика 36.

Слика 35. Ефект за поставување на слушател кај поштенското сандаче



Слика 36.

## Екран за управување со клима уред

За корисникот да може да управува со клима уредите во неговото домаќинство креиран е екран за управување со клима уреди (AirCondionerRemoteScreen), а за креирање на овој екран симулиран е изгледот на далечинскиот управувач од клима уредот кој што се управува. Изгледот на екранот е составен од повеќе компоненти и е прикажан на Слика 37.

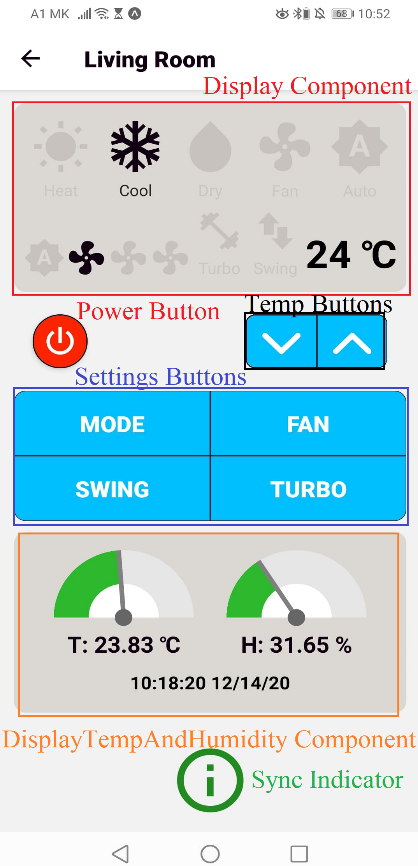
Во екранот се чуваат две состојбени променливи за режимот на работа на клима уредот што се управува. Во нив се чуваат податоци за тоа дали климата е вклучена, модот на работа на клима уредот (ладење, греење, вентилатор, автоматски режим на работа и Dry режим на работа), температурата што треба да ја постигне клима уредот, турбо режим на работа и swing режим на работа. Разликата помеѓу двете состојбени променливи е во тоа што едната состојбена променлива се испраќа од мобилната апликација до микроконтролерот со тоа што се запишува во Firebase Realtime Database на патеката /{idNaUred}/app\_to\_ino и служи за управување со клима уредот, а другата состојбена променлива е таа што се чита од Firebase Realtime Database од патеката /{idNaUred}/ino\_to\_app/ac и претставува потврдна информација која се добива од микроконтролерот откако тој успешно ќе управува со клима уредот. Покрај овие податоци, во состојбените променливи се чува времето во UNIX формат за тоа кога тие се запишани во Firebase Realtime Database, кое понатаму се користи за да се провери дали состојбените променливи се синхронизирани. Поради поголемата комплексност на состојбените променливи, тие се креирани со користење на useReducer hook-от.

Display компонентата претставува мал екран за прикажување на моменталниот режим на работа на климата. Режимите на работа кои се прикажуваат се креирани со компонентата ModeIcon во која како Props се доделуваат иконата и текстот кои треба да се прикажат на екран, како и податок за тоа дали тој режим на работа е моментално активен. Компонентата ModeIcon е транспарентна во случај кога таа рендерира режим на работа не е активен во моментот. Во случај кога клима уредот е исклучен, целата Display компонента е транспарентна. Компонентата Display се ре-рендерира со секоја нова промена на сосотјбената променлива за управување со клима уредот така што секогаш ја покажува последната инструкција која е испратена до микроконтролерот.

За управување со клима уредот креирани се повеќе копчиња кои при клик извршуваат функција која повикува акција за менување на вредноста на состојбената променлива за управување со клима уредот. Меѓутоа, бидејќи се симулирани функционалностите на далечинскиот управувач од клима уредот што се управува, функциите имаат одредени ограничувања како што се: температурата може да има вредности само од 16 – 31 °C, турбо режимот на работа може да биде вклучен само при греење, ладење или Dry мод на работа, и јачината на вентилаторот не може да се менува при Dry мод на работа. Со цел да се избегне непотребно рендерирање на функциите, за секоја од нив е искористен useCallback hook-от. Како прв аргумент на hook-от е доделена функцијата која ја повикува акцијата за менување на вредноста на состојбената променлива за управување со клима уредот, а како втор аргумент е доделена низа од променливи од кои зависи функцијата. На пример функцијата за менување на температурата (onTempChangeHandler) зависи само од што се чува во состојбената променлива, а функцијата за подесување на турбо режимот на работа зависи и од модот на работа на климата.

Индикаторот за синхронизација покажува дали корисникот успешно управувал со клима уредот. Креирана е функција areStatesSynced во која како аргументи се доделуваат вредностите на двете состојбени променливи за режимот на работа на климата. Во случај кога двете променливи имаат иста вредност, односно кога корисникот испратил инструкција за управување со клима уредот и микроконтролерот вратил потврдна информација за успешно управување со клима уредот, индикаторот има зелена боја, а во спротивно тој има жолта боја.

* Submit ac state
* Synchronise Indicator 2 minutes
* DisplayTemperatureAndHumidity (Props, invalid values, state variables, effect & cleanup Funtion)



# Cena?

# Заклучок

# Користена литература