|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** |  |

Јован Јокић

**Дизајн и имплементација IoT-базираног система за праћење еколошких уређаја са интеграцијом података у реалном времену**

Дипломски рад

- Основне академске студије -

Нови Сад, 2024.

|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Датум: |
|  |
| **ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ (BACHELOR) РАДА** | Лист: |
| 1/1 |

*(Податке уноси предметни наставник - ментор)*

| Врста студија: | **Основне академске студије** |
| --- | --- |
| Студијски програм: | **Софтверско инжењерство и информационе технологије** |
| Руководилац студијског програма: | **проф. др Мирослав Зарић** |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | **Јован Јокић** | Број индекса: | **SV47/2020** |
| Област: | **Примењене рачунарске науке и информатика** | | |
| Ментор: | **Проф. др Милан Видаковић** | | |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:   * проблем – тема рада; * начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; * литература | | | |

**НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:**

|  |
| --- |
| **Дизајн и имплементација IoT-базираног система за праћење еколошких уређаја са интеграцијом података у реалном времену** |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| 1. Анализирати стање у области.  2. Израдити спецификацију захтева софтверског решења.  3. Израдити спецификацију дизајна софтверског решења.  4. Имплементирати софтверско и хардверско решење према израђеној спецификацији.  5. Демонстрирати имплементирано решење.  6. Документовати (1), (2), (3), (4) и (5). |

|  |  |
| --- | --- |
| Руководилац студијског програма: | Ментор рада: |
|  |  |

|  |
| --- |
| Примерак за:  - Студента;  - Ментора |

**Садржај**

[1 Увод 5](#_Toc175263881)

[2 Oснове IoT система 6](#_Toc175263882)

[2.1 Увод у IoT системе 6](#_Toc175263883)

[2.2 Архитектура IoT система 6](#_Toc175263884)

[2.3 Комуникациони протоколи и стандарди 6](#_Toc175263885)

[2.4 Сигурност у IoT системима 6](#_Toc175263886)

[2.5 Примене IoT система 7](#_Toc175263887)

[3 Технологије и концепти 8](#_Toc175263888)

[3.1 Git 8](#_Toc175263889)

[3.2 PostgreSQL 9](#_Toc175263890)

[3.3 TimescaleDB 10](#_Toc175263891)

[3.4 Python 12](#_Toc175263892)

[3.4.1 REST (Representational State Transfer) 13](#_Toc175263893)

[3.4.2 JWT (JSON Web Token) 14](#_Toc175263894)

[3.4.3 WebSocket 16](#_Toc175263895)

[3.4.4 mDNS (Multicast DNS) 17](#_Toc175263896)

[3.4.5 Pydantic 18](#_Toc175263897)

[3.4.6 SQLAlchemy 18](#_Toc175263898)

[3.4.7 FastAPI 19](#_Toc175263899)

[3.5 Arduino IDE 20](#_Toc175263900)

[3.6 ESP модули 21](#_Toc175263901)

[3.6.1 ESP32 22](#_Toc175263902)

[3.6.2 ESP8266 22](#_Toc175263903)

[3.7 Flutter 23](#_Toc175263904)

[3.8 Docker 24](#_Toc175263905)

[4 Мотивација 26](#_Toc175263906)

[5 Спецификација 27](#_Toc175263907)

[5.1 Спецификација захтева 28](#_Toc175263908)

[5.1.1 Функциконални захтеви 28](#_Toc175263909)

[5.1.2 Нефункционални захтеви 32](#_Toc175263910)

[5.2 Спецификација система 33](#_Toc175263911)

[5.2.1 Модел података 33](#_Toc175263912)

[5.2.2 Архитектура система 36](#_Toc175263913)

[6 Имплементација система 39](#_Toc175263914)

[6.1 Паметни уређаји 39](#_Toc175263915)

[6.1.1 Thermo Humid Meter 40](#_Toc175263916)

[6.1.2 Waste Sorter 45](#_Toc175263917)

[6.2 Локални сервер (хаб) 52](#_Toc175263918)

[6.3 Удаљени сервер 58](#_Toc175263919)

[6.4 Клијентска апликација 63](#_Toc175263920)

[7 Демонстрација 67](#_Toc175263921)

[7.1 Пријава на систем 67](#_Toc175263922)

[7.2 Почетна страница 67](#_Toc175263923)

[7.3 Преглед свих корисникових паметних уређаја и њихове доступности 68](#_Toc175263924)

[7.4 Додавање новог паметног уређаја 68](#_Toc175263925)

[7.5 Ажурирање креденцијала хаба 69](#_Toc175263926)

[7.6 Ажурирање мрежних креденцијала паметног уређаја 69](#_Toc175263927)

[7.7 Детаљан преглед паметног уређаја и његових прикупљених података 70](#_Toc175263928)

[7.8 Измена основних информација о паметном уређају 72](#_Toc175263929)

[8 Закључак 73](#_Toc175263930)

[9 Литература 74](#_Toc175263931)

[10 Биографија 75](#_Toc175263932)

# Увод

У савременом друштву, **еколошки проблеми** постају све израженији и захтевају иновативна решења за њихово праћење и управљање. Недостатак **интегрисаних система** који омогућавају прикупљање и анализу података у реалном времену често компликује праћење и оптимизацију **еколошких уређаја**. Мотивација за решавање овог проблема лежи у потреби за ефикасним системима који помажу у очувању животне средине и који су једноставни за коришћење и имплементацију.

Специфичан проблем који се решава у овом дипломском раду је дизајн и имплементација **IoT система** за праћење **еколошких уређаја**, са посебним нагласком на интеграцију података у реалном времену и лакоћу употребе. Систем укључује **удаљени сервер**, **базе података** за кориснике и уређаје, као и **базе података за временске серије** које складиште податке које уређаји шаљу. **Локални сервери (хабови)** у кућама корисника функционишу као мост између уређаја и сервера, при чему се подаци од уређаја шаљу до хаба путем **MQTT протокола**, док се подаци са хаба преносе на удаљени сервер путем **HTTP протокола**.

Овај систем је јединствен по томе што комбинује модерне технологије као што су **FastAPI** за HTTP комуникацију, **Mosquitto** као MQTT брокер, **PostgreSQL** и **TimescaleDB** за управљање подацима, и **Flutter** за развој корисничког интерфејса. Ова комбинација омогућава ефикасну примену у свакодневном животу и лако коришћење. Систем је пројектован да буде не само функционалан већ и изузетно једноставан за коришћење и лако скалабилан, што омогућава једноставно подешавање, управљање и додавање нових типова уређаја.

У првом поглављу представљен је увод у тему рада, укључујући основне концепте и циљеве пројекта. Друго поглавље разматра **теоријске основе** које подржавају развој IoT система, укључујући кључне концепте и принципе. Треће поглавље се бави основним концептима коришћених **софтверских технологија**, као што су FastAPI, Mosquitto, PostgreSQL, TimescaleDB и Flutter. Четврто поглавље детаљно описује **спецификацију система**, укључујући захтеве и функционалности. Пето поглавље фокусира се на **дизајн система**, објашњавајући архитектуру и компоненте. Шесто поглавље обухвата процес **имплементације система**, укључујући техничке аспекте и изазове. Седмо поглавље разматра **унапређења система**, предлоге за будући рад, као и ограничења која су се јавила током пројекта. На крају, у осмом поглављу представљени су **закључци рада**, резимирање постигнутог и нагласак на кључним резултатима и доприносима.

# Oснове IoT система

## Увод у IoT системе

**Интернет ствари (IoT)** представља концепт који омогућава уређајима да буду повезани преко интернета и да размењују податке. IoT системи интегришу физичке уређаје са дигиталним платформама како би омогућили праћење, контролу и оптимизацију различитих процеса у реалном времену. Основне компоненте IoT система укључују **сензоре**, **актуаторе**, **комуникационе протоколе** и **платформе за обраду података**. Овај концепт се развијао током времена и сада обухвата широк спектар примене од паметних домова до индустријске аутоматизације.

## Архитектура IoT система

Архитектура IoT система обично се састоји од више слојева. **Сензорски слој** прикупља податке из физичког окружења. Сензори могу бити различитих типова, као што су температурни, влажностни или сензори за квалитет ваздуха. **Комуникациони слој** осигурава пренос података између уређаја и сервера. Овај слој може користити различите протоколе као што су MQTT, CoAP или HTTP. **Слој обраде података** анализира прикупљене информације и пружа корисничке увиде. Овде се могу применити различите технике анализа података као што су **машинско учење** или **статистичке анализе**. На крају, **кориснички интерфејс** омогућава корисницима да комуницирају са системом и приступају подацима. Овај интерфејс може бити **веб апликација**, **мобилна апликација** или **контролна табла**.

## Комуникациони протоколи и стандарди

**Комуникациони протоколи** играју кључну улогу у IoT системима. **MQTT** (Message Queuing Telemetry Transport) је лаган протокол који се често користи за комуникацију између уређаја и сервера, посебно у окружењима са ограниченим ресурсима. MQTT омогућава низак ниво латенције и ефикасан пренос порука. **CoAP** (Constrained Application Protocol) је пројектован за коришћење у окружењима са ограниченим ресурсима и обухвата опције за ефикасно управљање комуникацијом. CoAP је дизајниран да буде сличан HTTP-у, али оптимизован за окружења са ограниченим ресурсима. **HTTP**, иако је општији протокол, такође се користи у IoT системима за интеракцију са веб апликацијама и сервисима. Он пружа стандартизован начин за размену података преко интернета.

## Сигурност у IoT системима

**Безбедност** је критичан аспект IoT система због великог броја уређаја који размењују осетљиве податке. Основне мере безбедности укључују **аутентификацију** и **ауторизацију** корисника и уређаја, **шифровање података** у преносу и складиштењу, и **редовне безбедносне ажурирања** и закрпе. Аутентификација осигурава да само овлашћени корисници и уређаји могу приступити систему. Ауторизација контролише које акције су дозвољене за одређене кориснике или уређаје. Шифровање обезбеђује да подаци остану приватни током преноса и у складишту. Развој сигурних IoT система захтева пажљиво планирање и имплементацију безбедносних мера на свим нивоима архитектуре.

## Примене IoT система

IoT системи имају широк спектар **применa** у различитим областима. У **паметним домовима**, IoT уређаји могу управљати осветљењем, температуром и сигурношћу. **Паметни градови** користе IoT технологије за управљање саобраћајем, отпадом и јавним осветљењем. У **индустријској аутоматизацији**, IoT уређаји помажу у праћењу и оптимизацији производних процеса. У **здравственој заштити**, IoT уређаји могу пратити виталне знаке пацијената и обезбедити прецизне медицинске податке. Ове примене показују потенцијал IoT система да значајно побољшају квалитет живота и ефикасност различитих система и услуга.

# Технологије и концепти

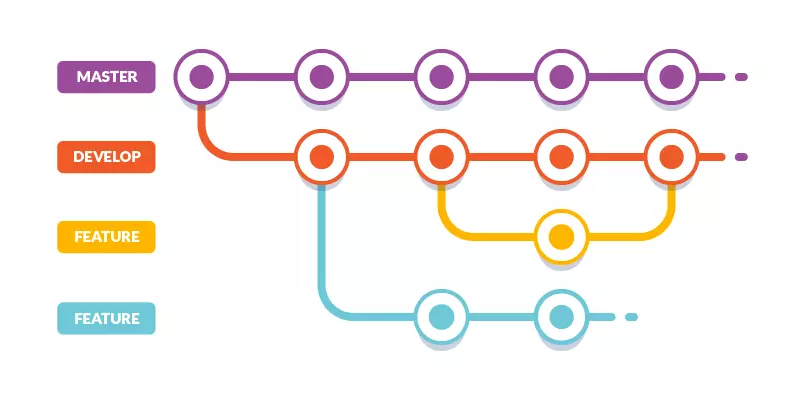
Ово поглавље ће се фокусирати на **технолошке компоненте** и **концепте** који су коришћени у развоју **IoT система** описаног у овом раду. За реализацију система за праћење **еколошких уређаја**, примењене су различите **напредне технологије** и **алати**. У наставку су детаљно представљене **кључне технологије** и **концепти** који чине основу целокупног система.

## Git

**Git** је систем за управљање **верзијама кода** који омогућава праћење и управљање променама у изворном коду током развоја софтверских пројеката. Развијен од стране **Линус Торвалдса**, **Git** је постао стандард у индустрији захваљујући својој **флексибилности** и **ефикасности**.

Основне карактеристике **Git-а** укључују:

* **Историја промена**: **Git** омогућава праћење свих промена у коду, укључујући ко је направио промену и када. Ово омогућава лако враћање на претходне верзије и преглед историје промена.
* **Гранање и спајање**: Подржава рад на различитим **гранама** (**branches**), што омогућава паралелни развој нових функција или исправку грешака без утицаја на главну верзију кода. Када су промене завршене, могу се спојити (**merge**) у главну грану као што је илустровано на слици 1.
* **Сарадња у тиму**: Олакшава рад у тимовима тако што омогућава више људи да истовремено раде на истом пројекту. Промене из различитих извора могу се интегрисати и синхронизовати, чиме се спречавају конфликти и губитак података.
* **Ефикасност и брзина**: Дизајниран је да буде брз и ефикасан, са локалним операцијама које минимизују зависност од мрежних веза и користе ефикасне механизме за компресију података.
* **Подршка за различите врсте репозиторијума**: Подржава рад са локалним и удаљеним **репозиторијумима**, укључујући платформе као што су **GitHub**, **GitLab** и **Bitbucket**, што омогућава лако дељење кода и глобалну сарадњу.
* **Безбедност и интегритет**: Користи **SHA-1 хеширање** за осигурање интегритета података, што гарантује да су сви подаци заштићени од неовлашћених измена.



Слика 1 Илустрација издвајања и спајања грана у систему за верзионирање Git[[1]](#footnote-1)

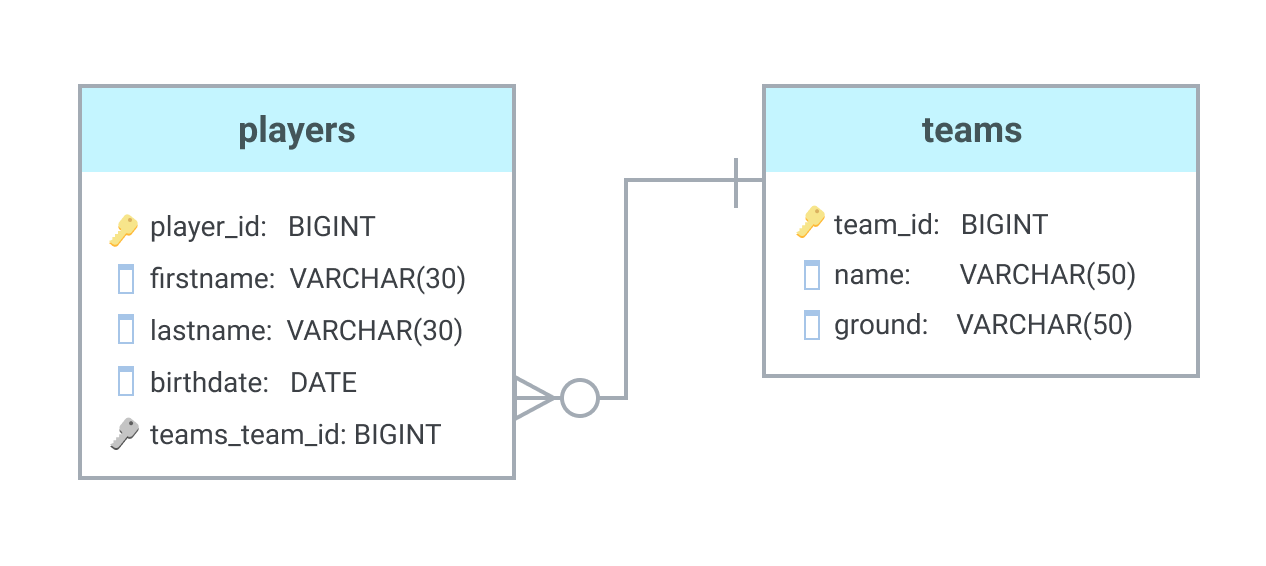
**Git** се интегрише са различитим алатима и платформама за управљање пројектима, као што су **CI/CD системи**, алати за преглед кода и менаџмент пројеката, што га чини основним алатом у савременом развоју софтвера.

## PostgreSQL

**PostgreSQL** је напредна **релациона база података** отвореног кода, позната по својој високој **поузданости** и **флексибилности**. Као један од водећих система за управљање базама података, **PostgreSQL** се користи за обраду и управљање структурираним подацима у различитим апликацијама, укључујући и **IoT системе**.

Карактеристике **PostgreSQL-а** укључују:

* **Поузданост** и **историјска подршка**: **PostgreSQL** има дугу историју у индустрији база података и доказану поузданост. Ова база података користи се у различитим продукционим окружењима и подржава широк спектар апликација, од малих веб сајтова до великих корпоративних система.
* **Сложени релациони модел**: **PostgreSQL** користи релациони модел података који омогућава стварање сложених и повезаних структура података. Овај модел је идеалан за управљање подацима организованим у табелама и њиховим односима. Пример овога можемо видети на слици 2.
* **Разноврсност типова података**: **PostgreSQL** подржава широк спектар типова података, укључујући традиционалне типове као што су текст, бројеви и датуми, као и напредне типове као што су **JSON** и **XML**. Ово омогућава обраду различитих врста информација унутар једне базе података.
* **Транзакциона конкурентност** и **изолација**: Систем пружа напредне механизме за управљање транзакцијама, укључујући изолацију транзакција, што осигурава конзистентност и тачност података чак и када више корисника или процеса раде истовремено.
* **Подршка за индексирање и анализу података**: **PostgreSQL** укључује напредне технике индексирања, као што су **Б-стабла** и **Хаш индекси**, што побољшава перформансе упита. Поред тога, нуди моћне аналитичке функције и функције прозора за дубинску анализу података.
* **Проширива структура и додатци**: **PostgreSQL** нуди могућност проширења функционалности путем додатака и корисничких типова података. На пример, **PostGIS** додатак омогућава рад са просторним подацима, што је корисно у многим напредним применама.
* **Безбедност и контрола приступа**: **PostgreSQL** укључује напредне механизме за управљање безбедношћу, као што су корисничке улоге, права приступа и **шифровање података**, чиме се осигурава заштита и контрола података у складу са захтевима безбедности.

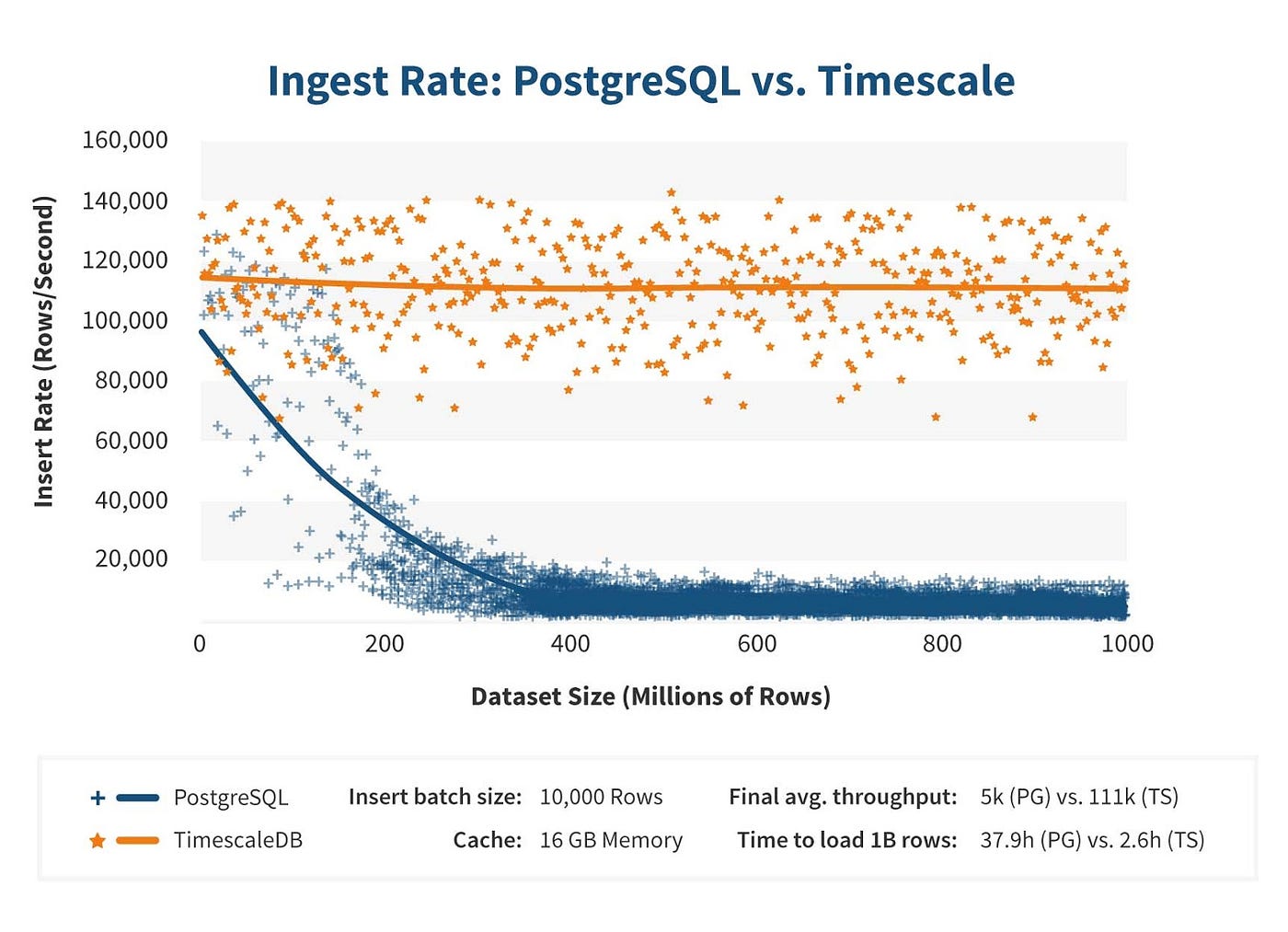


Слика 2 Пример релације две табеле[[2]](#footnote-2)

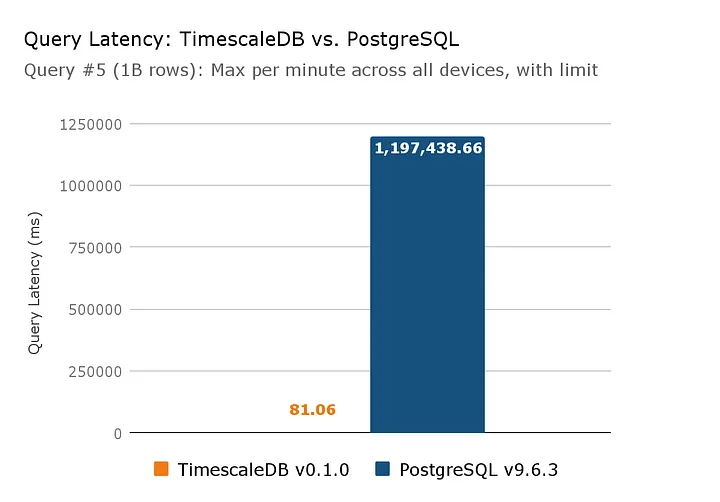
## TimescaleDB

**TimescaleDB** је напредно проширење **PostgreSQL-а** специјално дизајнирано за обраду и анализу података временских серија. Временске серије су подаци који су прикупљени у континуитету и повезани са временом, као што су мерење температуре, вредности са сензора или подаци о потрошњи ресурса. **TimescaleDB** је развијен да понуди побољшану перформансу и ефикасност у управљању овим врстама података.

Једна од кључних карактеристика **TimescaleDB-а** је његова способност да користи **хипертабеле**, што представља напредну структуру за складиштење података. **Хипертабеле** су организоване по временским интервалима и представљају начин за разграничење и индексирање података. Ова архитектура омогућава **TimescaleDB-у** да се ефикасно скалира и управља великим количинама података, што је нарочито корисно у окружењима са великим бројем сензора или података из других извора који се стално генеришу. Поређење база **PostgreSQL** и **TimescaleDB** можемо видети на сликама 3 и 4.



Слика 3 Поређење брзине уписивања у PostgreSQL и TimescaleDB[[3]](#footnote-3)



Слика 4 Поређење брзине читања милион редова из база PostgreSQL и TimescaleDB3

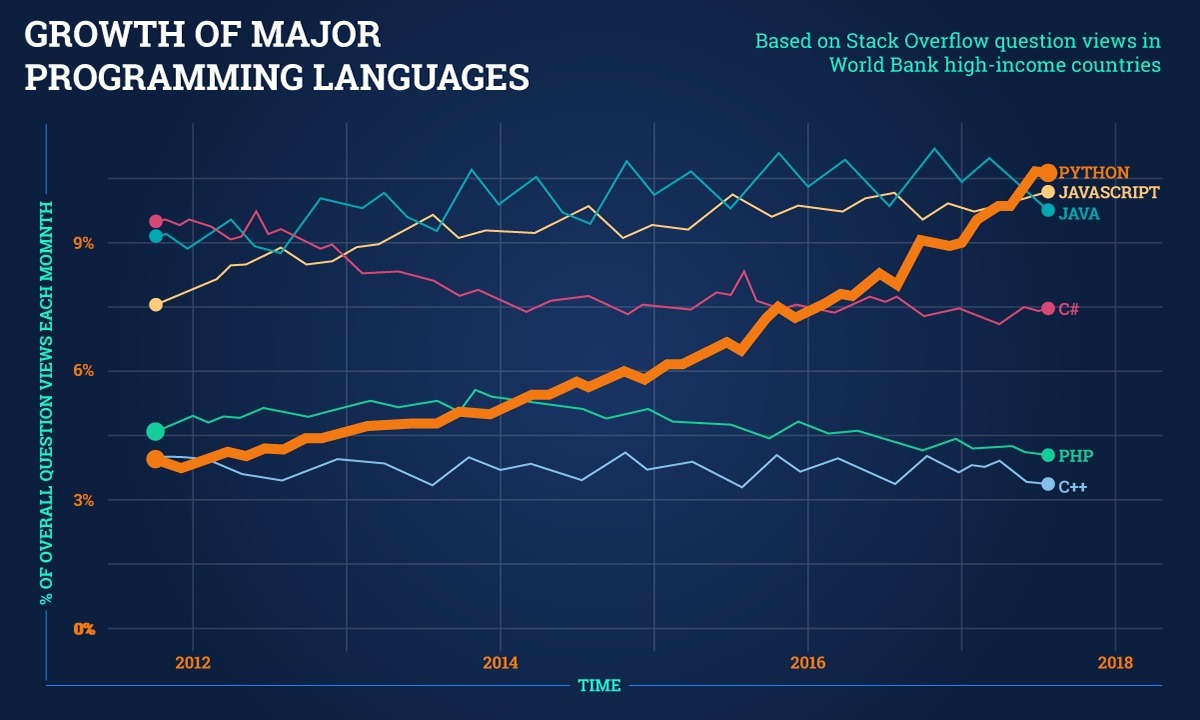
**TimescaleDB** нуди напредне функције за управљање временским серијама, укључујући **агрегацију података по временским интервалима**, **израду временских прозора** и **претраживање података у реалном времену**. Ове функције су интегрисане у **PostgreSQL-ову** SQL синтаксу, што омогућава корисницима да користе познате SQL упите и алате. **TimescaleDB** такође подржава сложене аналитичке операције, као што су укупне, просечне и медијанске вредности, које се могу извршити унутар временских прозора.

Поред тога, **TimescaleDB** користи **хибридни приступ** који комбинује релациони модел података са специјализованим структурама за временске серије. Ова комбинација омогућава **TimescaleDB-у** да користи релационе способности **PostgreSQL-а**, као што су транзакције и подршка за сложене упите, док истовремено нуди оптимизације за рад са временским серијама.

**TimescaleDB** се такође истиче у управљању великим обимом података и нуди могућности за комбиновање различитих типова података, што је корисно за интеграцију података из различитих извора. Њена способност да пружи брзу и прецизну анализу података чини је идеалним решењем за системe који захтевају обраду великих количина података у реалном времену, као што су **IoT системи**, **апликације за праћење и мониторинг**, и друга подручја где је потребна анализа великих количина временских података.

## Python

**Python** је програмски језик познат по својој **једноставности** и **читљивости**, што га чини идеалним за развој софтвера. Његова **интуитивна синтакса** олакшава брзо учење и продуктиван рад, омогућавајући програмерима да се фокусирају на решавање проблема уместо на сложене техничке детаље. Ово је довело до великог раста у популарности **Python-а** последњих година.



Слика 5 Растућа популарност програмског језика Python[[4]](#footnote-4)

Једна од главних предности **Python-а** је његова **богата библиотека пакета** и **радних оквира**. Ове библиотеке, попут **NumPy** и **Pandas** за обраду података, и **Flask** и **Django** за развој веб апликација, значајно убрзавају развој и омогућавају лаку интеграцију различитих функција без потребе за изградњом сваког аспекта система од нуле.

**Python** такође нуди одличну подршку за интеграцију са другим језицима и технологијама, што га чини корисним у **мултијезичним окружењима**. Његова способност да комуницира са различитим системима и библиотекама олакшава изградњу комплексних решења и повећава функционалност система.

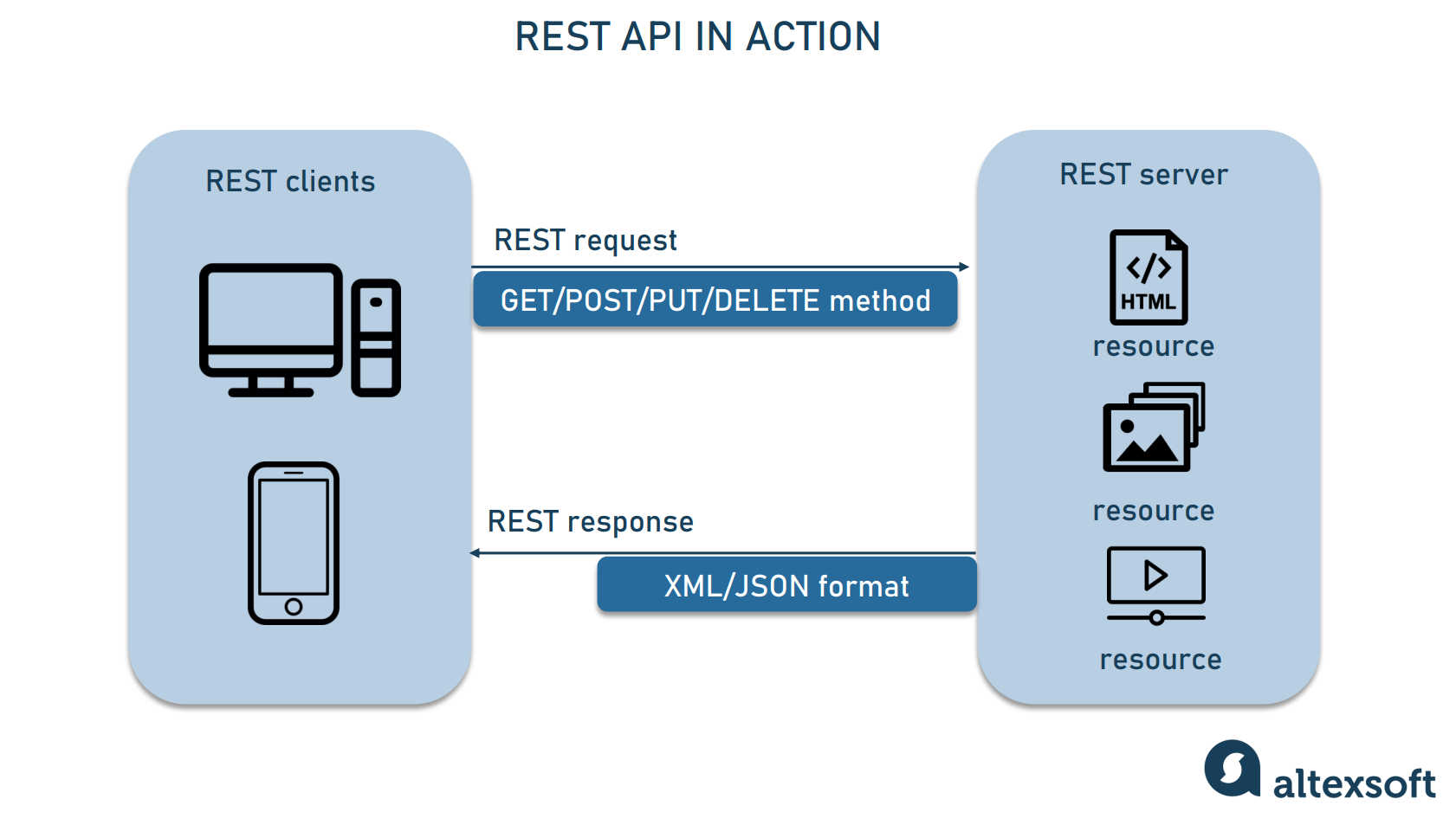
Са подршком за разне **парадигме програмирања**, укључујући **објектно-оријентисано** и **функционално програмирање**, **Python** омогућава програмерима да изаберу најприкладнији приступ за решавање конкретних проблема. Његова једноставност и моћни алати чине га идеалним избором за развој софтвера и одржавање система.

### REST (Representational State Transfer)

**REST (Representational State Transfer)** је архитектонски стил за развој веб сервиса који користи стандардне HTTP методе као што су **GET**, **POST**, **PUT** и **DELETE**. Основна идеја REST-а је управљање ресурсима представљеним у форматима као што су **JSON** или **XML**.

**Кључне карактеристике REST-а су:**

* **Стање се не чува**: Сви подаци потребни за обраду захтева укључени су у сам захтев. Сервер не чува стање између захтева, што олакшава скалабилност и смањује сложеност.
* **Клијент-сервер архитектура**: Клијент и сервер су одвојени, што омогућава независни развој клијентских и серверских компоненти. Клијент шаље захтеве серверу преко HTTP-а, а сервер обрађује захтеве и шаље одговоре.
* **Кеширање**: REST подржава кеширање одговора ради побољшања перформанси. Сервер може назначити које ресурсе је могуће кеширати, чиме се смањује број захтева који се шаљу серверу и убрзавају се одговори.
* **Хипермедија као покретач стања апликације (HATEOAS)**: Одговори сервера садрже хипер-везе ка другим ресурсима. Ово омогућава клијентима да динамички откривају доступне акције и ресурсе, чиме се олакшава навигација и интеграција.
* **Ресурси и идентификација**: Ресурси су идентификовани јасним **URL-овима**. Сваком ресурсу се приступа преко његовог **URI (Uniform Resource Identifier)**, што омогућава јасну и лаку идентификацију ресурса.
* **Стандардни формати и кодови**: REST користи стандардне **HTTP кодове статуса** (као што су 200 - OK, 404 - Not Found) и формате података (као што су JSON и XML) за размену информација, што омогућава лако разумевање и интероперабилност.
* **Модуларност**: REST омогућава разбијање услуга на више крајњих тачака (**endpoints**), што чини системе модуларнијим и лакшим за управљање и одржавање. Ово такође омогућава еволуцију и проширење услуга без утицаја на друге делове система



Слика 6 Изглед комуникације између клијента и сервера путем REST-а[[5]](#footnote-5)

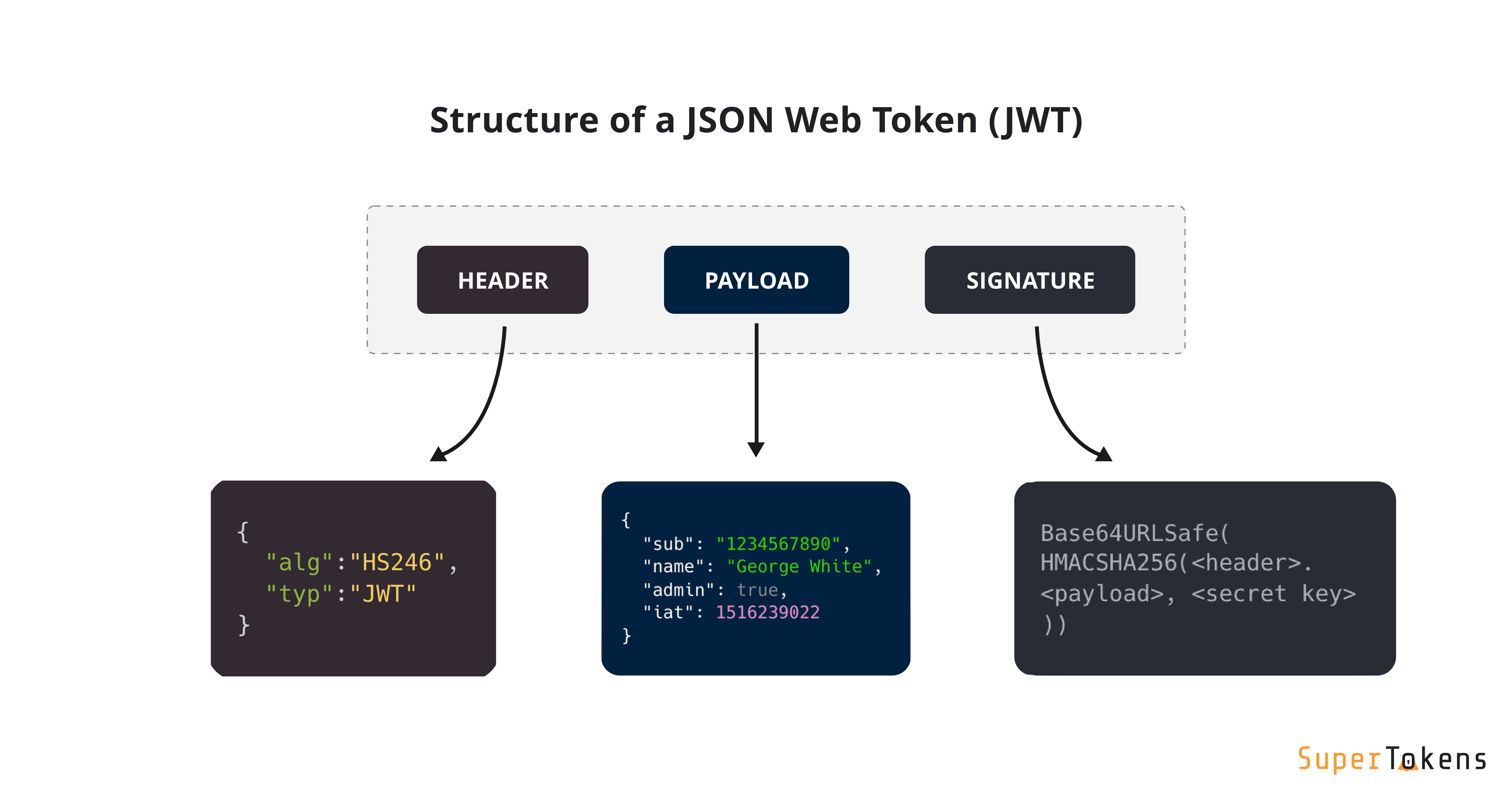
Ове карактеристике чине REST стил погодним за изградњу флексибилних, скалабилних и лако одрживих веб услуга.

### JWT (JSON Web Token)

**JWT (JSON Web Token)** је стандард за сигурну размену информација између страна у форми **JSON** објекта. Овај стандард је широко коришћен у веб апликацијама за управљање **аутентификацијом** и **ауторизацијом**. JWT је погодан за пренос података који су потписани и, ако је потребно, шифровани ради сигурности.

JWT се састоји од три главна дела (приказана и на слици 7):

* **Заглавље**: Заглавље обично садржи информације о врсти токена и алгоритму који се користи за потписивање. На пример, заглавље може указивати на то да се користи **HMAC SHA256** алгоритам за потписивање. Заглавље је кодирано у **Base64Url** формату и представља први део JWT-а.
* **Терет (Payload)**: Терет, познат и као "**тврђење**" (claims), садржи податке о идентитету корисника и његовим правима приступа. Ови подаци могу укључивати кориснички **ID**, улогу, и друга својства која могу бити корисна за аутентификацију и ауторизацију. Терет је такође кодиран у **Base64Url** формату и представља други део JWT-а.
* **Потпис**: Потпис се генерише коришћењем криптографских алгоритама, што осигурава интегритет и аутентичност података. Потпис се користи за проверавање да ли су подаци унутар токена непромењени и аутентични. Он се ствара комбиновањем заглавља и терета са тајним кључем или приватним кључем у зависности од коришћеног алгоритма.



Слика 7 Пример делова Json Web Token-а[[6]](#footnote-6)

JWT нуди више кључних предности:

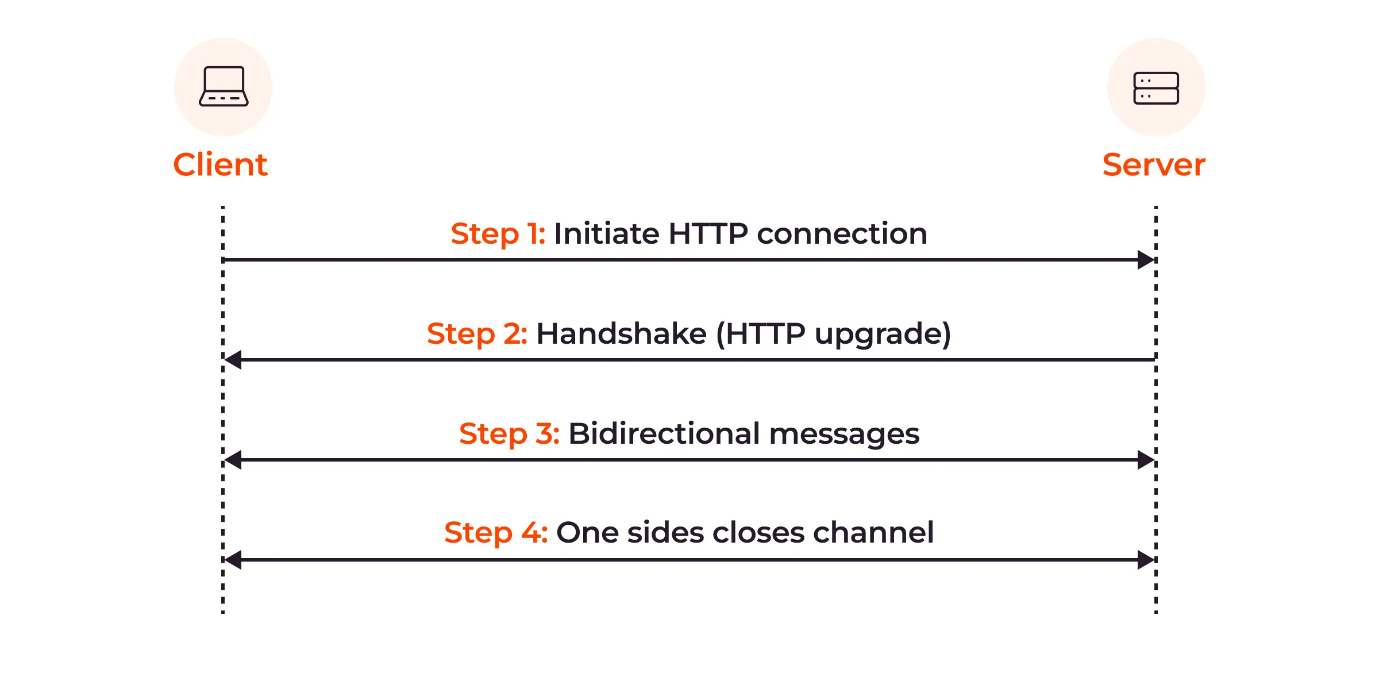
* **Безбедност**: Једна од главних предности је сигурност. Користећи стандардне криптографске алгоритме, JWT осигурава да су подаци унутар токена аутентични и непромењени. Потписивање токена обезбеђује да се подаци не могу модификовати без откривања.
* **Рок трајања и обнављање**: Подешавање истека токена и његово обнављање омогућава ограничење приступа подацима на одређени временски период, чиме се избегава потреба за поновном аутентификацијом. Параметри као што су exp (време истека) могу се укључити у терет за контролу трајања токена.
* **Преносивост и компатибилност**: JWT је лак за пренос и компатибилан са различитим платформама и програмским језицима, што омогућава једноставну интеграцију у различите системе, укључујући дистрибуиране системе и микросервисе.

JWT пружа једноставан и ефикасан начин за управљање аутентификацијом и ауторизацијом, осигуравајући сигурно и поуздано средство за размену података у савременим веб и мобилним апликацијама.

### WebSocket

**WebSocket** представља комуникациони протокол који омогућава **двосмерну** и **перзистентну комуникацију** између клијента и сервера. За разлику од традиционалних **HTTP комуникација**, које функционишу на принципу захтева и одговора, **WebSocket** протокол омогућава непрекидну везу у којој оба краја могу активно размењивати податке у **реалном времену**.

Једна од кључних предности **WebSocket-а** је његова способност да одржи **дуготрајну везу**. Након иницијалног „**handshake**“ захтева, који се изводи коришћењем **HTTP протокола**, веза се успоставља и остаје отворена, што омогућава слободну и константну размену података без потребе за поновним успостављањем конекције. Ова особина, приказана на слици 8, је од велике користи за апликације које захтевају честу и актуелну размену информација, као што су **апликације за размену порука**, **онлајн игре** или **системи за праћење података у реалном времену**.



Слика 8 Успостављање комуникације и размена порука путем WebSocket протокола[[7]](#footnote-7)

Процес комуникације преко **WebSocket-а** започиње **HTTP "handshake" захтевом** који иницира договор о преласку на **WebSocket протокол**. Након успешног handshaking-а, HTTP протокол се замењује са **WebSocket протоколом**, што омогућава директну и ефективну размену порука. Подаци се могу слати у **бинарном** или **текстуалном формату**, што обезбеђује флексибилност у врсти преношених информација, укључујући текстуалне поруке као што су **JSON** или **XML**, као и бинарне податке.

**Безбедност комуникације** преко **WebSocket-а** може се побољшати коришћењем **HTTPS протокола**, који пружа **шифровање** и **аутентификацију**. Иако сам **WebSocket протокол** не укључује ове безбедносне карактеристике, његова интеграција са **HTTPS-ом** омогућава сигурно управљање осетљивим информацијама и заштиту од потенцијалних безбедносних претњи.

**WebSocket** је изузетан алат за развој **веб апликација** које захтевају сталну и брзу размену података. Његове карактеристике, укључујући способност одржавања дуготрајне везе и подршку за разне формате података, чине га погодним избором за апликације које захтевају комуникацију у реалном времену.

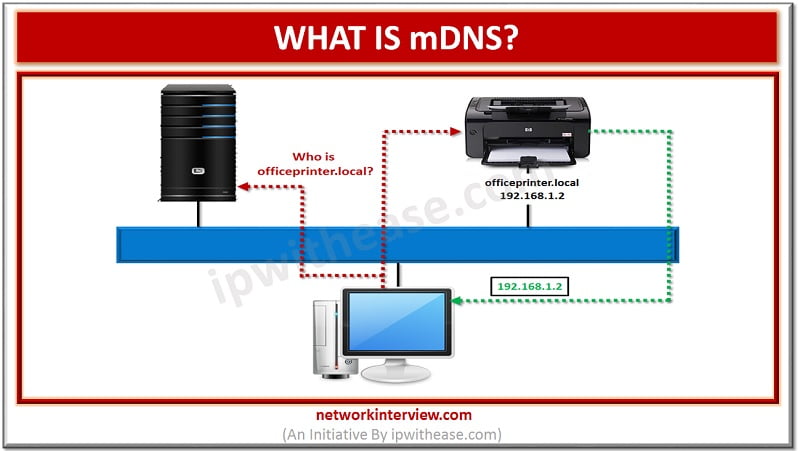
### mDNS (Multicast **DNS**)

**mDNS (Multicast DNS)** је протокол који омогућава решавање имена у мрежи користећи **мултикаст** комуникацију, што омогућава уређајима у локалној мрежи да се идентификују и комуницирају без потребе за централним **DNS сервером**. Овај протокол је део **DNS-а (Domain Name System)** и користи мултикаст пакете за размену информација о именима у мрежи, чиме се омогућава аутоматска конфигурација и откривање сервиса у локалним мрежама.

mDNS функционише тако што сваки уређај у мрежи шаље захтеве за решавање имена и одговоре користећи мултикаст адресе. На тај начин, уређаји који су повезани на исту локалну мрежу могу једноставно пронаћи једни друге и разменити информације без потребе за централизованим сервером. Овакав приступ је користан у окружењима где се уређаји често повезују и искључују, као што су **кућне** или **малe пословне** мреже.

**Кључне карактеристике mDNS-а** укључују:

* **Динамичко решавање имена**: Омогућава уређајима да аутоматски открију и идентификују друге уређаје у мрежи без потребе за ручном конфигурацијом, што је визуализовано на слици 9.
* **Мултикаст комуникација**: Податке о именима шаље коришћењем мултикаст адреса, што омогућава свим уређајима у мрежи да примају и обрађују те поруке.
* **Подршка за аутоматску конфигурацију**: Употреба mDNS-а омогућава уређајима да се лако и брзо конфигуришу у мрежи, што је корисно за сценарије са често променљивом конфигурацијом.



Слика 9 Начин рада mDNS у оквиру мреже[[8]](#footnote-8)

mDNS је посебно користан у окружењима која користе **IoT уређаје** и системе који захтевају лако откривање и комуникацију између различитих компоненти. Његова способност да ради у локалној мрежи без потребе за спољним DNS сервером чини га идеалним за употребу у динамичким и разноликим мрежним окружењима.

### Pydantic

**Pydantic** је библиотека у Python-у која поједностављује валидацију и серијализацију података користећи типске наговештаје. Основна функција Pydantic-а је да омогући **дефинисање класа са типом**, чиме се аутоматски проверавају и обрађују подаци према дефинисаним типовима. Ова библиотека пружа **механизме за валидацију улазних података и конверзију између различитих формата**, као што је JSON, и Python објеката. Пример модела уређаја дефинисан помоћу Pydantic библиотеке са опционим атрибутима можемо видети на листингу 1.

from pydantic import BaseModel  
from typing import Optional  
  
class Device(BaseModel):  
 id: UUID  
 title: str  
 description: str  
 owner\_id: Optional[UUID] = None  
 type: DeviceType  
 is\_online: bool  
 linked\_timestamp: Optional[datetime.datetime] = None

Листинг 1 Дефиниција модела уз помоћ Pydantic-а

Pydantic користи типске наговештаје да аутоматски провери исправност података и изврши трансформације, што помаже у одржавању кода чистим и безбедним. На пример, ако је очекивани тип података int, Pydantic ће покушати да конвертује примљене вредности у одговарајући тип, а у случају неуспеха ће бацити грешку. Ово је посебно корисно у развоју веб апликација и API-ја, где осигурава да подаци који долазе у систем буду у исправном формату пре него што буду обрађени.

Интеграција са **FastAPI-јем** представља једну од најзначајнијих предности Pydantic-а. Користећи Pydantic за дефинисање модела података, FastAPI може лако управљати валидацијом захтева и одговора, што поједностављује рад са сложеним структуром података. Ова интеграција омогућава аутоматско генерисање документације и побољшава развојне токове, чинећи рад са подацима ефикаснијим и поузданијим.

### SQLAlchemy

**SQLAlchemy** је библиотека за рад са релационим базама података у Python-у, која пружа **објектно-релационо мапирање (ORM)** и алате за рад са SQL упитима. Омогућава развој и управљање базом података кроз Python објекте и класе, што поједностављује рад са подацима и чини код читљивијим и одрживијим.

Основна карактеристика SQLAlchemy-а је његов **ORM слој**, који омогућава програмерима да дефинишу моделе података као Python класе. Ове класе су затим повезане са табелама у бази података, што омогућава рад са подацима на објектно оријентисан начин. Поред ORM-а, SQLAlchemy пружа и алат за изградњу SQL упита, који омогућава директно манипулисање базом података користећи SQL синтаксу. У следећа два исечка из кода, листинзи 2 и 3, приказана је дефиниција модела базног корисника који се мапира на табелу корисника у бази података, као и добављање корисника из базе помоћу његове „email“ адресе.

from sqlalchemy.orm import relationship, declarative\_base  
  
postgres\_base = declarative\_base()  
class BaseUser(postgres\_base):  
 \_\_tablename\_\_ = "users"  
  
 id = Column(UUID(as\_uuid=True), primary\_key=True, default=uuid.uuid4, nullable=False)  
 email = Column(String, unique=True, index=True, nullable=False)  
 hashed\_password = Column(String, nullable=False)  
 is\_active = Column(Boolean, default=True, nullable=False)  
  
 role = Column(Enum(Role), nullable=False)  
 \_\_mapper\_args\_\_ = {  
 'polymorphic\_on': role  
 }

Листинг 2 Дефинисање модела базног корисника који се мапира на табелу у бази података

from sqlalchemy.orm import Session  
from app.entities import models  
  
def get\_base\_user\_by\_email(db: Session, email: str):  
 return db.query(models.BaseUser).filter(models.BaseUser.email == email).first()

Листинг 3 Добављање корисника из базе података помоћу његове email адресе користећи SQLAlchemy

SQLAlchemy нуди флексибилност у управљању базама података, подржавајући различите релационе базе података као што су PostgreSQL, MySQL и SQLite. Ово га чини погодним за различите врсте пројеката и окружења. Поред тога, SQLAlchemy обезбеђује подршку за сложене транзакције, упите и управљање везама, што помаже у изградњи робусних апликација.

### FastAPI

**FastAPI** је напредан радни оквир за развој веб API-ја у Python-у, дизајниран да обезбеди **брзу** и **једноставну** изградњу апликација. Овај радни оквир се ослања на стандард **OpenAPI** за документацију и **Pydantic** за валидацију података, што омогућава аутоматско генерисање и ажурирање документације API-а као и валидацију улазних података на основу Python типова.

Кључна карактеристика FastAPI-а је његова **асинхрона природа**, која користи async и await синтаксу за обраду захтева. Ово омогућава високе перформансе и способност да се истовремено обради велики број захтева, што је корисно за апликације које захтевају брзу реакцију и реално време. FastAPI такође подржава различите функције као што су **аутентификација**, **овлашћење** и **управљање грешкама**, што га чини изузетно флексибилним за различите примене.

Тиме што комбинује једноставност и ефикасност, FastAPI је постао све популарнији избор за развој **микросервиса** и апликација, омогућавајући брзу интеграцију и лако одржавање кода. Његова способност да аутоматски генерише документацију и валидацију података значајно убрзава развојни процес и побољшава квалитет кода. Минималан код потребан за креирање HTTP сервера са једним „**endpoint-ом**“ приказан је у листингу 4.

from fastapi import FastAPI  
  
app = FastAPI()  
  
@app.get("/")  
async def root():  
 return {"message": "Hello World"}

Листинг 4 Минималан код потребан за креирање endpoint-а помоћу FastApi-а у програмском језику Python

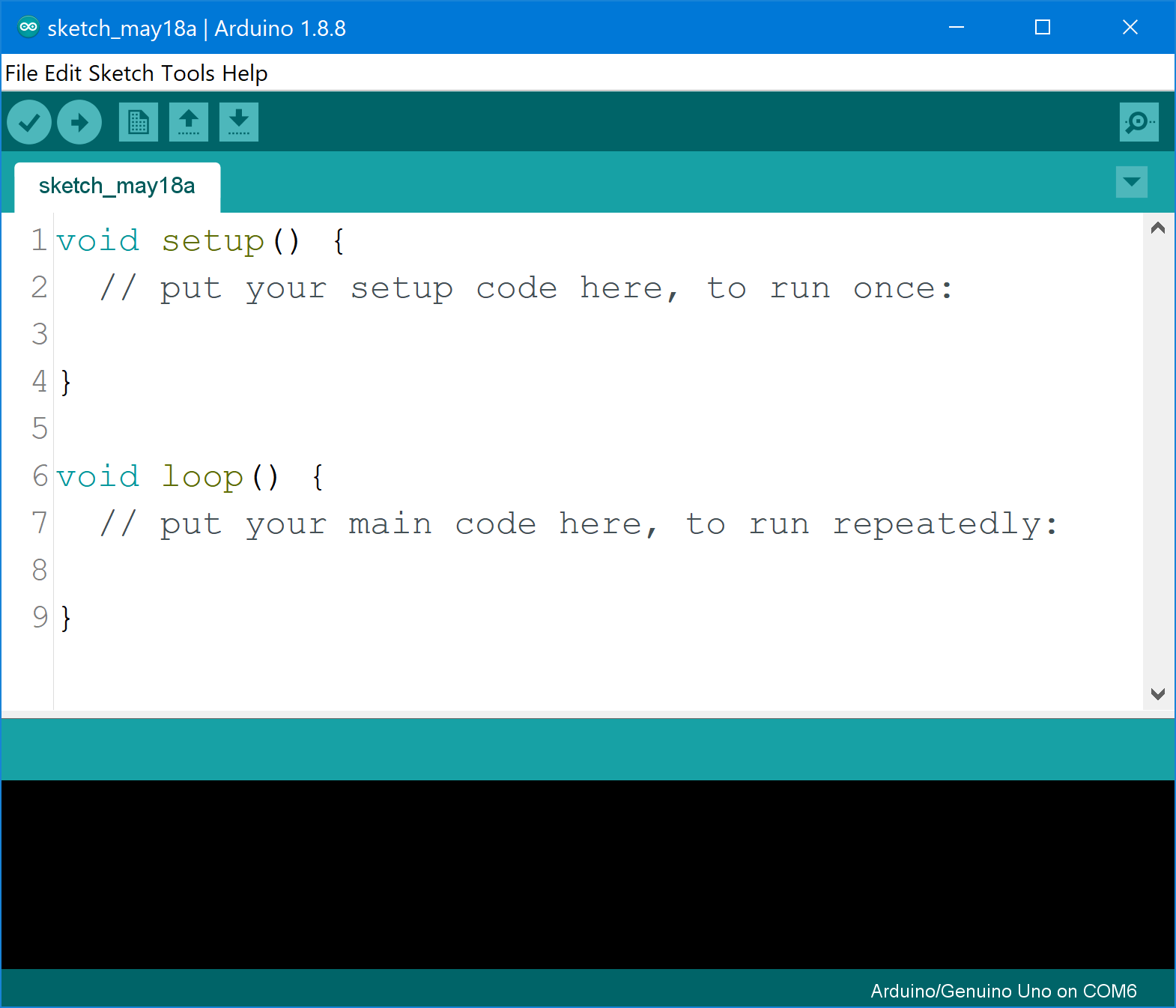
## Arduino IDE

**Arduino IDE** (Integrated Development Environment) је алат за развој који омогућава корисницима да пишу, компајлирају и учитавају код на **Arduino плоче**. Ово окружење пружа **интуитиван** и **приступачан интерфејс** за програмере, што олакшава креирање и тестирање програма за различите Arduino плоче.

Основне карактеристике Arduino IDE-а укључују:

* **Језик Програмирања**: Arduino IDE користи **C** и **C++** као основне програмске језике. Код написан у IDE-у се компајлира у машински код који микроконтролер може да изврши. Ово омогућава корисницима да креирају сложене и ефикасне програме за управљање хардвером.
* **Једноставан Интерфејс**: Интерфејс Arduino IDE-а је једноставан и лак за коришћење, с простором за писање кода, конзолом за приказ грешака и порука, и менијем за управљање пројектима.
* **Кампајлирање и Учитавање**: IDE пружа функције за **компајлирање кода** и учитавање компајлираног програма на Arduino плочу. Овај процес претвара написан код у машински код који микроконтролер може да изврши.
* **Подршка за више плоча**: Arduino IDE подржава велики број различитих Arduino плоча и хардверских платформи, као што су Arduino Uno, Arduino Mega, Arduino Nano, и многе друге. Корисници могу лако изабрати одговарајућу плочу из менија у IDE-у.
* **Библиотеке и Примери**: IDE укључује широк спектар **библиотека** и **примера кода** који помажу корисницима да започну са својим пројектима. Ове библиотеке олакшавају интеграцију различитих сензора и модула у пројекте.
* **Подршка за Серијску комуникацију**: Arduino IDE пружа функцију **серијске комуникације** која омогућава корисницима да шаљу и примају податке између рачунара и Arduino плоче. Ово је корисно за дебаговање и надгледање података у реалном времену.

Програми написани у Arduino IDE састоје се од две главне функције: **setup()** и **loop()** Функција setup() се извршава само једном при покретању и служи за иницијализацију пинова и конфигурацију. Функција loop() се непрекидно извршава и обавља све главне операције, као што су читање сензора и управљање актуаторима. Ова структура омогућава лако и ефикасно развијање хардверских пројеката. На слици 10 је приказан IDE са новим скечом и његове основне две функције.



Слика 10 Arduino IDE[[9]](#footnote-9)

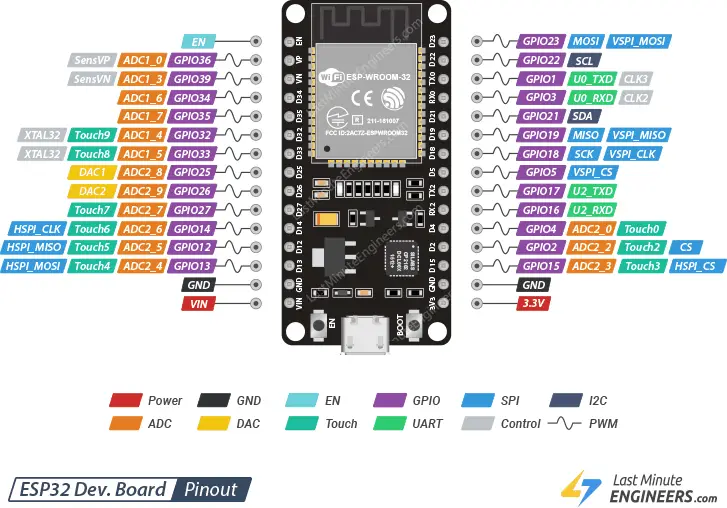
Arduino IDE је основни алат за развој и имплементацију пројеката на Arduino платформи, чинећи га важним делом за све кориснике Arduino хардвера.

## ESP модули

**ESP модули** представљају серију софистицираних **Wi-Fi** и **Bluetooth** чипова који су развијени од стране **Espressif Systems**. Ови модули су изузетно популарни у **IoT пројектима** због своје **ниске цене**, **велике перформансе** и **широких могућности повезивања**.

### ESP32

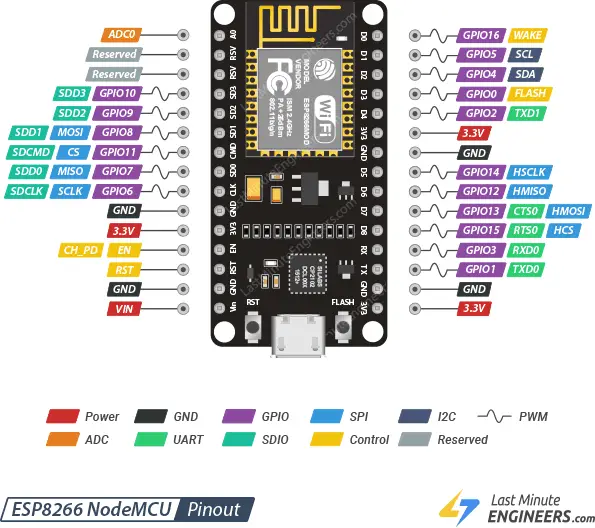
**ESP32** је напредни микроконтролер чип који подржава **Wi-Fi** и **Bluetooth** (Classic и BLE). Са **двојезграним процесором** и великим бројем **I/O пинова**, ESP32 пружа велику флексибилност у развоју пројеката. Његове карактеристике укључују интегрисани Wi-Fi модул, **Bluetooth комуникацију**, и могућност управљања различитим **сензорима** и **актуаторима**. ESP32 се користи у пројектима који захтевају брзу и поуздану комуникацију у реалном времену, као што су **паметни уређаји**, **системи за праћење** и **управљање**. „Pinout“ микроконтролера ESP32 је приказан на слици 11.



Слика 11 Приказ ESP32 заједно са његовим пиновима као и подржаним протоколима за сваки пин[[10]](#footnote-10)

### ESP8266

**ESP8266** је старији и једноставнији **Wi-Fi чип**, али и даље веома популаран због своје **приступачности** и **лаке интеграције** у пројекте. Са својим уграђеним Wi-Fi модулом, ESP8266 је идеалан за пројекте који захтевају основне Wi-Fi комуникације. Иако нема исте напредне функције као **ESP32**, ESP8266 је погодан за пројекте као што су **паметне кућне апликације**, **уређаји за праћење** и **једноставнији IoT системи**. „Pinout“ микроконтролера ESP8266 је приказан на слици 12.



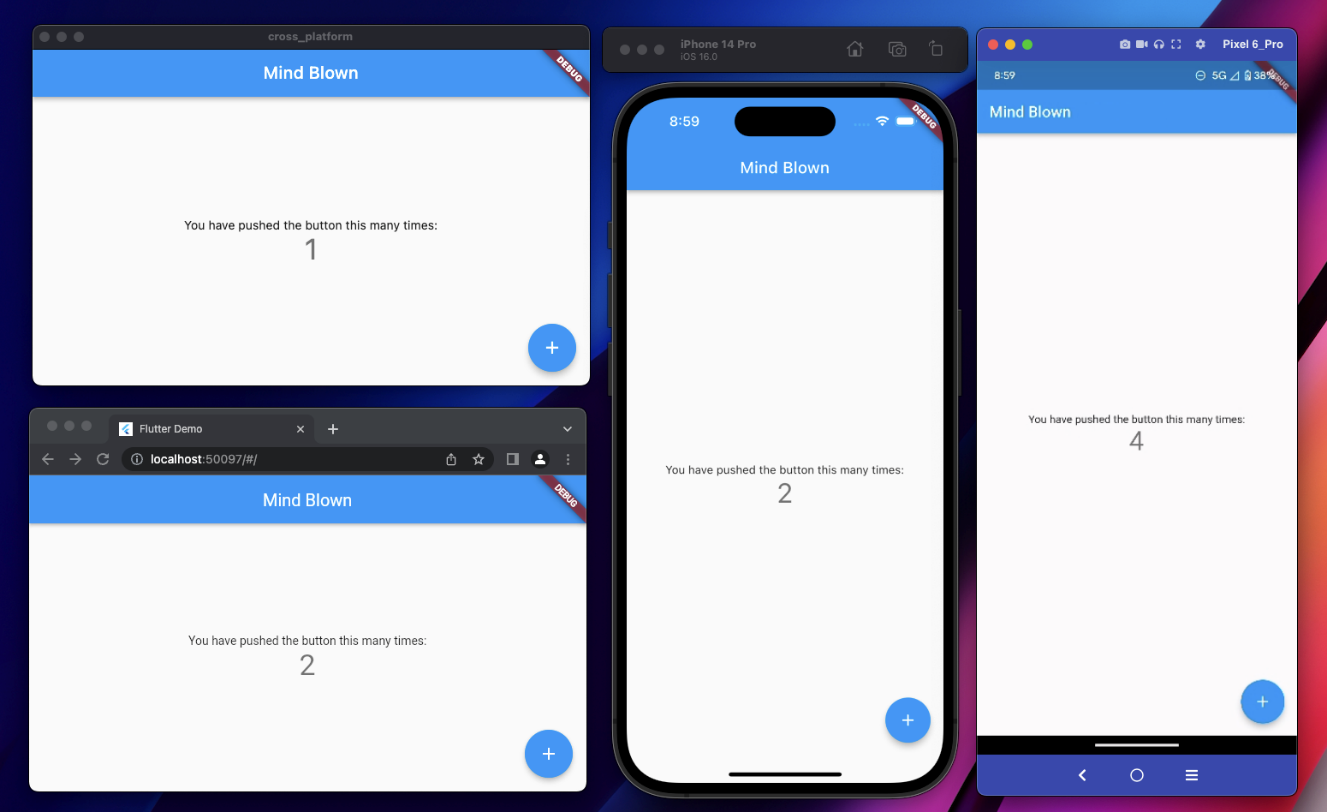
Слика 12 Приказ ESP8226 заједно са његовим пиновима као и подржаним протоколима за сваки пин[[11]](#footnote-11)

## Flutter

**Flutter** је радни оквир за развој корисничких интерфејса који омогућава креирање **међу-платформских** мобилних и веб апликација са једним кодом (слика 13). Развијен од стране Google-а, Flutter се истиче својом способношћу да пружи **високе перформансе** и **атрактивне корисничке интерфејсе** користећи један изворни код за iOS и Android платформе.

Један од главних аспеката Flutter-а је његова употреба **Dart програмског језика**, који је осмишљен да ради у савршеној синхронизацији са радним оквиром. Ово омогућава **брзо компајлирање** и **ефикасно извршавање**, што резултује високим перформансама и добрим корисничким искуством. Flutter нуди богат скуп **„widget-a“** који помажу у брзој изградњи визуелно привлачних и функционалних корисничких интерфејса. Такође, корисници могу лако прилагодити изглед и осећај апликације помоћу прилагођених widget-a и анимација.

Flutter подржава брзо развијање апликација захваљујући својој могућности за ажурирање кода у реалном времену, што је познато као **„Hot Reload“**. Ова функција омогућава програмерима да виде промене одмах, без потребе за поновним покретањем апликације. Ово значајно убрзава развојни процес и побољшава продуктивност. Уз широк спектар библиотека и алата, Flutter је одличан избор за развој савремених апликација које захтевају **ефикасност** и **високу визуелну привлачност**.



Слика 13 Flutter комајлован за iOS, Android i Web[[12]](#footnote-12)

## Docker

**Docker** је платформа за креирање, распоређивање и управљање контеризованим апликацијама. Омогућава програмерима да пакују апликације и све њихове зависности у један контејнер, који се може покренути било где, било да је реч о локалном развојном окружењу, серверу или у облаку.

Један од **кључних аспеката Docker-а** је његова способност да пружи **конзистентно окружење** за развој и продукцију. Користећи Docker, програмери могу бити сигурни да ће апликације радити на идентичан начин у свим окружењима, што значајно смањује проблеме са компатибилношћу и "works on my machine" проблеме. Docker користи **контејнере**, који су лаки и изоловани, чиме омогућавају различитим апликацијама да функционишу на истом систему без конфликата. Структура овог решења је приказана на слици 14.

Docker такође пружа моћне алате за **управљање и оркестрацију контејнера**, као што су **Docker Compose** и **Docker Swarm**. Docker Compose омогућава дефинисање и управљање мулти-контејнерским апликацијама помоћу једног конфигурационог фајла, док Docker Swarm пружа функционалности за распоређивање и управљање контернизованим апликацијама на више хостова (host).



Слика 14 Принцип рада Docker-a[[13]](#footnote-13)

Платформа је изузетно популарна због своје **флексибилности** и **лакоће коришћења**, што је чини идеалним решењем за модерне DevOps и **CI/CD (Continuous Integration/Continuous Deployment)** токове рада. Docker омогућава брзо развијање, тестирање и постављање апликација, доприносећи ефикасности и скалабилности развојног процеса.

Комбиновањем ових технологија, систем пружа свеобухватно решење за праћење и управљање еколошким уређајима, интегришући напредне функције и пружајући корисницима једноставан и ефикасан начин за управљање и праћење података у реалном времену.

# Мотивација

Развој пројекта у области паметних уређаја и IoT (Internet of Things) технологија одражава растућу глобалну тенденцију која се фокусира на побољшање квалитета живота кроз технологију и одрживост. Како се популарност паметних уређаја убрзано повећава, све више корисника трага за решењима која могу побољшати њихов свакодневни живот, а истовремено смањити негативан утицај на животну средину.

Повећана употреба паметних уређаја омогућава корисницима да лакше управљају разним аспектима својих домова и животне средине, као што су **енергетска ефикасност**, **управљање отпадом** и **мониторинг животне средине**. Ови уређаји омогућавају корисницима да прикупљају и анализирају податке, што им помаже да донесу информисане одлуке које могу довести до значајних уштеда у ресурсима и смањењу отпада.

Поред технолошког аспекта, пројекат је инспирисан и глобалним покретом за очување планете и одрживи развој. Кључна мотивација је у усклађивању са трендовима који подстичу еколошке иновације и развој система који подржавају одрживе праксе. Повећање еколошке свести и стварање решења која доприносе смањењу негативног утицаја на животну средину постаје све важније у контексту глобалних изазова.

Пројекат такође има за циљ да се одговори на потребу за једноставним и приступачним решењима која ће омогућити корисницима да интегришу паметне уређаје у својим домовима са минималним напором, што доприноси широј адаптацији и примени ових технологија. Све ове аспекте пројекат настоји да обухвати и унапреди, доприносећи напретку у технологији и одрживости.

# Спецификација

Ово поглавље је подељено у два основна дела: "Спецификација захтева" и "Спецификација система". Циљ овог поглавља је да прецизно дефинише све функционалне и не-функционалне захтеве, као и системске карактеристике које су потребне за реализацију пројекта. У овом делу рада неће бити укључене конкретне техничке имплементације, већ ће се уместо тога дати јасна слика о томе шта систем треба да обезбеди и како треба да буде структурисан.

Систем треба пројектовати са нагласком на **флексибилност** и **скалабилност**, како би се омогућило лако додавање нових типова паметних уређаја у екосистем. Пројекат треба да подржава интеграцију нових типова уређаја са минималним напором и без значајних измена у основном функционисању система. Овај приступ треба да обезбеди да се екосистем може лако проширивати и прилагођавати новим захтевима и технологијама, чиме се обезбеђује дугорочна одрживост и раст.

Процес додавања новог паметног уређаја у кориснички налог треба да буде дизајниран тако да буде што **једноставнији** и **интуитивнији**. Корисници треба да уложе минимални напор за интеграцију нових уређаја, а почетна конфигурација треба да буде прилагођена корисницима. Основно подешавање главног локалног сервера (хаба) у корисниковом дому може захтевати помоћ стручњака, али након тога, свако накнадно додавање уређаја корисник треба да буде у могућности да изврши самостално, без потребе за додатном помоћи. Ради демонстрације функционалности система, развиће се два типа паметних уређаја овог екосистема. Ова структура треба да омогући корисницима да лако управљају и прилагођавају систем према својим потребама, чиме се поједностављује његово проширење и прилагођавање.

Када корисник отвори апликацију, приказује се страница за пријаву. Након успешне пријаве, корисник долази до почетне странице на којој су приказани сви паметни уређаји овог екосистема. На овој страници, корисник може да види статус уређаја – да ли су на мрежи или недоступни, као и описе, типове и имена свих паметних уређаја.

На почетној страници се налази и **"плутајуће" дугме** које омогућава иницирање скенирања QR кода. Овим поступком нови паметни уређај се лако повезује са корисничким налогом. Када корисник кликне на одређени паметни уређај, отвара се страница са детаљима о том уређају. Овде се могу прегледати и изменити подаци о уређају, а информације о последњем прикупљеном податку су такође доступне.

Поред најскоријих информација, кориснику су приказани и **историјски подаци** прикупљени у претходном периоду. Одабир распона датума за преглед историјских информација врши се преко календарског поља, које омогућава избор жељених датума.

Са почетне странице, корисник има приступ страници **"Hub"**, где може да пошаље своје креденцијале на хаб како би добио права приступа за бележење података које паметни уређај шаље на удаљени сервер. На страници **"Device AP"**, корисник може да пошаље креденцијале своје интернет мреже паметном уређају како би могао да се повезује на интернет и шаље прикупљене податке до хаба. Ова акција захтева да корисник буде повезан на WiFi мрежу коју уређај емитује уколико не може да се повеже на другу мрежу.

Цела корисничка апликација треба да буде дизајнирана у складу са **„златним правилима“ корисничког интерфејса**, чиме се осигурава да корисник увек буде јасно информисан о свом тренутном стању у апликацији и о корацима које треба предузети за извршење жељених акција. Ова дизајнерска филозофија требало би да омогући интуитивно и прегледно корисничко искуство, смањујући могућност конфузије и омогућавајући корисницима да лако и брзо управљају свим функцијама апликације.

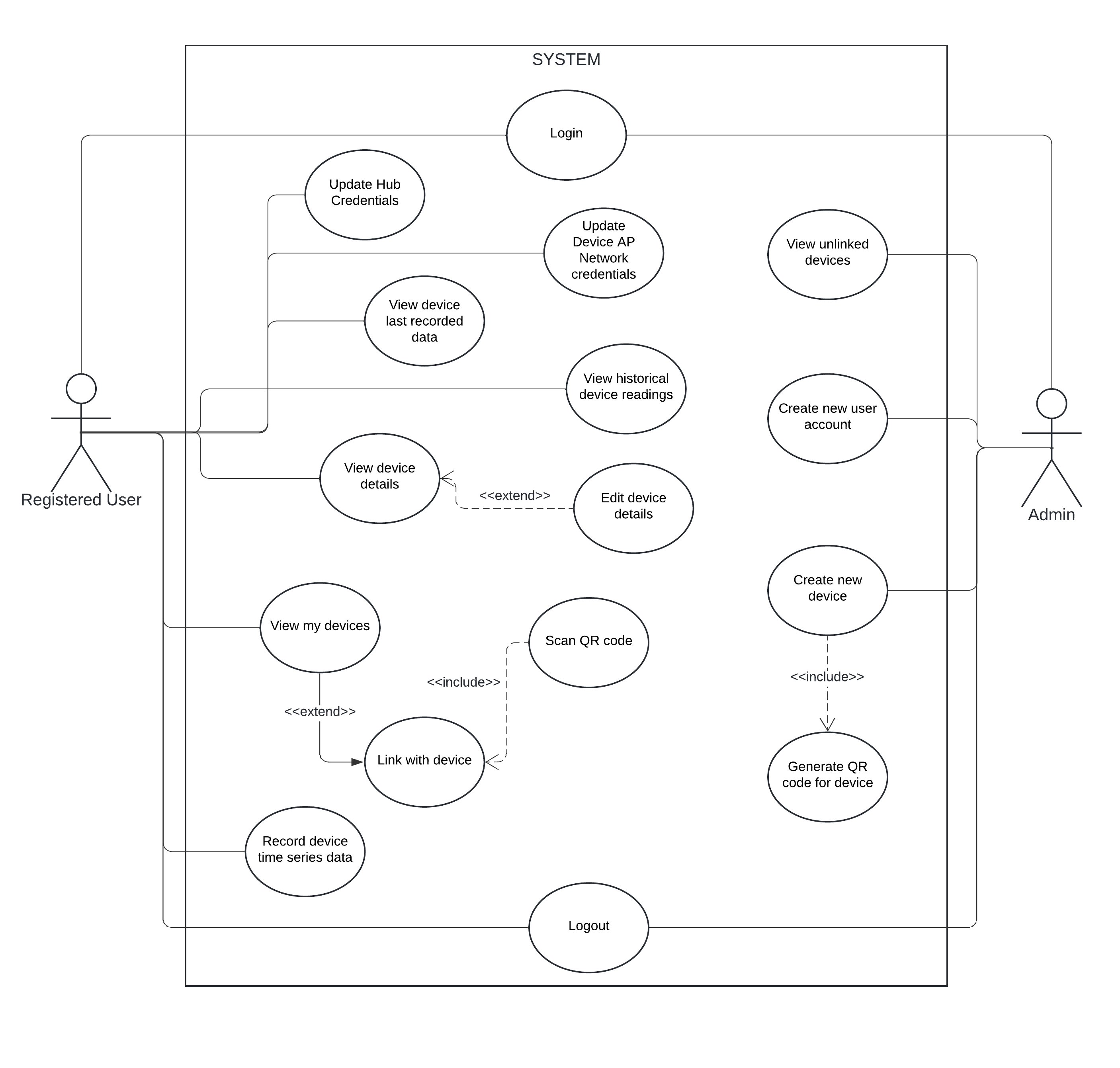
## Спецификација захтева

У овом делу детаљно су описани сви функционални и нефункционални захтеви које систем треба да испуњава. Основни акценат је на прецизирању шта је све потребно имплементирати, док се технички детаљи имплементације остављају за касније фазе. Функционални захтеви се фокусирају на специфичне функције и карактеристике које систем мора да обезбеди, док нефункционални захтеви укључују аспекте као што су перформансе, сигурност и употребљивост. Ова спецификација представља основу за јасно дефинисање циљева и очекивања пре почетка развоја и имплементације система.

### Функциконални захтеви

**Када је реч о овом систему**, могу се користити **дијаграми случајева коришћења** за представљање главних функционалности система. Ови дијаграми визуализују како корисници комуницирају са системом и које функционалности су доступне. Након дијаграма (приказаног на слици 15) , сваки случај коришћења се детаљно описује, укључујући **предуслове**, **кораке**, **резултате** и могуће **изузетке**.  
  
У главне случајеве коришћења спадају:

1. **Пријава на систем**
2. **Преглед паметних уређаја**
3. **Додавање новог уређаја**
4. **Преглед детаља уређаја**
5. **Преглед историјских података прикупљених од стране уређаја**
6. **Слање креденцијала на хаб**
7. **Слање WiFi креденцијала уређају**



Слика 15 Дијаграм случајева коришћења

1. **Пријава на систем**
   * **Предуслови**: Корисник има налог у систему и има приступ интернету.
   * **Кораци**:
     1. Корисник отвара апликацију.
     2. Корисник уноси мејл адресу име и лозинку.
     3. Систем проверава податке и пријављује корисника.
   * **Резултат**: Корисник је успешно пријављен и преусмерен на почетну страницу са прегледом својих паметних уређаја.
   * **Изузеци**:
     1. Ако корисничко име или лозинка нису тачни, систем приказује поруку о грешци и тражи поновни унос.
2. **Преглед паметних уређаја**
   * **Предуслови**: Корисник је пријављен у систем.
   * **Кораци**:
     1. Корисник приступа почетној страници апликације.
     2. Систем приказује листу свих паметних уређаја повезаних са корисничким налогом.
     3. Корисник може видети статус (на мрежи или недоступан) и основне информације о сваком уређају.
   * **Резултат**: Корисник види све своје паметне уређаје са њиховим статусом и основним информацијама.
   * **Изузеци**:
     1. Ако неки уређаји нису доступни, систем приказује поруку о недоступности.
3. **Додавање новог уређаја**
   * **Предуслови**: Корисник је пријављен у систем и има паметан уређај који поседује камеру.
   * **Кораци**:
     1. Корисник бира опцију за додавање новог уређаја.
     2. Корисник користи скенер QR кода за повезивање новог уређаја.
     3. Систем проверава податке и додаје уређај на корисников налог.
   * **Резултат**: Нови паметан уређај је успешно повезан и приказан на почетној страници.
   * **Изузеци**:
     1. Ако скенирање QR кода не успе, корисник може покушати поново или унети податке ручно.
4. **Преглед детаља уређаја**
   * **Предуслови**: Корисник је пријављен и има паметан уређај повезан са налогом.
   * **Кораци**:
     1. Корисник бира одређени уређај са листе.
     2. Систем приказује детаље о изабраном уређају.
     3. Корисник може изменити информације о уређају.
   * **Резултат**: Корисник има приступ детаљним информацијама и може изменити податке о уређају.
   * **Изузеци**:
     1. Ако подаци о уређају нису доступни, систем приказује поруку о грешци.
5. **Преглед историјских података прикупљених од стране уређаја**

* **Предуслови**: Корисник је пријављен и има паметан уређај повезан са налогом.
* **Кораци**:

1. Корисник бира одређени уређај са листе паметних уређаја.
2. Систем приказује детаље о изабраном уређају, укључујући најскорије прикупљене информације.
3. Испод најскоријих информација, корисник може приступити историјским подацима.
4. Корисник бира распон датума преко поља означеног календаром за који жели да види историјске податке.
5. Систем генерише и приказује графикон или табелу са историјским подацима за изабрани распон датума.

* **Резултат**: Корисник може прегледати историјске податке прикупљене од стране уређаја у формату који је визуелно лако разумљив и анализиран.
* **Изузеци**:
  + Ако нема доступних историјских података за изабрани распон датума, систем приказује поруку о недостатку података.
  + Ако се деси грешка у учитавању података, систем приказује поруку о грешци и омогућава кориснику да покуша поново.

1. **Слање креденцијала на хаб**
   * **Предуслови**: Корисник је пријављен и има паметан уређај повезан са налогом. Хаб је на истој мрежи као и корисников паметни уређај, и упаљен је.
   * **Кораци**:
     1. Корисник приступа страници за слање креденцијала на хаб.
     2. Корисник уноси потребне креденцијале.
     3. Систем шаље креденцијале хабу.
   * **Резултат**: Хаб добија потребне креденцијале за приступ удаљеном серверу како би трајно чувао прикупљене податке од паметних уређаја.
   * **Изузеци**:
     1. Ако слање креденцијала не успе, систем приказује поруку о грешци и омогућава поновни покушај.
2. **Слање WiFi креденцијала уређају**
   * **Предуслови**: Корисник је пријављен и има паметан уређај повезан са налогом. Уређај је упаљен и налази се у моду „WiFi access point-а“ у којем емитује WiFi мрежу на коју је корисник повезан.
   * **Кораци**:
     1. Корисник приступа страници за слање WiFi креденцијала.
     2. Корисник уноси податке за WiFi мрежу.
     3. Систем шаље WiFi креденцијале паметном уређају.
   * **Резултат**: Паметан уређај се повезује на WiFi мрежу и може слати податке.
   * **Изузеци**:
     1. Ако паметан уређај не може да се повежe на WiFi мрежу, систем приказује поруку о грешци и омогућава кориснику да покуша поново или провери мрежне поставке.

### Нефункционални захтеви

**Нефункционални захтеви** дефинишу квалитетне карактеристике система и играју кључну улогу у обезбеђивању да систем не само функционише како је предвиђено, већ и да пружи **задовољавајуће корисничко искуство** и одржава **високе стандарде перформанси**.

За систем који омогућава корисницима праћење и управљање паметним уређајима, следећи нефункционални захтеви су идентификовани:

1. **Перформансе система**: Систем треба да осигура брзе и поуздане одговоре на захтеве корисника, што подразумева минимално време одговора и високу доступност услуга. Потребно је обезбедити да се систем може носити са великом количином података и истовременим захтевима без значајног успоравања.
2. **Корисничко искуство (User Experience)**: Корисничко искуство треба да буде интуитивно и лако за употребу. Интерфејс треба да буде једноставан и јасан, што укључује лаку навигацију и разумљиве интеракције. Ово ће помоћи корисницима да брзо и ефикасно управљају својим паметним уређајима.
3. **Користнички интерфејс**: Интерфејс треба да буде визуелно привлачан и лак за коришћење. Употреба стандарда за дизајн и инспирација из најбољих пракси у дизајну интерфејса може помоћи у стварању пријатног корисничког искуства.
4. **Респонзивни дизајн**: Front-end апликација треба да буде респонзивна, што значи да ће се адекватно прилагодити различитим величинама екрана и уређајима. Ово је посебно важно за омогућавање коришћења апликације на различитим уређајима као што су паметни телефони, таблети и рачунари.
5. **Вишејезичност (Локализација)**: Систем би требало да подржава више језика, омогућавајући локализацију за различите регије и кориснике. Ово ће помоћи у осигурању да апликација буде доступна ширем кругу корисника и прилагођена њиховим потребама.
6. **Комппатибилност са форматима и стандардима**: Систем треба да буде компатибилан са стандардним форматима података као што су JSON и XML, као и са индустријским стандардима који су релевантни за размену и управљање подацима.
7. **Преносивост и портабилност**: Систем треба да буде преносив између различитих платформи, што подразумева да се може користити на различитим оперативним системима и уређајима без потребе за значајним променама.
8. **Отпорност на грешке и опоравак**: Систем треба да буде отпоран на грешке, што укључује адекватне механизме за откривање и исправљање грешака. Такође, потребно је увести стратегије за опоравак од грешака како би се осигурало да систем настави са радом у случају непредвиђених проблема.

Ови нефункционални захтеви ће осигурати да систем буде ефикасан, лако коришћен и одговарајући потребама корисника, што ће допринети успешности и прихватљивости апликације у различитим окружењима.

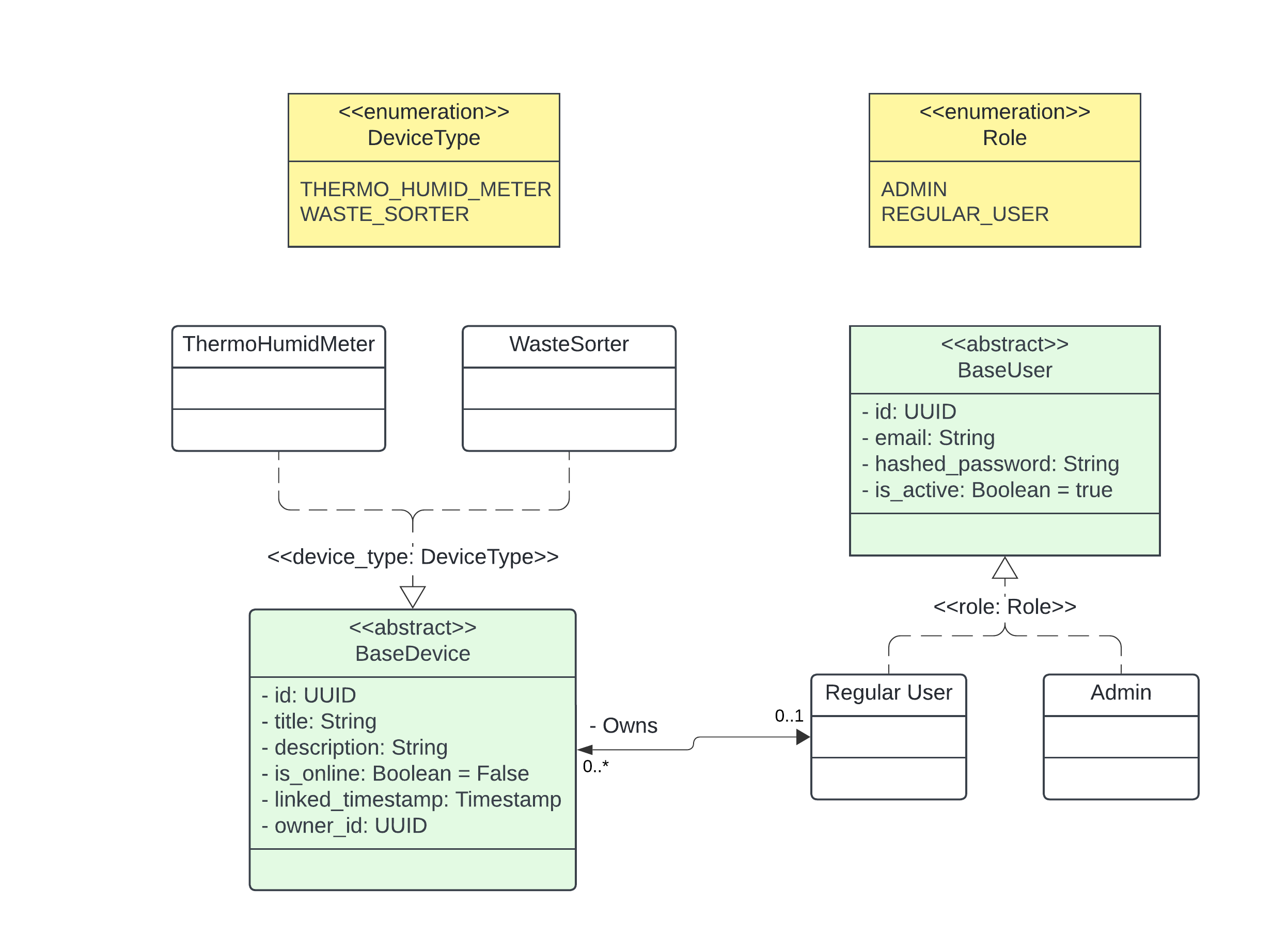
## Спецификација система

Спецификација система детаљно описује све компоненте и архитектуру система који ће бити развијен. Ово укључује опис хардверских и софтверских делова, њихових функција, као и начине на које су међусобно повезани и интегрисани. Овај део рада разматра конкретне елементе система, као што су сервери, базе података, комуникациони протоколи и кориснички интерфејси, и дефинише како ће они допринети укупном функционисању система. Спецификација служи као основа за техничку реализацију и осигурава да све компоненте раде у складу са претходно постављеним захтевима и очекивањима.

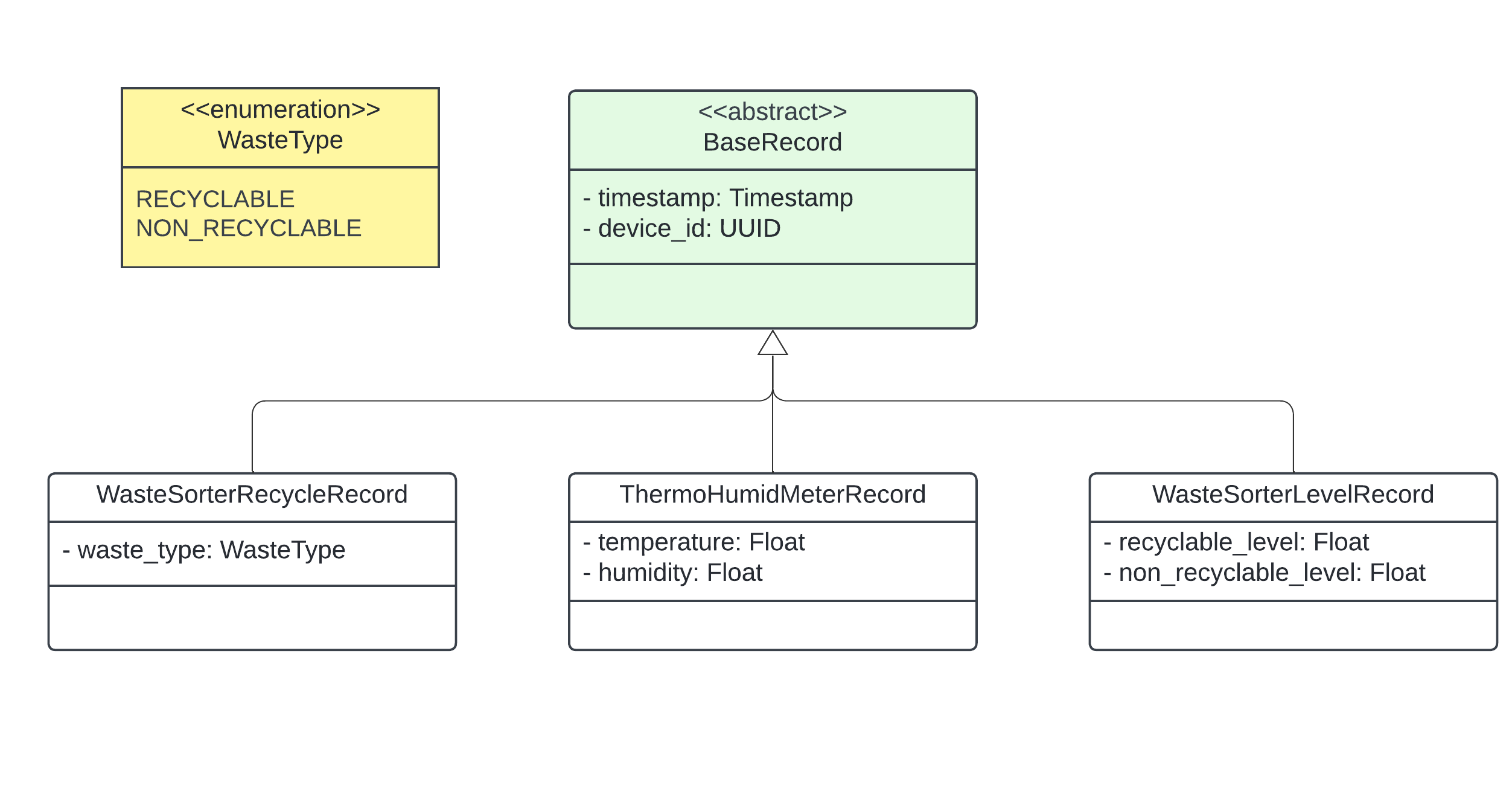
За креирање ових дијаграма коришћен је алат LucidChart[[14]](#footnote-14).

### Модел података

У овом одељку се представља модел података система. Дијаграм класа пружа структурални приказ модела у систему и односа између њих. Они визуализују главне класе, атрибуте и релације. Постоје два дијаграма класа: један за „стабилне“ моделе и други за моделе који чувају податке прикупљене у реалном времену од стране уређаја. Класе у систему се мапирају један на један са табелама у бази података, што значи да није направљен посебан дијаграм шеме базе података. Дијаграми класа, приказани на слици 16 и слици 17, показују како су модели организовани и како су различите класе повезане.



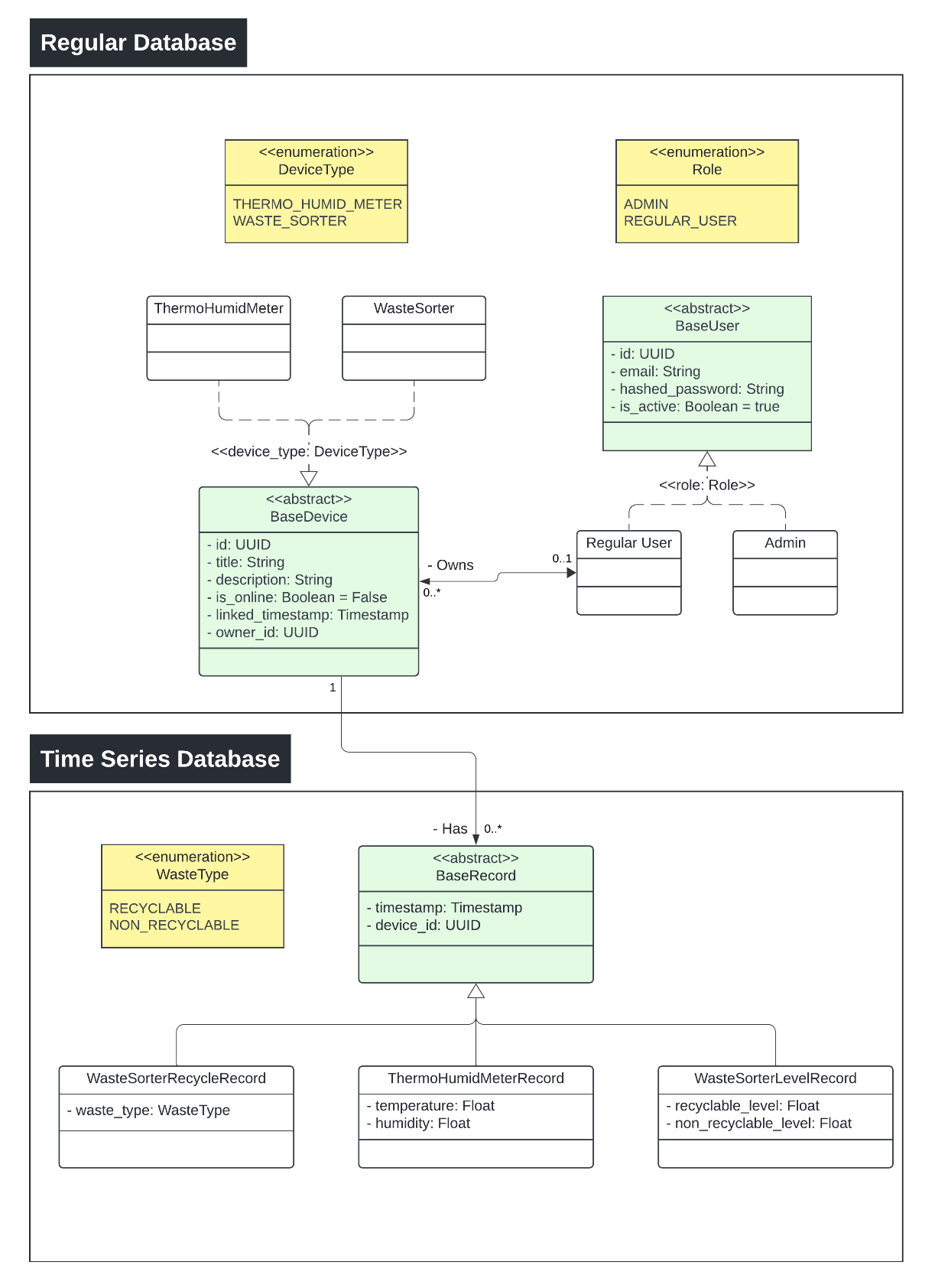
Слика 16 Класни дијаграм за руковање са „стабилним“ моделима



Слика 17 Класни дијаграм за руковање моделима за податке припљене у реалном времену

Систем је осмишљен тако да се један класни дијаграм мапира на једну базу података која се бави **„стабилним“ подацима**, док се други мапира на базу података која обрађује **податке прикупљене у реалном времену**. Ова архитектура омогућава ефективно управљање различитим типовима података и осигурава да су оба типа података адекватно организована и доступна у складу са својим потребама.

На следећој слици (слика 18) је приказан однос између ова два типа модела, тј. веза између модела који се су намењени за чување у различитим базама података.

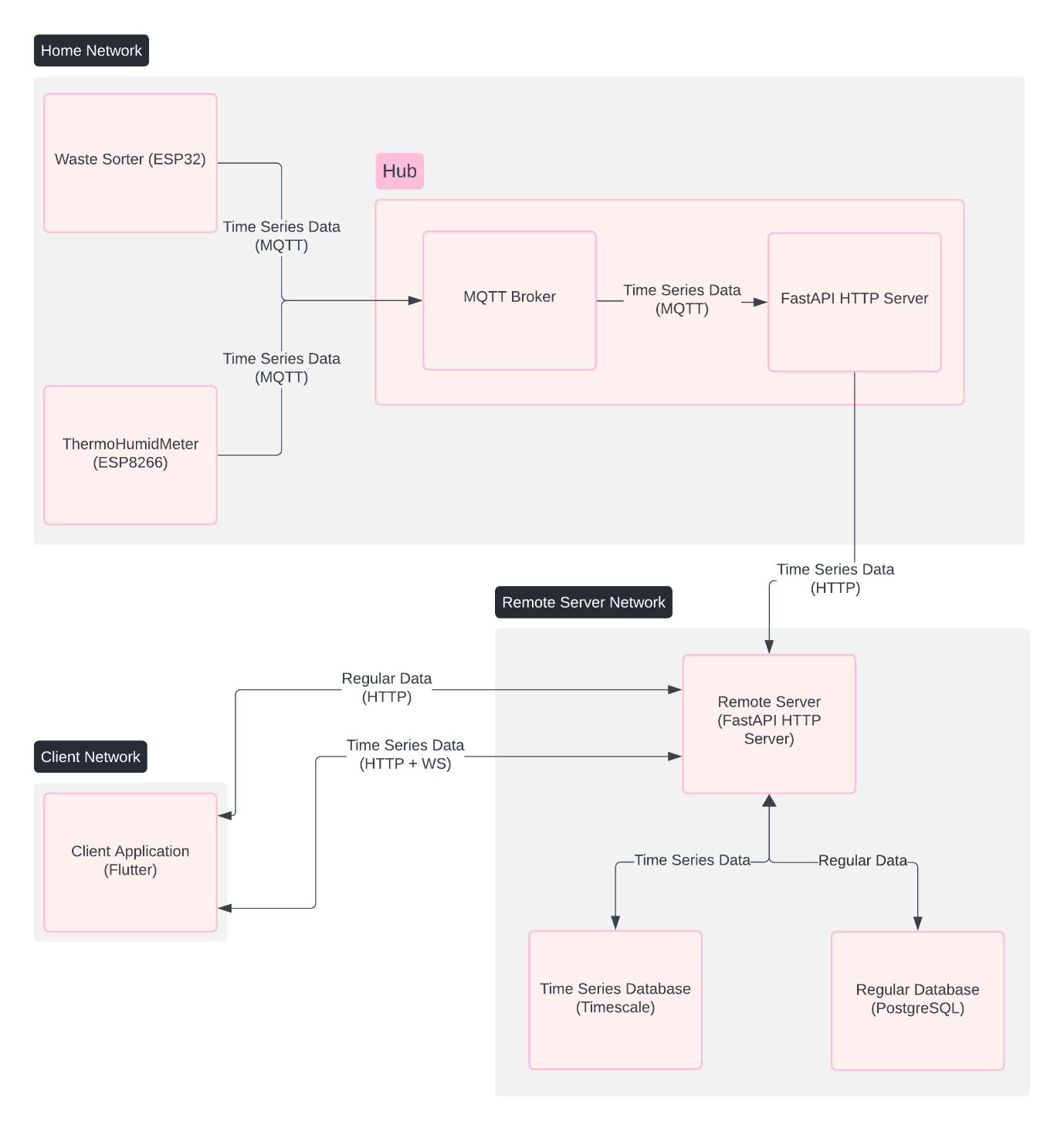


Слика 18 Однос модела између два класна дијаграма

Дијаграм класа помаже у разумевању организације података и њихових веза, што је критично за правилну имплементацију и управљање подацима у систему.

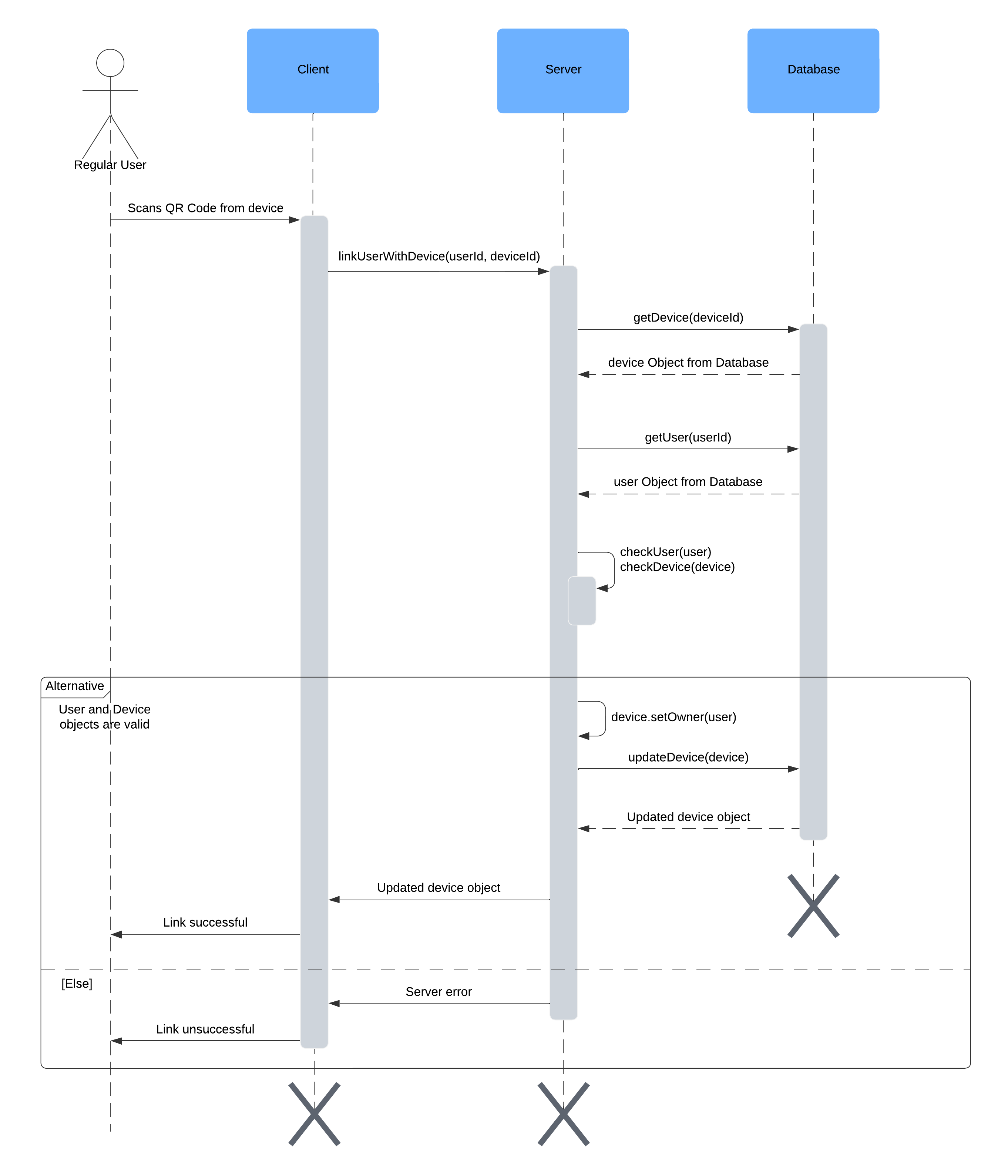
### Архитектура система

Архитектура система се представља кроз дијаграм компоненти. Овај дијаграм, приказан на слици 19, илуструје структуру система, укључујући различите компоненте и њихове интеракције. Дијаграм компоненти пружа преглед главних функционалних јединица система и њихових зависности, што је кључно за разумевање како различити делови система сарађују и комуницирају.

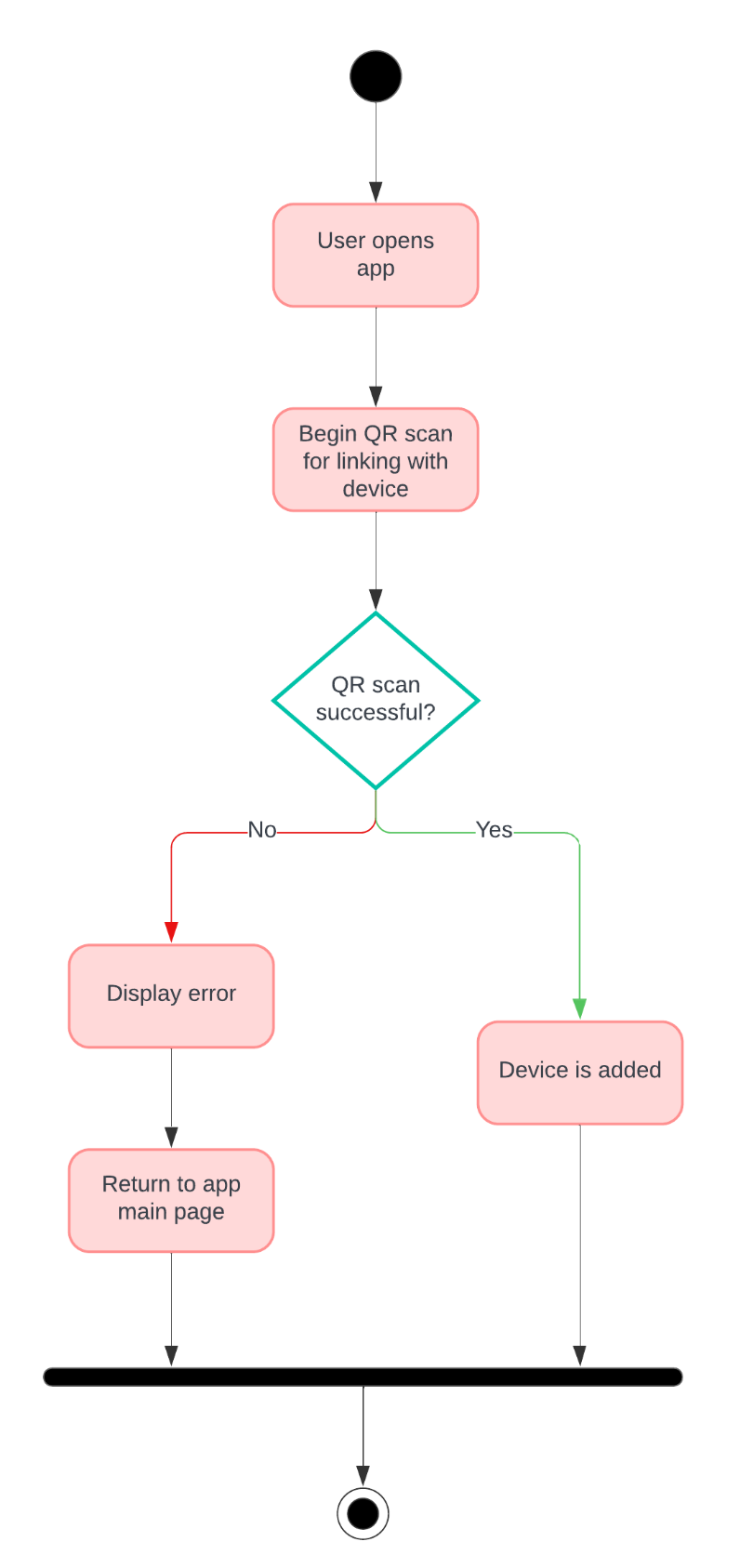


Слика 19 Дијаграм компоненти система

Поред тога, изабрана је једна од битнијих функционалности система, повезивање уређаја са корисничким налогом. Ова функционалност је илустрована на дијаграму секвенци и дијаграму активности. Дијаграм секвенци, приказан на слици 20, илуструје редослед акција и комуникације између различитих компоненти система у сценарију повезивања уређаја са корисничким налогом. Дијаграм активности, који приказан на слици 21, представља ток рада и поступке који се изводе током овог процеса.



Слика 20 Дијаграм секвенце који илуструје комуникацију између компоненти током повезивања уређаја са корисничким налогом



Слика 21 Дијаграм активности који илуструје процес повезивања уређаја са корисничким налогом

Ови дијаграми пружају свеобухватан преглед архитектуре система и његових компоненти, омогућавајући детаљно разумевање структуре и динамике система.

# Имплементација система

**У овом поглављу је представљена имплементација система за управљање паметним уређајима у оквиру пројекта Waste-Free Home. Овај систем обухвата интеграцију два кључна типа уређаја: Thermo Humid Meter и Waste Sorter.**

**Поглавље служи за детаљан опис свих битних елемената имплементације софтверског система, укључујући и технологије наведене у претходном поглављу. Описаће се како су различите технологије примењене у пројекту, са фокусом на конкретне аспекте програмске реализације и техничке изазове током развоја.**

**У наставку ће бити објашњени сви важни аспекти система, укључујући структуру кода, коришћене библиотеке, и решења за превазилажење програмерских проблема. За илустрацију ће бити укључени листинзи фрагмената програмског кода који демонстрирају конкретне функције и методе.**

## Паметни уређаји

**У оквиру развоја система, примењена су два типа паметних уређаја: Thermo Humid Meter и Waste Sorter.** Ови уређаји су кључне компоненте IoT система и сваки од њих игра значајну улогу у прикупљању и обради података.

**Оба уређаја користе различите библиотеке за подршку функционалности својих сензора и актуатора.** Поред библиотека специфичних за компоненте уређаја, као што су Adafruit\_PN532 и Adafruit\_SSD1306 за NFC/RFID и OLED дисплеј, оба уређаја користе и заједничке библиотеке које омогућавају компатибилност и управљање сензорима и комуникацијом.

Списак коришћених библиотека укључује:

* **Adafruit\_BusIO[[15]](#footnote-15)**
* **Adafruit\_GFX\_Library[[16]](#footnote-16)**
* **Adafruit\_SSD1306[[17]](#footnote-17)**
* **Adafruit\_Unified\_Sensor[[18]](#footnote-18)**
* **DHT\_sensor\_library[[19]](#footnote-19)**
* **ESP32Servo[[20]](#footnote-20)**
* **HCSR04[[21]](#footnote-21)**
* **NDEF[[22]](#footnote-22)**
* **PN532[[23]](#footnote-23)**
* **PN532\_I2C[[24]](#footnote-24)**
* **PubSubClient[[25]](#footnote-25)**

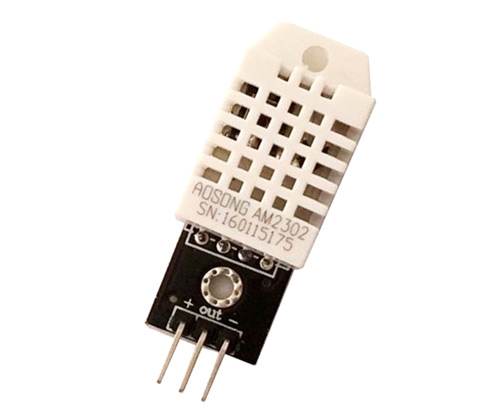
Интеграција библиотека са компонентама оба уређаја омогућава ефикасно прикупљање података и комуникацију са **локалним сервером и MQTT брокером**, што је основа за развој и функционисање целокупног IoT система.

Такође, један од кључних аспеката имплементације оба уређаја био је минимизирање **потрошње електричне енергије.** С обзиром на то да оба уређаја функционишу у контексту IoT система, где је важно осигурати дуготрајну и ефикасну употребу батерија, пажња је посвећена оптимизацији потрошње енергије. Ово укључује употребу **енергетски ефикасних** **компоненти**, као и примену **техника уштеде енергије у програмском коду**. Таква стратегија омогућава одржавање функције уређаја са минималним утицајем на енергетске ресурсе, чиме се побољшава одрживост и ефикасност целокупног система.

Приликом свог **укључивања,** оба уређаја извршавају провере у својој **меморији** ради утврђивања доступности сачуваних **креденцијала за интернет мрежу.** Уколико креденцијали нису доступни или су нетачни, уређаји прелазе у режим Wi-Fi access point-а, у којем емитују Wi-Fi мрежу. Корисник се повезује на ову мрежу и помоћу апликације, која ће бити касније описана, шаље исправне креденцијале интернет мреже на којој се налази локални сервер. Након примања креденцијала, уређај се рестартује и покушава поново да се повеже на мрежу. Уколико повезивање буде успешно, уређај прелази у свој нормалан режим рада, прикупља податке и шаље их на хаб. Овај процес осигурава да уређаји буду правилно конфигурисани за комуникацију у оквиру система.

### Thermo Humid Meter

**Thermo Humid Meter, представља паметни термометар и хидрометар који користи DHT22 (слика 23) сензор за мерење температуре и влажности ваздуха.** Вредности добијене овим сензором преносе се на **локални сервер**, где се даље обрађују и анализирају. Thermo Humid Meter је имплементиран на **ESP8266** (слика 22) микроконтролеру, који укључује интегрисани **Wi-Fi** чип, што омогућава повезивање са Wi-Fi мрежом у кући.



Слика 22 ESP8266 микроконтролер[[26]](#footnote-26) Слика 23 DHT22 сензор[[27]](#footnote-27)

На следећем листингу (листинг 5) су приказане функције које користи овај уређај.

void handleSensorReading();

void checkAndReconnectWiFi();

void checkAndReconnectMQTT();

void discoverMDNSService();

void startAccessPoint();

void stopAccessPoint();

void handleWiFiCredentialsUpdate();

void writeStringToEEPROM(int startAddress, String data);

String readStringFromEEPROM(int startAddress);

void writeIntToEEPROM(int address, int value);

int readIntFromEEPROM(int address);

Листинг 5 Функције које користи паметни уређај Thermo Humid Meter

Појашњење сваке од ових функција следи у наставку:

* void handleSensorReading();  
  Ова функција је одговорна за читање података са DHT22 сензора. Функција узима тренутне вредности температуре и влажности ваздуха, и те податке прослеђује даље на обраду. Ова функција се обично позива у одређеним интервалима како би се осигурало да уређај редовно прикупља ажуриране податке.
* void checkAndReconnectWiFi();  
  Ова функција проверава статус Wi-Fi везе и, уколико је веза прекинута, покушава да се поново повежe на мрежу користећи претходно сачуване креденцијале. Ово је важно за континуитет рада уређаја, јер без интернет везе уређај не може слати податке на локални сервер.
* void checkAndReconnectMQTT();  
  Слично као и функција за Wi-Fi, ова функција проверава статус MQTT конекције и покушава поново да се повеже уколико је веза прекинута. MQTT протокол се користи за слање података од уређаја до хаба, тако да је ова функција кључна за непрекинуту комуникацију.
* void discoverMDNSService();  
  Ова функција користи mDNS протокол да пронађе локални сервер (хаб) на мрежи. mDNS омогућава уређају да открије и идентификује друге уређаје унутар исте локалне мреже без потребе за статичком IP конфигурацијом. Успешно откривање хаба је неопходно да би уређај знао где да шаље прикупљене податке.
* void startAccessPoint();  
  Када уређај не може да се повеже на Wi-Fi мрежу, ова функција активира Wi-Fi access point мод, омогућавајући кориснику да се повежe на уређај и унесе нове мрежне креденцијале. Овај мод се обично користи током почетне конфигурације уређаја или када је потребно ажурирати постојеће креденцијале.
* void stopAccessPoint();  
  Ова функција деактивира Wi-Fi access point мод и враћа уређај у нормалан режим рада. Ово се ради након што су креденцијали за Wi-Fi мрежу успешно унесени и уређај се повеже на мрежу.
* void handleWiFiCredentialsUpdate();  
  Ова функција се користи за обраду и чување нових Wi-Fi креденцијала које корисник уноси када је уређај у Wi-Fi access point моду. Функција прима креденцијале и чува их у меморији уређаја (у EEPROM-у)
* void writeStringToEEPROM(int startAddress, String data);  
  Ова функција записује текстуалне податке (String) у EEPROM меморију на одређену адресу. EEPROM је тип трајне меморије, што значи да подаци остају сачувани чак и када је уређај искључен.
* String readStringFromEEPROM(int startAddress);  
  Ова функција чита текстуалне податке (String) из EEPROM меморије са одређене адресе.
* void writeIntToEEPROM(int address, int value);  
  Ова функција записује целобројне податке (int) у EEPROM меморију на одређену адресу.
* int readIntFromEEPROM(int address);  
  Ова функција чита целобројне податке (int) из EEPROM меморије са одређене адресе.

На листингу 6 се налази setup() функција овог уређаја. Функција setup() у овом уређају служи за иницијализацију основних компоненти и постављање почетних услова за рад уређаја. Прво, серијска комуникација се покреће на брзини преноса података од 9600, што омогућава дебаговање и праћење рада уређаја путем серијског монитора. Затим се иницијализује EEPROM меморија са капацитетом од 512 бајтова, што омогућава читање и писање података који се могу сачувати и након рестартовања уређаја. Након тога, иницијализује се DHT22 сензор за мерење температуре и влажности ваздуха. Уређај затим покушава да се повеже на WiFi мрежу путем функције checkAndReconnectWiFi(). Уколико је успостављена успешна WiFi веза, покреће се mDNS сервис који омогућава уређају да се лакше открије на мрежи под именом "ThermoHumidMeter". У случају да WiFi веза није успостављена, уређај то пријављује путем серијског монитора и прелази у WiFi Access Point режим, покрећући функцију startAccessPoint(), што омогућава кориснику да уноси нове креденцијале за приступ мрежи.

void setup() {

  Serial.begin(9600);

  EEPROM.begin(512);

  // Init DHT22 sensor

  dht\_sensor.begin();

  // Try connecting to WiFi

  checkAndReconnectWiFi();

  if (WiFi.status() == WL\_CONNECTED) {

    if (!MDNS.begin("ThermoHumidMeter")) {

      Serial.println("Error starting mDNS");

    }

  } else {

    // Handle the case where WiFi connection failed or is not available

    Serial.println("WiFi not connected. mDNS cannot be initialized. Starting Access Point...");

    startAccessPoint();

  }

}

Листинг 6 setup() функција уређаја Thermo Humid Meter

Функција loop() (листинг 7) је срце сваког Arduino кода и извршава се непрекидно током рада уређаја. У овој имплементацији, loop() функција обавља неколико важних задатака у зависности од тога да ли је уређај у WiFi Access Point режиму или у нормалном режиму рада.

Уколико је уређај у **AP (Access Point) режиму**, прво се проверава стање променљиве apMode. Ако је вредност apMode постављена на true, уређај се налази у AP режиму и обрађује HTTP захтеве корисника путем функције server.handleClient(). Ово омогућава кориснику да уноси мрежне креденцијале преко веб интерфејса.

Ако уређај није у AP режиму, односно ако apMode има вредност false, функција loop() врши низ операција повезаних са нормалним радом уређаја:

1. **Провера и поновно повезивање на WiFi и MQTT**: Функције checkAndReconnectWifi() и checkAndReconnectMQTT() служе за одржавање стабилне везе са WiFi мрежом и MQTT брокером.
2. **Проналажење MQTT сервиса помоћу mDNS**: Уколико је WiFi веза успостављена, али није могуће повезати се на MQTT брокер, уређај покушава да пронађе брокера помоћу mDNS сервиса кроз функцију discoverMDNSService().
3. **MQTT loop**: Функција client.loop() обрађује све долазне поруке са MQTT брокера и одржава везу активном.
4. **Очитавање сензора**: Када су све комуникације са мрежом успостављене, функција handleSensorReading() прикупља податке са сензора и шаље их на хаб.

На крају сваке итерације loop() функције, уређај прави паузу од 60 секунди пре него што поново прође кроз исти циклус, чиме се обезбеђује континуиран рад уређаја и периодично освежавање података.

void loop() {

  if (apMode) {

    server.handleClient(); // Handle HTTP requests in AP mode

  } else {

    checkAndReconnectWiFi();

    checkAndReconnectMQTT();

    // If not connected to MQTT Broker , search Broker service with mDNS

    if(WiFi.status() == WL\_CONNECTED && !client.connected()){

      Serial.println("Discovering services...");

      discoverMDNSService();

    }

    // MQTT loop

    client.loop();

    handleSensorReading();

  }

  delay(60000);

}

Листинг 7 loop() функција уређаја Thermo Humid Meter

Функција handleSensorReading() (листинг 8) је задужена за очитавање података са DHT22 сензора, који мери температуру и влажност, као и за објављивање тих података на MQTT брокеру.

1. **Очитавање података са сензора**: На почетку функције, очитавају се вредности температуре и влажности користећи методе readTemperature() и readHumidity() из библиотеке за DHT22 сензор. Вредности које су очитане чувају се у променљивама temperature и humidity.
2. **Провера валидности података**: Након очитавања, проверава се да ли су добијене вредности валидне. Уколико је било који од података NaN (Not a Number), функција закључује да је читање са сензора неуспешно и исписује поруку грешке на серијски монитор
3. **Формирање и објављивање поруке**: Уколико су подаци валидни, формира се MQTT порука у JSON формату која садржи кључеве temperature и humidity, са одговарајућим вредностима. Прво се креира топик за уређај користећи променљиву device\_id, која се комбинује са предефинисаним префиксом и суфиксом да би се добио комплетан MQTT топик (device\_topic). Потом се креира порука thermo\_humid\_message, која садржи вредности температуре и влажности у JSON формату.
4. **Слање поруке**: Коначно, порука се објављује на MQTT брокеру помоћу функције client.publish(), користећи раније формирани топик и поруку. Након тога, порука се исписује на серијски монитор за потребе праћења и дебаговања.

void handleSensorReading(){

    // Read from sensor and publish data

    float temperature = dht\_sensor.readTemperature();

    float humidity = dht\_sensor.readHumidity();

    if (isnan(temperature) || isnan(humidity)) {

      Serial.println("Failed to read from DHT sensor!");

    } else {

      String device\_topic = String(MQTT\_DEVICE\_TOPIC\_PREFIX) + String(device\_id) + String(MQTT\_RECORD\_TOPIC\_SUFIX);

      String thermo\_humid\_message = "{\"temperature\":\"" + String(temperature) + "\", \"humidity\":\"" + String(humidity) + "\"}";

      client.publish(device\_topic.c\_str(), thermo\_humid\_message.c\_str());

      Serial.print("Thermo Humid Meter message: ");

      Serial.println(thermo\_humid\_message.c\_str());

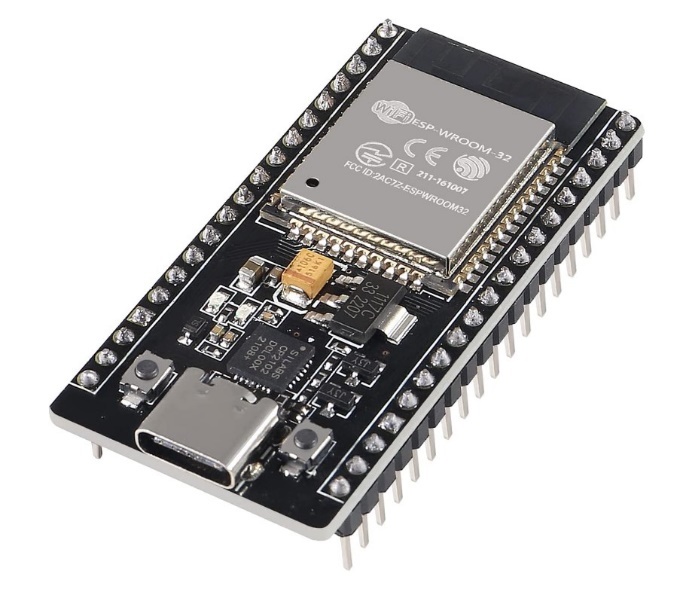
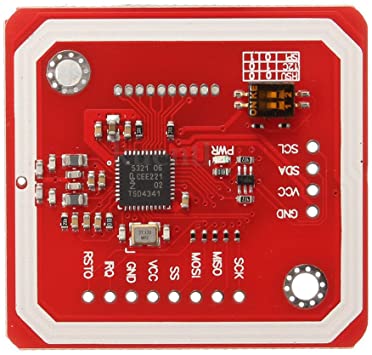
    }

}

Листинг 8 handleSensorReading() функција уређаја Thermo Humid Meter

### Waste Sorter

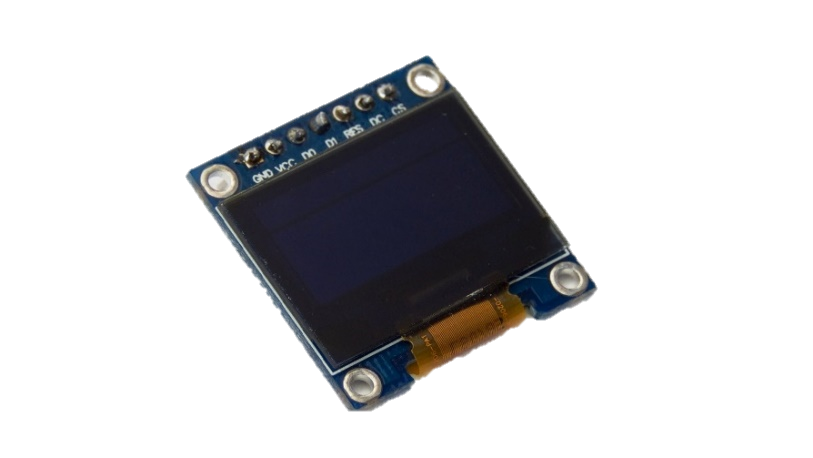
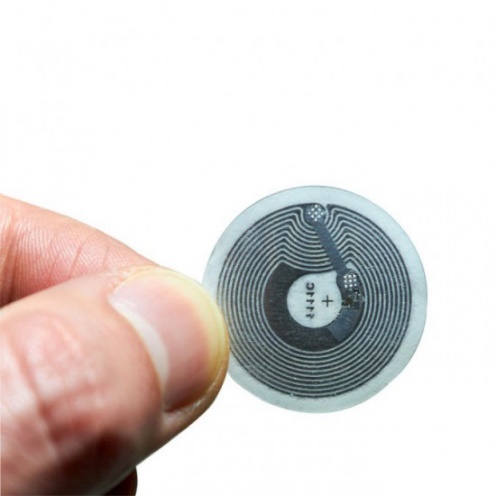
**Waste Sorter, представља паметну канту дизајнирану да разликује отпад који је погодан за рециклажу од отпада који није.** Waste Sorter је опремљен **NFC/RFID** сензором **PN532** (слика 25). Када предмет са **RFID** налепницом (слика 29) буде приложен на читач, сензор очитава налепницу и одређује у коју преграду отпад треба да буде стављен. Канта има две преграде: једну за рециклабилан отпад и другу за нерциклабилан. На основу одлуке коју донесе уређај, актуатори у облику серво мотора (**mg90s**, слика 27) отварају одговарајућу преграду. Поред тога, Waste Sorter користи **HC-SR04** (слика 26) ултразвучне сензоре за одређивање напуњености обе преграде, што омогућава ефикасно управљање отпадом. Waste Sorter је опремљен и дисплејем (**SSD1306**, слика 28) који приказује визуелне информације о стању преграда. На дисплеју се исцртавају правоугаоници који показују колики део сваке преграде је попуњен у односу на капацитет. Ова визуелизација омогућава кориснику да лако прати ниво напуњености преграда и на основу тога донесе одговарајуће одлуке о изношењу отпада. Овај уређај је имплементиран на **ESP32** (слика 24) микроконтролеру, који такође садржи интегрисани Wi-Fi чип, што омогућава повезивање на исту Wi-Fi мрежу као и Thermo Humid Meter.

Слика 24 ESP32 микроконтролер[[28]](#footnote-28) Слика 25 PN532 NFC/RFID чип[[29]](#footnote-29)

Слика 26 HC-SR04 сензор [[30]](#footnote-30) Слика 27 mg90s микро серво[[31]](#footnote-31)

Слика 28 SSD1306 OLED екран[[32]](#footnote-32) Слика 29 Wet Inlay 213 RFID налепница[[33]](#footnote-33)

Функција setup() (листинг 9) за уређај **Waste Sorter** поставља основну конфигурацију уређаја и омогућава почетно повезивање са мрежом, слично као код уређаја **Thermo Humid Meter**. На почетку, иницијализују се потребне хардверске компоненте, укључујући I2C комуникацију (Wire.begin()), серијску комуникацију (Serial.begin(9600)), NFC модул (nfc.begin()), серво моторе за управљање преградама за отпад, и SSD1306 дисплеј за приказ информација.

* **Инцијализација серво мотора**: Серво мотори који управљају преградама за рециклабилни и нерециклабилни отпад су иницијализовани и постављени на почетни положај (80 степени).
* **Иницијализација дисплеја**: SSD1306 дисплеј се иницијализује и проверава се његова функционалност. Уколико је иницијализација успешна, дисплеј се чисти и поставља се текст који ће бити приказан.
* **Повезивање на Wi-Fi мрежу**: Након иницијализације хардвера, уређај покушава да се повеже на Wi-Fi мрежу. Уколико успе да се повеже, покушава се иницијализација mDNS сервиса за лакше откривање уређаја на мрежи. Ако не успе да се повеже на мрежу, уређај прелази у Access Point мод, омогућавајући кориснику да конфигурише Wi-Fi креденцијале преко мобилне апликације.

void setup() {

  Wire.begin();

  Serial.begin(9600);

  // Initialize NFC

  nfc.begin();

  // Initialize Servos

  recycableServo.attach(RECYCABLE\_SERVO\_PIN);

  nonrecycableServo.attach(NON\_RECYCABLE\_SERVO\_PIN);

  recycableServo.write(80);

  nonrecycableServo.write(80);

  // Initialize the SSD1306 display

  if(!display.begin(SSD1306\_SWITCHCAPVCC, SSD1306\_I2C\_ADDRESS)) {

    Serial.println(F("SSD1306 allocation failed"));

    for(;;);

  }

  display.clearDisplay();

  display.setTextSize(1);

  display.setTextColor(SSD1306\_WHITE);

  display.display();

  // Try connecting to WiFi

  checkAndReconnectWiFi();

  if (WiFi.status() == WL\_CONNECTED) {

    if (!MDNS.begin("WasteSorter")) {

      Serial.println("Error starting mDNS");

    }

  } else {

    // Handle the case where WiFi connection failed or is not available

    Serial.println("WiFi not connected. mDNS cannot be initialized. Starting Access Point...");

    startAccessPoint();

  }

}

Листинг 9 setup() функција уређаја Waste Sorter

Функција loop() (листинг 10) уређаја **Waste Sorter** управља основним оперативним функцијама уређаја. Принцип рада је сличан као код **Thermo Humid Meter** уређаја:

* **Рад у Access Point моду**: Ако је уређај у AP моду, обрађују се HTTP захтеви од корисника који се повезују на Wi-Fi мрежу коју емитује уређај.
* **Повезивање на мрежу и MQTT**: Ако уређај није у AP моду, понавља се провера и поновно повезивање на Wi-Fi мрежу и MQTT брокер, слично као код **Thermo Humid Meter**. Уколико је уређај повезан на Wi-Fi, али није на MQTT брокер, покушава се откривање брокера користећи mDNS.
* **Обрада очитавања са NFC/RFID тагова**: Уређај очитава податке са NFC/RFID тагова који се налазе у близини, и на основу њих одлучује у коју преграду ће се отпад сврстати. Ово је специфичан део логике за **Waste Sorter**.

Функција loop() осигурава да уређај континуирано обрађује захтеве и податке, чинећи га способним за ефикасно сортирање отпада у паметном дому.

void loop() {

  if (apMode) {

    server.handleClient(); // Handle HTTP requests in AP mode

  }

  else {

    checkAndReconnectWiFi();

    checkAndReconnectMQTT();

    // If not connected to MQTT Broker , search Broker service with mDNS

    if(WiFi.status() == WL\_CONNECTED && !client.connected()){

      Serial.println("Discovering services...");

      discoverMDNSService();

    }

    // MQTT loop

    client.loop();

    // Tag handling

    readNfcTag();

  }

  delay(1000);

}

Листинг 10 loop() функција уређаја Waste Sorter

Функција readNfcTag() служи за читање података са NFC тагова и одређивање типа отпада на основу тих података. Ова функција је кључна за рад уређаја **Waste Sorter** јер омогућава аутоматско сортирање отпада на основу информација прочитаних са RFID/NFC тагова. Кораци у функцији су следећи :

1. **Детекција тагова**: Функција започиње исписом поруке кориснику да постави NFC таг на читач. Затим, функција користи метод nfc.tagPresent(500) да провери присуство тагова у року од 500 милисекунди. Ако је таг присутан, наставља се са читањем.
2. **Читање NDEF поруке**: Уколико таг садржи NDEF поруку, функција nfc.read() учитава таг, а затим се пролази кроз све NDEF записе које порука садржи. За сваки запис, функција извлачи податке (payload) и конвертује их у читљиви текстуални низ, који се затим исписује у серијском монитору ради дебаговања.
3. **Обрада података са тагова**: Функција покушава да из текста који је прочитан са тагова извуче информацију о типу отпада, користећи помоћну функцију extractWasteType(). Ако је тип отпада успешно идентификован, аутоматски се позива функција handleThrownWaste() која покреће одговарајући процес сортирања отпада, у зависности од тога да ли је отпад рециклабилан или нерециклабилан.
4. **Управљање грешкама**: Ако таг не садржи NDEF поруку или је прочитана порука невалидна, функција исписује одговарајућу поруку и поново покушава читање.

Ова функција је суштинска за правилно функционисање уређаја **Waste Sorter**, омогућавајући му да ефикасно сортира отпад на основу података који се добијају са RFID/NFC тагова, што чини процес рециклирања аутоматизованим и кориснику пријатнијим.

void readNfcTag() {

  Serial.println("\nPlace an NFC tag on the reader.");

  // Look for tag for a few seconds

  if (nfc.tagPresent(500)) {

    NfcTag tag = nfc.read();

    if (tag.hasNdefMessage()) {

      NdefMessage message = tag.getNdefMessage();

      Serial.println("NDEF message found:");

      for (int i = 0; i < message.getRecordCount(); i++) {

        NdefRecord record = message.getRecord(i);

        int payloadLength = record.getPayloadLength();

        byte payload[payloadLength];

        record.getPayload(payload);

        // Extract text without the language code

        int languageCodeLength = payload[0];

        String payloadString = "";

        for (int c = languageCodeLength + 1; c < payloadLength; c++) {

          payloadString += (char)payload[c];

        }

        // Print the entire payload for debugging

        Serial.print("Record ");

        Serial.print(i);

        Serial.print(": ");

        Serial.println(payloadString);

        // Extract the waste\_type

        String wasteType = extractWasteType(payloadString);

        if (wasteType.length() > 0) {

          if (wasteType == WASTE\_TYPE\_RECYCLABLE || wasteType == WASTE\_TYPE\_NON\_RECYCLABLE) {

            handleThrownWaste(wasteType);

          } else {

            Serial.println("Invalid waste\_type.");

          }

        } else {

          Serial.println("waste\_type not found.");

        }

      }

    } else {

      Serial.println("No NDEF message found. Retrying...");

    }

  }

}

Листинг 11 readNfcTag() функција уређаја Waste Sorter

Функција handleThrownWaste() је централна за обраду сортираног отпада у овом уређају. Ова функција се активира када се прочита тип отпада са NFC таг-а, а њена улога је да аутоматски управља поклопцима контејнера и прати ниво попуњености сваког од њих. Кораци ове функције су следећи:

1. **Отварање поклопца**: Прва акција коју функција извршава је отварање одговарајућег поклопца на основу типа отпада који је прочитан са таг-а. Функција openLidForWasteType() се користи за управљање серво моторима који контролишу поклопце.
2. **Мерење нивоа попуњености**: Функција затим мери растојање унутар контејнера помоћу ултразвучних сензора recycableDistance и nonrecycableDistance. На основу ових података, израчунава се ниво попуњености контејнера помоћу функције calculateFillage().
3. **Ажурирање дисплеја**: Након израчунавања нивоа попуњености, функција updateDisplay() ажурира OLED дисплеј, приказујући кориснику актуелне нивое попуњености за рециклабилни и нерециклабилни отпад.
4. **Публикација MQTT порука**: Функција затим конструише и шаље две MQTT поруке на сервер:
   * Прва порука садржи информације о типу отпада који је управо сортиран.
   * Друга порука садржи податке о нивоу попуњености контејнера за оба типа отпада.
5. **Дебаговање**: За потребе дебаговања, функција исписује садржај обе поруке у серијски монитор, омогућавајући преглед података који су послати на сервер.

Ова функција је такође кључна за правилно функционисање уређаја **Waste Sorter**, јер омогућава аутоматско сортирање отпада и праћење нивоа попуњености контејнера, чиме се доприноси ефикасности и одрживости целокупног система.

void handleThrownWaste(const String& wasteType) {

  // Open corresponding lid

  openLidForWasteType(wasteType);

  // Calculate fillage for both waste types

  float recyclableDistanceValue = recycableDistance.measureDistanceCm();

  float nonRecyclableDistanceValue = nonrecycableDistance.measureDistanceCm();

  int recyclableFillage = calculateFillage(recyclableDistanceValue);

  int nonRecyclableFillage = calculateFillage(nonRecyclableDistanceValue);

  // Update display with fill levels

  updateDisplay(recyclableFillage, nonRecyclableFillage);

  // Define topic for messages

  String device\_topic = String(MQTT\_DEVICE\_TOPIC\_PREFIX) + String(device\_id) + String(MQTT\_RECORD\_TOPIC\_SUFIX);

  // Publish waste type message

  String waste\_type\_message = "{\"waste\_type\":\"" + wasteType + "\"}";

  client.publish(device\_topic.c\_str(), waste\_type\_message.c\_str());

  // Publish fillage message

  String fillage\_message = "{\"recyclable\_level\":\"" + String(recyclableFillage) + "\", \"non\_recyclable\_level\":\"" + String(nonRecyclableFillage) + "\"}";

  client.publish(device\_topic.c\_str(), fillage\_message.c\_str());

  // Print for debug

  Serial.print("Waste type message: ");

  Serial.println(waste\_type\_message.c\_str());

  Serial.print("Waste level message: ");

  Serial.println(fillage\_message.c\_str());

}

Листинг 12 handleThrownWaste() функција уређаја Waste Sorter

## Локални сервер (хаб)

Локални сервер, или **хаб**, представља кључну компоненту у IoT систему за управљање и обраду података који долазе од паметних уређаја, као што су Thermo Humid Meter и Waste Sorter. Хардверска и софтверска платформа овог локалног сервера је дизајнирана да ради и на микрорачунарима попут **Raspberry Pi**, чије су предности приступачност, флексибилност и ниска потрошња енергије, што их чини идеалним за употребу у **енергетски ефикасним IoT системима**.

На овом микрорачунару су имплементиране две кључне софтверске компоненте које раде паралелно и омогућавају свеобухватну функционалност система:

* **Eclipse Mosquitto MQTT**[[34]](#footnote-34) брокер и
* **FastAPI HTTP[[35]](#footnote-35)** сервер.

**MQTT брокер** је задужен за посредовање у комуникацији између уређаја у систему. Он слуша на одређеном мрежном порту и прихвата поруке од IoT уређаја. Ове поруке могу садржати различите врсте података, попут очитавања температуре и влажности или информација о врсти отпада који је убачен у Waste Sorter. MQTT је протокол дизајниран за лагану и ефикасну размену података, што је кључно за IoT системе који често имају ограничене ресурсе. Коришћење MQTT брокера омогућава ефикасно руковање подацима у реалном времену, уз минимално оптерећење мреже и ресурса на уређајима. Такође, овај брокер омогућава скалабилност система, што значи да се могу додавати нови уређаји без значајног утицаја на постојећи рад система. Конфигурација брокера је приказана на листингу 13.

listener 1883

protocol mqtt

allow\_anonymous false

password\_file /mosquitto/config/passwd

Листинг 13 Конфигурација Mosquitto MQTT брокера

**FastAPI** сервер, за разлику од MQTT брокера који се бави примањем и слањем података, одговоран је за конфигурацију и управљање уређајима. Овај сервер слуша на другом порту и омогућава клијентским уређајима који су повезани на исту мрежу да обављају разне операције конфигурације, као што је подешавање мрежних креденцијала или провера статуса локалног сервера. FastAPI је одабран због своје брзине и једноставности интеграције са различитим системима. Кроз овај HTTP сервер, корисници могу користити мобилну или веб апликацију за приступ конфигурацијским функцијама сервера. На пример, у случају да уређаји немају сачуване креденцијале корисничког налога, корисник може преко апликације послати нове креденцијале серверу, након чега сервер те податке користи зарад аутентификације и ауторизације. Endpoint-е дефинисане за овај сервер можете видети на листингу 14.

app = FastAPI(lifespan=lifespan)  
  
  
@app.get("/API/health")  
async def health\_check():  
 return {"status": "ok"}  
  
  
@app.put("/API/update-credentials")  
async def update\_credentials(credentials: UserCredentials):  
 settings.user\_email = credentials.email  
 settings.user\_password = credentials.password  
 update\_env\_file("user\_email", credentials.email)  
 update\_env\_file("user\_password", credentials.password)  
 return {"message": "Success"}

Листинг 14 Endpoint-и дефинисани за локални сервер

Локални сервер такође користи **mDNS (Multicast DNS)** технологију како би олакшао **откривање** своје услуге свим уређајима повезаним на исту мрежу. Коришћењем mDNS-а, сервер аутоматски емитује своју присутност и информације о мрежној адреси и портовима на којима су доступни различити сервиси, као што су MQTT брокер и HTTP сервер. Овај механизам омогућава уређајима у систему да без потребе за унапред дефинисаним IP адресама или ручним конфигурацијама открију локални сервер и започну комуникацију са њим. Употреба mDNS-а знатно поједностављује процес повезивања нових уређаја на систем и омогућава динамичније и флексибилније управљање IoT окружењем. За имплементацију овога се користи python библиотека zeroconf[[36]](#footnote-36) а код уз помоћ којег се то постиже је приказан на листингу 15.

def \_setup\_mdns(self):  
 desc = {'paths': ['/API/health', '/API/update-credentials']}  
 self.http\_hub\_service\_info = zeroconf.ServiceInfo(  
 type\_="\_http.\_tcp.local.",  
 name="WasteFreeHomeHTTPHub.\_http.\_tcp.local.",  
 addresses=[socket.inet\_aton(settings.hub\_hostname)],  
 port=settings.http\_port,  
 properties=desc,  
 server="waste-free-home-http-hub.local."  
 )  
 self.mqtt\_broker\_service\_info = zeroconf.ServiceInfo(  
 type\_="\_mqtt.\_tcp.local.",  
 name="WasteFreeHomeMQTTBroker.\_mqtt.\_tcp.local.",  
 addresses=[socket.inet\_aton(settings.hub\_hostname)],  
 port=settings.mqtt\_broker\_port,  
 properties=desc,  
 server="waste-free-home-mqtt-broker.local."  
 )  
 self.zeroconf.register\_service(self.http\_hub\_service\_info, ttl=3600)  
 print("mDNS HTTP Hub Service Registered")  
 self.zeroconf.register\_service(self.mqtt\_broker\_service\_info, ttl=3600)  
 print("mDNS Mqtt Broker Service Registered")

Листинг 15 Конфигурација и оглашавање сервиса хаба путем mDNS

На листингу 16 приказана је функција lifespan(), која служи за управљање акцијама током иницијализације и гашења FastAPI сервера. Ова функција обухвата низ корака који обезбеђују правилно функционисање локалног сервера (хаба) у оквиру IoT система.

При покретању хаба, он добија JWT токен како би био аутентификован на удаљеном серверу. Након успешне аутентификације, успоставља се конекција са MQTT брокером, чиме сервер добија могућност да прати поруке које шаљу уређаји на мрежи. У следећем кораку, у одвојеном loop-у, иницијализује се сервис који преко mDNS протокола оглашава присуство сервера на локалној мрежи, омогућавајући другим уређајима да му приступе.

Приликом гашења сервера, функција lifespan() осигурава правилно искључивање свих сервиса. Најпре се зауставља mDNS сервис, након чега се прекида конекција са MQTT брокером, чиме се затварају све активне сесије и обезбеђује интегритет података. Овај поступак омогућава контролисано гашење сервера без губитка података и са минималним утицајем на друге делове система.

@asynccontextmanager  
async def lifespan(app: FastAPI):  
 *# Initialize JWT and MQTT client* get\_jwt()  
 mqtt\_client.username\_pw\_set(settings.mqtt\_username, settings.mqtt\_password)  
 mqtt\_client.on\_connect = on\_connect  
 mqtt\_client.on\_message = on\_message  
 mqtt\_client.connect(host=settings.hub\_hostname, port=settings.mqtt\_broker\_port, keepalive=60)  
 mqtt\_client.loop\_start()  
  
 *# Initialize mDNS service* loop = asyncio.get\_event\_loop()  
 mdns\_service = await loop.run\_in\_executor(None, MDNSService)  
 try:  
 yield  
 finally:  
 *# Cleanup resources* await loop.run\_in\_executor(None, mdns\_service.close)  
 mqtt\_client.loop\_stop()  
 mqtt\_client.disconnect()

Листинг 16 Lifespan FastApi сервера

Листинг 17 приказује изглед **callback** функције (on\_message()) одговорне за обраду свих порука које локални сервер (хаб) прима путем MQTT протокола. Ова функција је кључна за правилно управљање комуникацијом између уређаја и хаба.

При примању поруке, callback функција најпре идентификује који је уређај послао поруку, као и тип акције која је извршена на уређају. Поред тога, функција анализира облик података који је укључен у поруци како би одредила на који начин треба обрадити и записати ту информацију. Након обраде, подаци се припремају за слање на удаљени сервер (листинзи 18).

Овај процес осигурава да информације са уређаја буду прецизно и ефикасно пренете на удаљени сервер, што омогућава адекватно праћење и анализу података у оквиру целокупног система.

def on\_message(client, userdata, msg):  
 topic\_parts = msg.topic.split("/")  
  
 *# Extract device ID* device\_id = topic\_parts[1]  
  
 *# Extract payload and convert it to json* data = msg.payload.decode("utf-8")  
 print(f"Received MQTT message: {data} from device {device\_id}")  
 json\_data = json.loads(data)  
  
 if topic\_parts[2] == "state":  
 toggle\_device\_state(device\_id, json\_data['state'])  
 elif topic\_parts[2] == "record":  
 send\_device\_record\_to\_server(device\_id, json\_data)  
 else:  
 print("Invalid topic")

Листинг 17 Callback финкција за пријем MQTT поруке на пријављеном topic-у

def send\_device\_record\_to\_server(device\_id, record):  
 *# Define the URL for the request* url = f"http://{settings.server\_hostname}:{settings.server\_port}/{settings.server\_records\_endpoint}/{device\_id}"  
 send\_request\_with\_retry(url=url, json\_data=record)

Листинг 18 Функција за слање записа на удаљени сервер

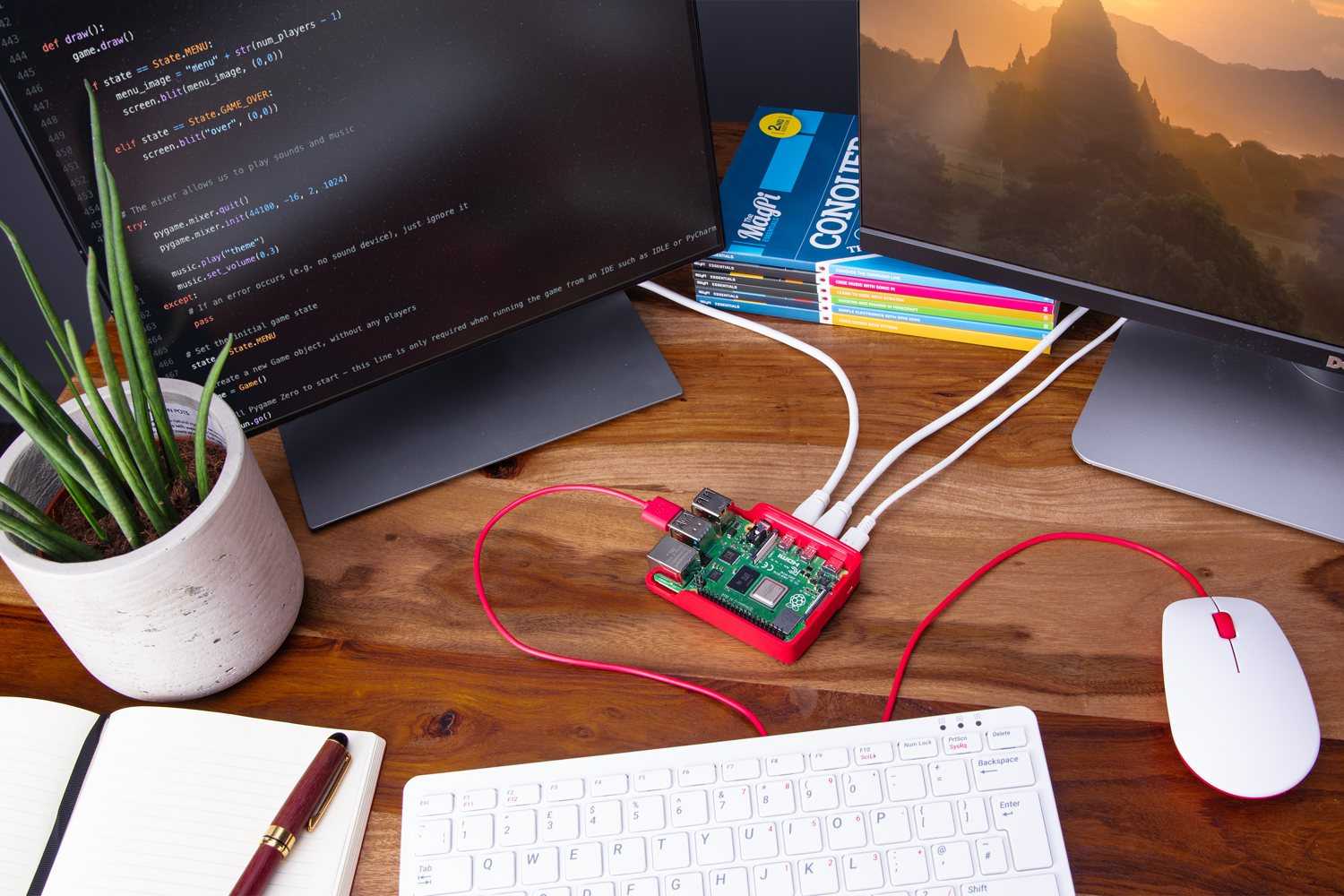
Функција request\_with\_retry() (листинг 19) је дизајнирана за слање HTTP POST захтева са подршком за понавне покушаје у случају неуспеха. Она користи библиотеку **httpx** за извршавање захтева и укључује заглавље за аутентификацију користећи **JWT** токен. Ако одговор садржи статусни код **401**, што указује на могући **истек** JWT токена, функција ће покушати да освежи токен и поново пошаље захтев до три пута. Такође, функција обрађује разне врсте грешака, укључујући специфичне HTTP статусне грешке и опште проблеме са захтевом, обезбеђујући да се све грешке правилно региструју. Овај механизам поновљених покушаја помаже у осигуравању да захтеви буду успешно обрађени чак и у случају привремених проблема са мрежом или сервером.

def send\_request\_with\_retry(url, json\_data=None, params=None, retries=3):  
 headers = {  
 "Authorization": f"Bearer {settings.jwt}"  
 }  
  
 try:  
 with httpx.Client() as client:  
 *# Send the POST request with both JSON data and query parameters* response = client.post(url, json=json\_data, headers=headers, params=params)  
 response.raise\_for\_status()  
 print(f"Response status code: {response.status\_code}")  
 return response  
  
 except httpx.HTTPStatusError as e:  
 *# Handle specific HTTP status errors* if e.response.status\_code == 401: *# Unauthorized, possibly due to expired JWT* if retries > 0:  
 get\_jwt() *# Refresh JWT token* return send\_request\_with\_retry(url=url, json\_data=json\_data, params=params, retries=retries - 1)  
 else:  
 print(f"HTTP Status error occurred: {e}")  
  
 except httpx.RequestError as e:  
 *# Handle request errors* if "Illegal header value" in str(e):  
 if retries > 0:  
 get\_jwt() *# Refresh JWT token* return send\_request\_with\_retry(url=url, json\_data=json\_data, params=params, retries=retries - 1)  
 else:  
 print(f"Request error occurred: {e}")  
  
 except Exception as e:  
 *# Handle any other unexpected errors* print(f"Unexpected error occurred: {e}")  
 return None

Листинг 19 Функција за слање HTTP POST захтева са поновним покушајима у случају грешке

**Raspberry Pi**[[37]](#footnote-37)(слике 30 и 31) представља одличан избор за хардвер хаба због своје енергетске ефикасности и способности да истовремено покреће више различитих операција. Овај микрорачунар омогућава конфигурацију оперативног система који подржава Docker окружење, што олакшава управљање и одржавање различитих сервиса. Иако Docker није обавезан, његова употреба може поједноставити надоградње и одржавање система.

Међутим, систем може бити имплементиран и на било којем другом хардверу који подржава Python, Docker и има адекватне спецификације. Важно је да хардвер има довољне ресурсе за покретање потребних сервиса и испуњава захтеве за енергетску ефикасност, али Raspberry Pi остаје изванредан избор због своје приступачности и флексибилности.

Слика 30 Raspberry Pi[[38]](#footnote-38) Слика 31 Raspberry Pi повезан са периферијама[[39]](#footnote-39)

Овај систем је осмишљен да буде једноставан за употребу и конфигурацију, али и да омогући напредне функције управљања подацима и уређајима у једној компактној и енергетски ефикасној платформи. Локални сервер, интегришући MQTT брокер и HTTP сервер, представља централну тачку комуникације и управљања у овом IoT систему, омогућавајући му да функционише поуздано и ефикасно у различитим условима.

За имплементацију локалног сервера коришћене су Python библиотеке, и њихове специфичне верзије, наведене на листингу 20.

annotated-types==0.7.0  
anyio==4.4.0  
certifi==2024.7.4  
click==8.1.7  
colorama==0.4.6  
dnspython==2.6.1  
email\_validator==2.2.0  
fastapi==0.111.0  
fastapi-cli==0.0.4  
h11==0.14.0  
httpcore==1.0.5  
httptools==0.6.1  
httpx==0.27.0  
idna==3.7  
ifaddr==0.2.0  
Jinja2==3.1.4  
markdown-it-py==3.0.0  
MarkupSafe==2.1.5  
mdurl==0.1.2  
orjson==3.10.6  
paho-mqtt==2.1.0  
pydantic==2.8.2  
pydantic-settings==2.3.4  
pydantic\_core==2.20.1  
Pygments==2.18.0  
python-dotenv==1.0.1  
python-multipart==0.0.9  
PyYAML==6.0.1  
rich==13.7.1  
shellingham==1.5.4  
sniffio==1.3.1  
starlette==0.37.2  
typer==0.12.3  
typing\_extensions==4.12.2  
ujson==5.10.0  
uvicorn==0.30.1  
watchfiles==0.22.0  
websockets==12.0  
zeroconf==0.132.2

Листинг 20 Python библиотеке коришћене приликом израде хаба

## Удаљени сервер

Удаљени сервер представља централну компоненту IoT система која је задужена за управљање и обраду података добијених од различитих уређаја, као и за омогућавање комуникације са клијентским апликацијама. На овом серверу се обављају кључни процеси који обезбеђују функционалност и сигурност целокупног система.

Сервер користи PostgreSQL за чување и управљање структурираним подацима, као што су информације о корисницима, уређајима и конфигурацијама. TimescaleDB, као проширење PostgreSQL-а, задужена је за управљање временским серијама података, омогућавајући праћење и анализу података кроз време, попут очитавања сензора у одређеним интервалима. Повезивање на обе базе је приказано на листингу 21, а неке од дефиниција модела на листингу 22.

class DatabaseManager:  
 def \_\_init\_\_(self, database\_url, models):  
 self.engine = create\_engine(database\_url)  
 self.SessionLocal = sessionmaker(autocommit=False, bind=self.engine)  
 models.metadata.create\_all(bind=self.engine)  
  
 def get\_session(self):  
 return self.SessionLocal()  
  
  
regular\_db\_manager = DatabaseManager(settings.regular\_database\_url, regular\_db\_base)  
time\_series\_db\_manager = DatabaseManager(settings.time\_series\_database\_url, time\_series\_db\_base)

Листинг 21 Инстацирање менаџера база података

regular\_db\_base = declarative\_base()

class BaseDevice(regular\_db\_base):  
 \_\_tablename\_\_ = "devices"  
  
 id = Column(UUID(as\_uuid=True), primary\_key=True, default=uuid.uuid4, nullable=False)  
 title = Column(String, nullable=False)  
 description = Column(String, nullable=False)  
 is\_online = Column(Boolean, default=False, nullable=False)  
 linked\_timestamp = Column(TIMESTAMP, primary\_key=True, server\_default=func.now())  
 owner\_id = Column(UUID(as\_uuid=True), ForeignKey("users.id", ondelete='SET NULL'), nullable=True)  
 owner = relationship("RegularUser", back\_populates="devices")  
  
 type = Column(Enum(DeviceType), nullable=False)  
 \_\_mapper\_args\_\_ = {  
 'polymorphic\_on': type  
 }  
  
  
class ThermoHumidMeter(BaseDevice):  
 \_\_mapper\_args\_\_ = {  
 'polymorphic\_identity': 'THERMO\_HUMID\_METER'  
 }  
  
  
class WasteSorter(BaseDevice):  
 \_\_mapper\_args\_\_ = {  
 'polymorphic\_identity': 'WASTE\_SORTER'  
 }

Листинг 22 Модели уређаја који се мапирају на табелу у бази података

Системом управља FastAPI, који комбинује HTTP и WebSocket сервере. HTTP сервер обрађује захтеве од клијентских апликација, пружајући одговоре у реалном времену и управљајући корисничким сесијама. WebSocket сервер омогућава реалновременску комуникацију, пријем података од уређаја и њихову дистрибуцију до клијентских апликација.

На серверу се користе FastAPI зависности (Depends) за верификацију JWT токена како би се осигурало да само овлашћени корисници могу приступити одређеним ресурсима. Поред тога, тамо где је та зависност потребна, сервер проверава да ли корисник има право приступа одређеним уређајима, чиме се осигурава да корисници могу управљати само оним уређајима који су у њиховом власништву. Ове зависности су приказане на листинзима 23 и 24.

async def get\_current\_user(token: Annotated[str, Depends(oauth2\_scheme)],  
 db: Annotated[Session, Depends(get\_regular\_db)]):  
 credentials\_exception = HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="Could not validate credentials",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
 try:  
 payload = decode\_access\_token(token)  
 if payload is None:  
 raise credentials\_exception  
 email: str = payload.get("sub")  
 if email is None:  
 raise credentials\_exception  
 token\_data = TokenData(email=email)  
 except InvalidTokenError:  
 raise credentials\_exception  
 user = get\_base\_user\_by\_email(db, token\_data.email)  
 if user is None:  
 raise credentials\_exception  
 return user

Листинг 23 Функција за верификацију валидности токена

def device\_dependency(device\_id: UUID,  
 current\_user: Annotated[RegularUser, Depends(get\_current\_active\_user)],  
 db: Session = Depends(get\_regular\_db)):  
 device = base\_device\_service.get\_device(db, device\_id)  
 if device is None or device.owner\_id != current\_user.id:  
 raise HTTPException(  
 status\_code=status.HTTP\_404\_NOT\_FOUND,  
 detail="Device not found",  
 )  
 return device

Листинг 24 Функција која поред валидности токена, проверава и посество уређаја

Када администратор дода нови уређај у систем, сервер генерише јединствени QR код за тај уређај, који кориснику касније омогућава да лако повежe свој уређај са својим налогом. Endpoint за ову акцију је приказан на листингу 25.

@device\_router.post(device\_router\_root\_path, tags=["Devices"])  
async def create\_device(  
 current\_admin: Annotated[Admin, Depends(user\_dependency([Role.ADMIN]))],  
 title: str = Form(...),  
 description: str = Form(...),  
 type: DeviceType = Form(...),  
 thumbnail: Optional[UploadFile] = Depends(validate\_thumbnail),  
 db: Session = Depends(get\_regular\_db)  
):  
 device\_schema = schemas.DeviceCreate(title=title, description=description, type=type)  
 db\_device = base\_device\_service.create\_device(db=db, device\_schema=device\_schema)  
  
 if thumbnail:  
 await image\_service.save\_device\_thumbnail(db\_device.id, thumbnail)  
  
 qr\_code\_img = utils.generate\_qr\_code(str(db\_device.id))  
 buf = io.BytesIO()  
 qr\_code\_img.save(buf)  
 buf.seek(0)  
  
 return StreamingResponse(buf, media\_type="image/png")

Листинг 25 Endpoint за креирање новог уређаја

Подаци који пристижу са корисничких хабова се обрађују и уписују у базу података. Осим тога, исти подаци се у реалном времену прослеђују клијентским апликацијама преко WebSocket-а. Endpoint за ову акцију као и конкретну функцију која врши упис у базу у зависности од атрибута послатог модела можемо видети на листинзима 26 и 27.

@records\_router.post(records\_router\_root\_path + "/{device\_id}", tags=["Records"],  
 response\_model=records\_response\_model)  
async def create\_device\_records(current\_device: Annotated[Device, Depends(device\_dependency)],  
 record: records\_response\_model,  
 models\_db: Session = Depends(get\_regular\_db),  
 time\_series\_db: Session = Depends(get\_time\_series\_db)):  
 db\_record = await records\_service.record\_device\_data(models\_db, time\_series\_db, current\_device.id, record)  
 if db\_record is None:  
 raise HTTPException(status\_code=400, detail="Bad request")  
  
 ws\_message = record.\_\_class\_\_(\*\*db\_record.\_\_dict\_\_).model\_dump()  
 ws\_message['timestamp'] = ws\_message['timestamp'].isoformat()  
 await records\_ws\_manager.broadcast(current\_device.id, str(ws\_message))  
 return db\_record

Листинг 26 Endpoint за чување записа уређаја

async def record\_device\_data(  
 models\_db: Session,  
 time\_series\_db: Session,  
 device\_id: UUID,  
 record):  
 device = base\_device\_service.get\_device(models\_db, device\_id)  
  
 model\_map = schema\_model\_map.get(device.type)  
 if model\_map is None:  
 return None  
 model\_class = model\_map.get(record.\_\_class\_\_)  
 if model\_class is None:  
 return None  
  
 model\_instance = model\_class(device\_id=device\_id, \*\*record.model\_dump(exclude={'timestamp'}))  
 time\_series\_db.add(model\_instance)  
 time\_series\_db.commit()  
 time\_series\_db.refresh(model\_instance)  
 return model\_instance

Листинг 27 Функција за упис записа уређаја у базу података

Цео систем је пројектован са флексибилношћу на уму, омогућавајући лако проширење и прилагођавање новим уређајима и типовима записа. Овај ток акција, укључујући обраду и чување података, као и њихово прослеђивање клијентским апликацијама, може се једноставно проширити додавањем нових уређаја у конфигурациону мапу

schema\_model\_map (листинг 28). Овим приступом омогућено је да се нови уређаји са специфичним типовима података интегришу у систем без потребе за значајним променама у постојећој архитектури.

schema\_model\_map = {  
 DeviceType.THERMO\_HUMID\_METER: {  
 schemas.ThermoHumidMeterRecord: time\_series\_models.ThermoHumidMeterRecord  
 },  
 DeviceType.WASTE\_SORTER: {  
 schemas.WasteSorterRecycleRecord: time\_series\_models.WasteSorterRecycleRecord,  
 schemas.WasteSorterLevelRecord: time\_series\_models.WasteSorterLevelRecord  
 }  
}

Листинг 28 Изглед мапе за тренутне уређаје и њихове типове записа

За имплементацију удаљеног сервера коришћене су Python библиотеке, и њихове специфичне верзије, наведене на листингу 29.

annotated-types==0.7.0  
anyio==4.3.0  
bcrypt==4.1.3  
certifi==2024.2.2  
click==8.1.7  
colorama==0.4.6  
dnspython==2.6.1  
email\_validator==2.1.1  
fastapi==0.111.0  
fastapi-cli==0.0.4  
greenlet==3.0.3  
h11==0.14.0  
httpcore==1.0.5  
httptools==0.6.1  
httpx==0.27.0  
idna==3.7  
iniconfig==2.0.0  
Jinja2==3.1.4  
markdown-it-py==3.0.0  
MarkupSafe==2.1.5  
mdurl==0.1.2  
orjson==3.10.3  
packaging==24.1  
passlib==1.7.4  
pluggy==1.5.0  
psycopg2==2.9.9  
pydantic==2.7.1  
pydantic-settings==2.2.1  
pydantic\_core==2.18.2  
Pygments==2.18.0  
PyJWT==2.8.0  
pypng==0.20220715.0  
pytest==8.3.2  
pytest-order==1.2.1  
python-dotenv==1.0.1  
python-multipart==0.0.9  
PyYAML==6.0.1  
qrcode==7.4.2  
rich==13.7.1  
shellingham==1.5.4  
sniffio==1.3.1  
SQLAlchemy==2.0.30  
starlette==0.37.2  
typer==0.12.3  
typing\_extensions==4.11.0  
ujson==5.10.0  
uvicorn==0.29.0  
watchfiles==0.21.0  
websockets==12.0

Листинг 29 Python библиотеке коришћене приликом израде удаљеног сервера

## Клијентска апликација

Клијентска апликација за овај систем представља крајњу компоненту која корисницима омогућава лако управљање и надзор над паметним уређајима у њиховим домовима. Развијена у Flutter-у, ова апликација је дизајнирана да ради на различитим платформама, обезбеђујући пријатан и интуитиван кориснички интерфејс.

Апликација омогућава корисницима да се пријаве на систем, управљају својим уређајима, и прегледају податке које ти уређаји генеришу. Једна од важних функционалности је могућност додавања нових уређаја путем QR кода који се генерише на удаљеном серверу. Корисник може брзо и једноставно додати уређај, након чега му апликација омогућава праћење података у реалном времену, као и историју измерених вредности.

Посебна пажња је посвећена испуњавању свих функционалних и нефункционалних захтева наведених у спецификацији пројекта. То укључује сигурност корисничких података, брзину одзива апликације, и могућност рада у реалном времену. Функционални захтеви као што су управљање уређајима и преглед података су потпуно имплементирани, док су нефункционални захтеви, као што су перформансе и скалабилност система, такође испоштовани.

Клијентска апликација користи следеће библиотеке како би обезбедила потребне функције и побољшала корисничко искуство:

* **dio** (^5.5.0+1)**:** библиотека за управљање HTTP захтевима на лак и ефикасан начин.
* **shared\_preferences**(^2.0.6)**:** користи се за чување података на уређају у трајном складишту, као што су корисничке поставке и информације о сесији.
* **auto\_route**(^9.0.0)**:** библиотека за управљање рутама у Flutter-у, која омогућава лако креирање и навигацију између различитих страница у апликацији.
* **flutter\_dotenv**(^5.0.2)**:** користи се за учитавање конфигурационих података из .env датотека, што олакшава управљање конфигурацијама у окружењу развоја и продукције.
* **fl\_chart**(^0.68.0)**:** библиотека за креирање интерактивних графикона и дијаграма, која омогућава визуализацију података у апликацији.
* **flutter\_date\_range\_picker**(^0.1.1)**:** омогућава избор опсега датума, што је корисно за филтрирање података у различитим временским периодима.
* **qr\_code\_scanner**(^1.0.1)**:** користи се за скенирање QR кодова, што омогућава брзо додавање нових уређаја у систем или приступање специфичним информацијама.
* **web\_socket\_channel**(^3.0.1)**:** омогућава комуникацију у реалном времену између апликације и сервера користећи WebSocket протокол.
* **bonsior**(^5.1.10)**:** библиотека која олакшава услуге мDNS (Multicast DNS) за откривање уређаја у локалној мрежи.

Ове библиотеке су изабране због својих могућности и подршке за функционалности које су кључне за рад апликације и њено повезивање са осталим компонентама IoT система.

Листинзи 30 и 31 приказују конфигурацију Flutter апликације, укључујући дефинисање тамне теме и основну поставку апликације. Први део кода дефинише darkTheme која се примењује када је апликација у тамном режиму. Ова тема укључује прилагођене боје, типографију, и подешавања за AppBar. На сличан начин је дефинисан и lightTheme. Други део кода иницијализује апликацију учитавањем конфигурационих података из .env датотеке, а затим покреће апликацију са конфигурисаним рутерима и темама. Ова поставка обезбеђује пријатан кориснички интерфејс у складу са изабраним светлосним или тамним режимом, и подржава динамичко усмеравање кроз апликацију.

ThemeData darkTheme = ThemeData(  
 brightness: Brightness.dark,  
 useMaterial3: true,  
 textTheme: const TextTheme(  
 displayLarge: TextStyle(fontSize: 32, fontWeight: FontWeight.*bold*),  
 bodyLarge: TextStyle(fontSize: 18, color: Colors.*white70*),  
 ),  
 appBarTheme: const AppBarTheme(  
 color: Color.fromRGBO(47, 95, 73, 1.0),  
 iconTheme: IconThemeData(color: Colors.*white*),  
 ),  
 colorScheme: ColorScheme.fromSeed(  
 seedColor: const Color.fromRGBO(21, 59, 48, 1.0),  
 brightness: Brightness.dark,  
 ),  
);

Листинг 30 Дефинисање тамне теме апликације

void main() async {  
 await dotenv.load(fileName: "assets/.env");  
 WidgetsFlutterBinding.*ensureInitialized*();  
 runApp(MyApp());  
}  
  
class MyApp extends StatelessWidget {  
 final \_appRouter = AppRouter();  
 MyApp({super.key});  
  
 @override  
 Widget build(BuildContext context) {  
 return MaterialApp.router(  
 routerConfig: \_appRouter.config(),  
 title: 'Waste Free Home',  
 debugShowCheckedModeBanner: false,  
 theme: lightTheme,  
 darkTheme: darkTheme,  
 );  
 }  
}

Листинг 31 Дефиниција korenske класе апликације

Листинг 31 приказује структуру рута у апликацији, дефинисану помоћу AutoRouter пакета у Flutter-у. Овај код омогућава навигацију корисника кроз различите делове апликације, укључујући пријаву, преглед и уређивање уређаја, и управљање хабом. Руте су заштићене коришћењем AuthGuard механизма (листинг 32), који осигурава да само аутентификовани корисници могу приступити одређеним страницама, чиме се обезбеђује додатни ниво сигурности у апликацији.

part 'app\_router.gr.dart';  
  
@AutoRouterConfig()  
class AppRouter extends RootStackRouter {  
 @override  
 List<AutoRoute> get routes => [  
 AutoRoute(path: '/login', page: LoginRoute.*page*),  
 AutoRoute(path: '/', page: MainRoute.*page*, initial: true, guards: [  
 AuthGuard()  
 ], children: [  
 AutoRoute(path: 'devices', page: MyDevicesRoute.*page*, initial: true),  
 AutoRoute(path: 'hub', page: HubRoute.*page*),  
 AutoRoute(path: 'device\_ap', page: DeviceAPRoute.*page*),  
 ]),  
 AutoRoute(  
 path: '/devices/:id',  
 page: DeviceDetailsRoute.*page*,  
 guards: [AuthGuard()]),  
 AutoRoute(  
 path: '/devices/:id/edit',  
 page: EditDeviceDetailsRoute.*page*,  
 guards: [AuthGuard()]),  
 AutoRoute(  
 path: '/devices/link/qr-scan',  
 page: QRScanRoute.*page*,  
 guards: [AuthGuard()]),  
 ];  
}

Листинг 32 Дефиниција рута апликације

class AuthGuard extends AutoRouteGuard {  
 @override  
 Future<void> onNavigation(  
 NavigationResolver resolver, StackRouter router) async {  
 final AuthService authService = AuthService();  
 try {  
 await authService.checkTokenValidity();  
 resolver.next(true);  
 } catch (e) {  
 router.replaceAll([const LoginRoute()]);  
 }  
 }  
}

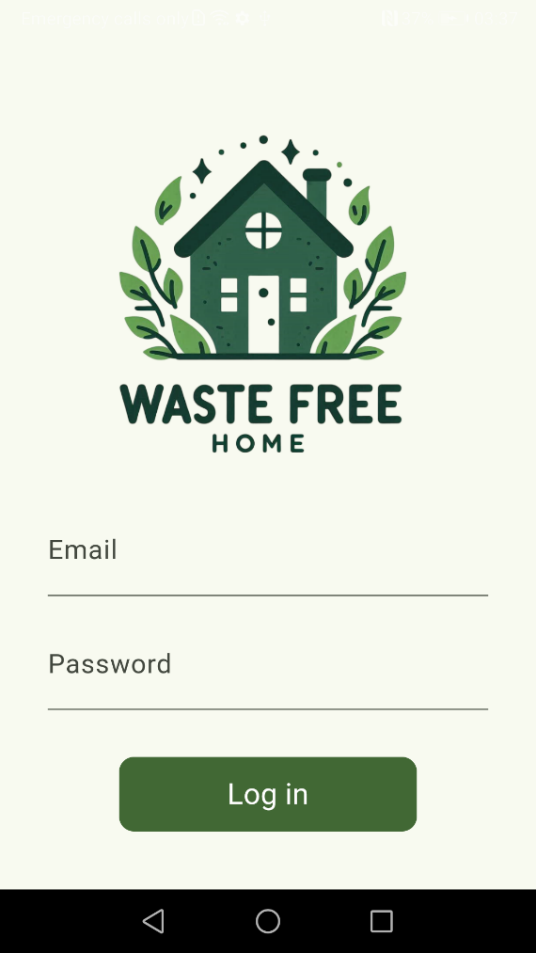
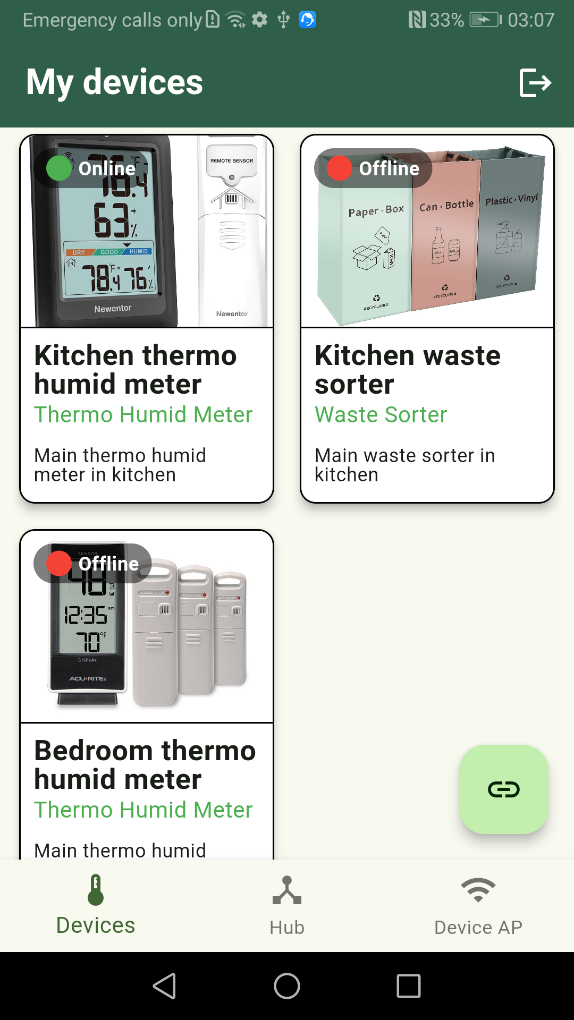
Листинг 33 Дефиниција класе за проверу аутентификованости

У наредном поглављу (Демонстрација) биће детаљније приказане функционалности које ова апликација омогућава корисницима. Ова демонстрација ће укључивати реалне примере коришћења апликације у различитим сценаријима, чиме ће се показати како систем функционише у пракси.

# Демонстрација

У овом поглављу биће приказане кључне радње које корисници могу извршити у апликацији *WasteFreeHome*. Радње ће бити демонстриране кроз слике екрана које илуструју процес њиховог извршавања. Ове демонстрације обухватају пријаву на систем, преглед свих уређаја и њихове доступности, додавање новог уређаја, ажурирање креденцијала хаба и уређаја, као и детаљан преглед података које уређај прикупља.

## Пријава на систем

Корисник уноси корисничко име и лозинку у формулар за пријаву (слика 32). Након притиска на дугме за пријаву, систем проверава унете креденцијале са сервером, и уколико су исправни пребацује корисника на главну страницу (слика 33) уз пропратну поруку. Уколико су креденцијали неисправни исписује се порука грешке.  
  
 

Слика 32 Страница за пријаву Слика 33 Приказ свих корисникових паметних уређаја

## Почетна страница

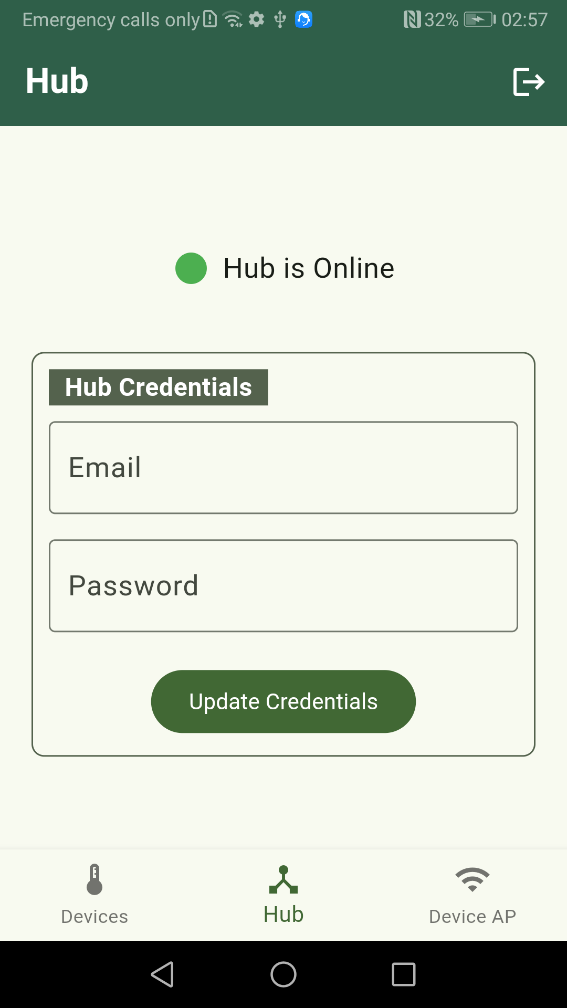
Након пријаве, корисник приступа главној страници апликације где је почетно приказан прозор за преглед свих корисникових уређаја (слика 33). На дну ове странице се налази и навигација за измену прозора главне странице која садржи преглед свих паметних уређаја, подешавање хаба и подешавање паметног уређаја. На врху странице се налази назив тренутне странице, као и дугме за одјаву са апликације.

## Преглед свих корисникових паметних уређаја и њихове доступности

Корисник бира опцију за преглед својих уређаја (симбол термометра са текстом „Devices“ на слици 33). У овом прозору се приказује листа свих уређаја повезаних са корисничким налогом, укључујући статус њихове доступности. Такође, на екрану је приказано и плутајуће дугме које омогућава повезивање новог уређаја са корисничким налогом.

## Додавање новог паметног уређаја

Корисник бира опцију за додавање новог уређаја (дугме са симболом ланца на слици 33) и прати упутства за повезивање уређај са налогом (слика 34). Процес се завршава скенирањем QR кода (углавном смештеним на уређају). Након скенирања кориснику се приказује пратећа порука о успешности додавања.

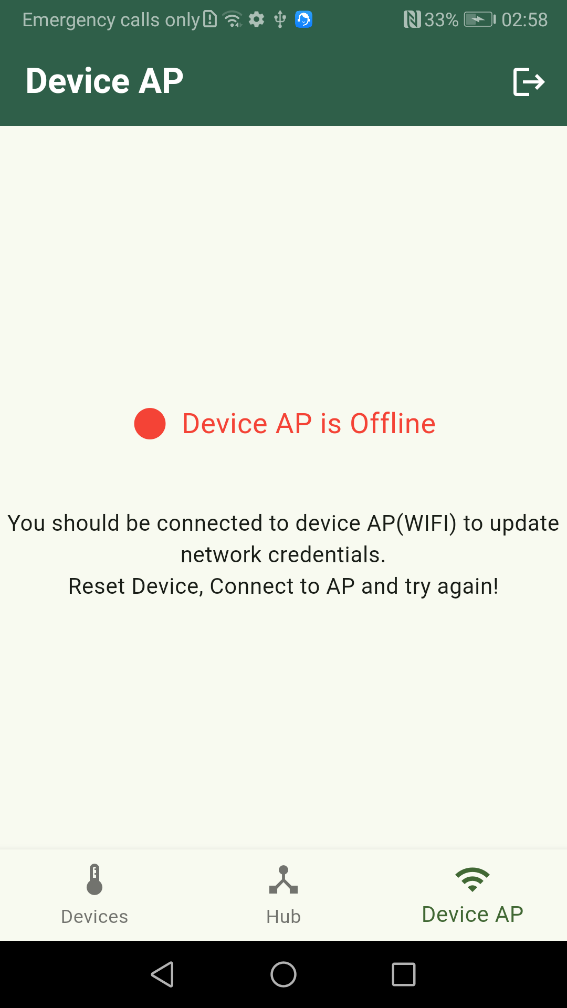
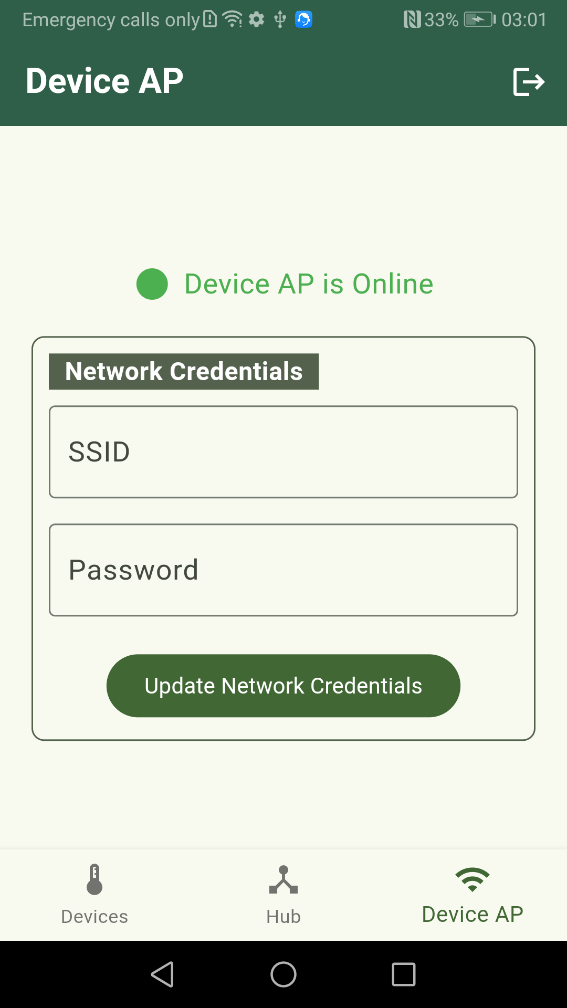
Слика 34 Скенирање QR кода приликом додавања уређаја Слика 35 Ажурирање креденцијала налога на хабу

## Ажурирање креденцијала хаба

Предуслов ове акције је повезаност корисничког уређаја, на ком је отворена апликација, на исту мрежу као и сам хаб. Корисник у навигацији проналази дугме (симбол конекције са текстом „Hub“ на слици 33) које га усмерава на прозор за конфигурацију хаба. На овом прозору се налази форма за унос корисникових креденцијала (слика 35). Након потврде, нови креденцијали се чувају у на хабу он се аутоматски конфигурише да их користи. Уколико се хаб не налази на истој мрежи као и кориснички уређај, на овом прозору ће се приказати пропратна порука.

## Ажурирање мрежних креденцијала паметног уређаја

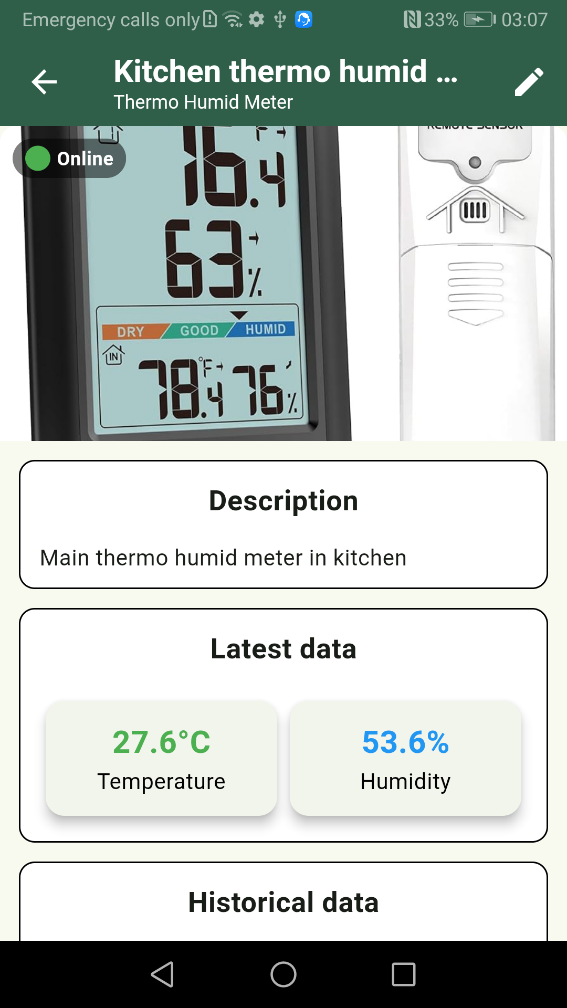
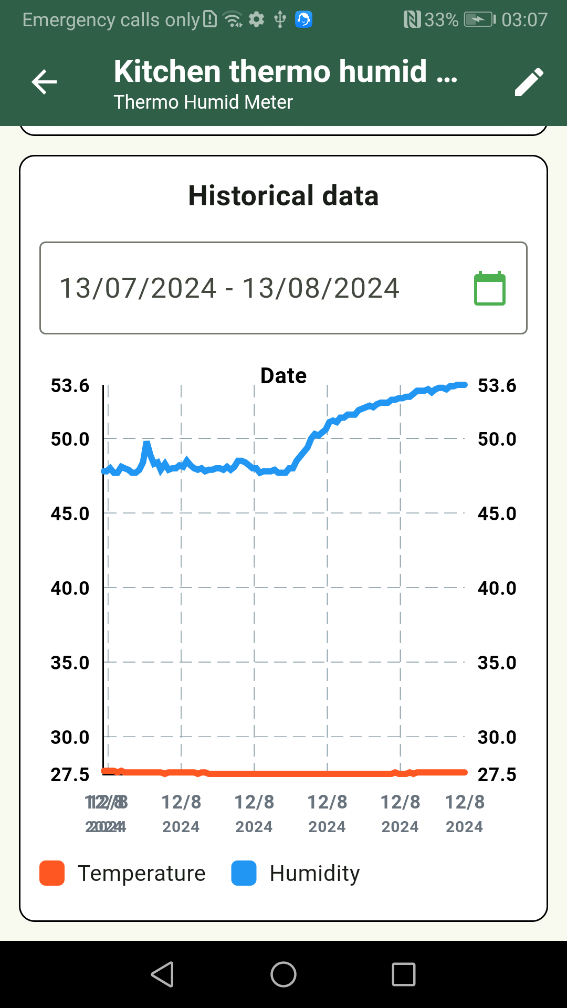
Предуслов ове акције је повезаност корисничког уређаја, на ком је отворена апликација, на Wi-Fi мрежу коју емитује паметни уређај који се конфигурише. Корисник у навигацији проналази дугме (симбол цртица сигнала са текстом „Device AP“ на слици 33) које га усмерава на прозор за конфигурацију паметног уређаја. Након уноса нових мрежних креденцијала у форму (слика 37) и притиска на дугме за потврду, креденцијали се шаљу на паметни уређај и он аутоматски ажурира своја подешавања. Након ажурирања подешавања, паметни уређај се рестартује и уколико су креденцијали тачни повезује се на мрежу, а затим и на хаб и креће са својим радом. Уколико се паметни уређај не налази на истој мрежи као и кориснички уређај, на овом прозору ће се приказати пропратна порука (слика 36).

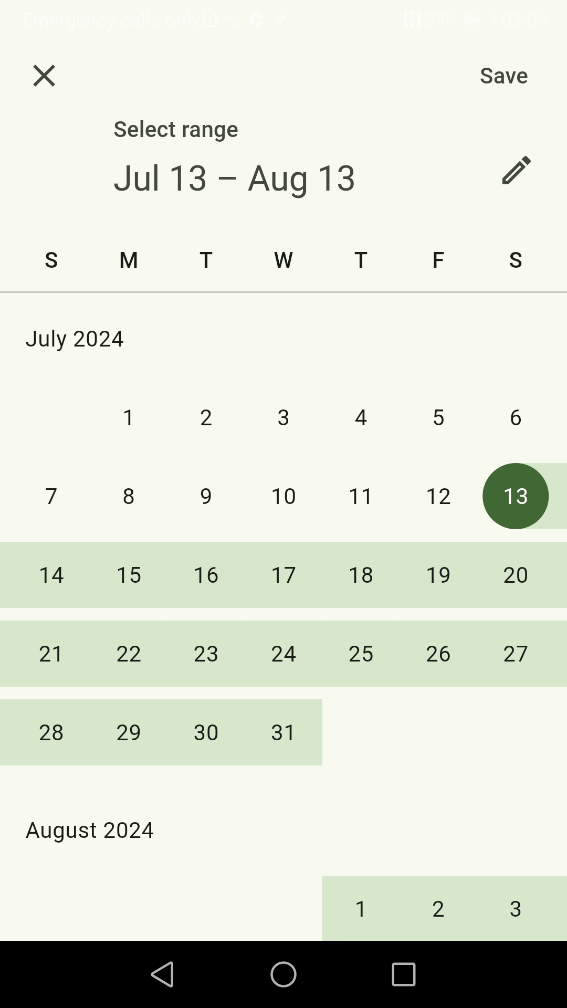
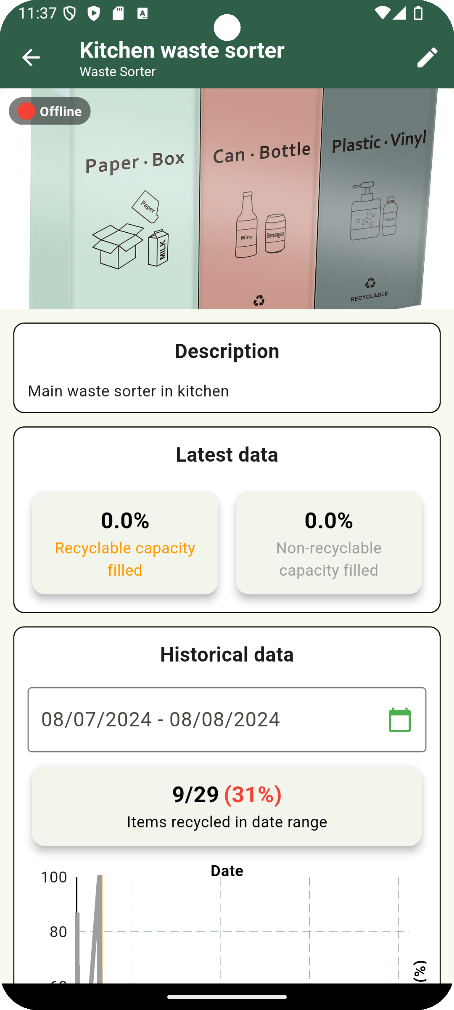
Слика 36 Уређај недоступан за конфигурацију Слика 37 Уређај доступан за конфигурацију

## Детаљан преглед паметног уређаја и његових прикупљених података

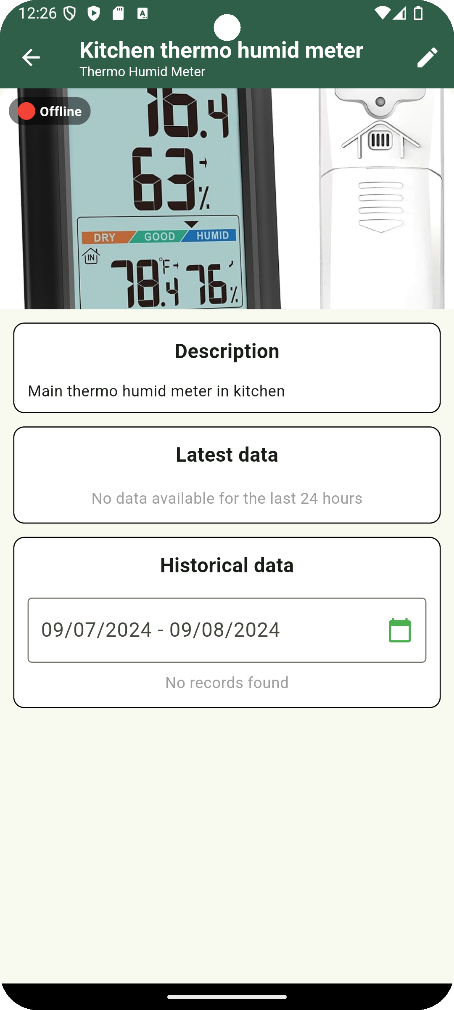
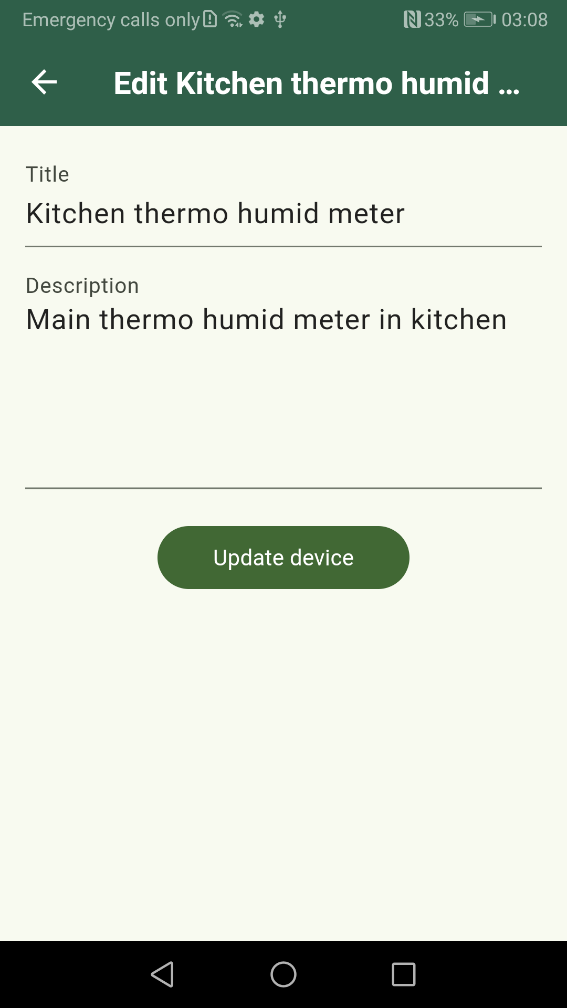
Корисник, који се налази на прозору „Devices“ бира уређај из листе, након чега му се отвара детаљан преглед тог уређаја (слике 38, 41, 42). На екрану се приказују основне информације о уређају, његова слика, најновији подаци прикупљени од уређаја, као и графички приказ историјских података (слика 39), омогућавајући анализу трендова. Одабир датума за који се прате трендови се врше притиском на симбол календара након чега се отвара прозор за одабир (слика 40).

Слика 38 Детаљан приказ уређаја Thermo Humid Meter Слика 39 Графички приказ историјских података

Слика 40 Календар за одабир датума Слика 41 Детаљан приказ уређаја Waste Sorter

Слика 42 Детаљан приказ уређаја Thermo   
Humid Meter без прикупљених података Слика 43 Измена основних информација о уређају

## Измена основних информација о паметном уређају

На екрану детаљног прегледа уређаја (слике 38, 41, 42), корисник притиска икону оловке у горњем десном углу, чиме се отвара прозор за измену основних информација о паметном уређају (слика 43). Корисник може ажурирати жељене информације у форми, а затим притиском на дугме за потврду, при чему се измене одмах ажурирају у систему. По успешној промени података, корисник се враћа на преглед уређаја, уз приказ поруке која потврђује успешност извршене акције.

# Закључак

У овом раду представљен је комплетан IoT систем који је осмишљен и имплементиран са циљем да обезбеди ефикасно, скалабилно и кориснички пријатно решење за управљање паметним уређајима у оквиру концепта „зеленог дома“. Испуњени су сви функционални и нефункционални захтеви дефинисани у спецификацији. Систем омогућава лако повезивање и управљање различитим типовима уређаја, као и прикупљање и анализу података у реалном времену, што резултира једноставним и флуидним корисничким искуством.

Имплементација апликације у Flutter-у омогућава висок степен прилагодљивости на различите платформе, док серверски део система, базиран на FastAPI-у и PostgreSQL/TimescaleDB базама података, обезбеђује оптималне перформансе и поуздано управљање подацима. Скалирање система је осигурано преко архитектуре која омогућава једноставно додавање нових уређаја и проширење постојећих функционалности. Такође, систем је осмишљен тако да се лако имплементира и одржава, уз минималне захтеве за техничким знањем корисника.

Иако је систем у потпуности функционалан, могуће су даље надоградње у циљу побољшања безбедносних аспеката. Једна од тих надоградњи је увођење сертификата за безбедну комуникацију између хабова и удаљеног сервера. Овим приступом, сваки хаб би био идентификован као јединствени ентитет у оквиру удаљеног сервера, чиме би се осигурало да сваки хаб самостално управља својим подацима, без слања информација у име корисника. Поред тога, употреба HTTPS протокола за све комуникације унутар система додатно би заштитила осетљиве податке који се размењују, чиме би се повећала укупна сигурност система. Ови механизми би додатно заштитили осетљиве податке који се размењују, чиме би се повећала укупна сигурност система.

Такође, употреба refresh token-а за управљање корисничким сесијама представља корак ка побољшању корисничког искуства. Овај механизам би омогућио дуже трајање сесија без потребе за честим поновним пријављивањем, чиме би се олакшала употреба апликације, посебно за кориснике који често користе систем.

С обзиром на будући развој система и потенцијално повећање обима посла, разматра се и могућност преласка са монолитне архитектуре на микросервисну. Овакав приступ би омогућио бољу дистрибуцију оптерећења, лакше одржавање и брже увођење нових функционалности. Увођење gateway-а у систем би додатно унапредило начин управљања саобраћајем између компоненти система, чиме би се побољшала ефикасност и сигурност целокупне архитектуре.

Сумирајући, имплементирани систем пружа снажну основу за даљи развој и проширивање функционалности, док истовремено обезбеђује висок ниво перформанси, сигурности и корисничког задовољства. У будућим фазама развоја, фокус ће бити на унапређењу безбедносних аспеката и преласку на архитектуре које омогућавају још већу скалабилност и прилагодљивост, чиме би се систем додатно унапредио и прилагодио све већим захтевима корисника.

# Литература

# Биографија

Јован Јокић је рођен 9. марта 2001. године у Сремској Митровици. Завршио је Основну школу „Свети Сава“ као носилац Вукове дипломе. Средњу школу завршио је у Митровачкој гимназији у Сремској Митровици, где је био одличан ученик на рачунарском смеру. Године 2020. уписао је Факултет техничких наука Универзитета у Новом Саду, где постиже изузетне резултате са просечном оценом 9,45. Током студија активно учествује на такмичењима из сајбер безбедности.

1. Слика преузета са [How to Use Git Branches & Buddy to Organize Project Code — SitePoint](https://www.sitepoint.com/use-git-branches-buddy/) [↑](#footnote-ref-1)
2. Слика преузета са [Part 1. Databases for beginners - What is a database? What is PostgreSQL? | ElephantSQL](https://www.elephantsql.com/blog/databases-for-beginners-what-is-a-database-what-is-postgresql.html) [↑](#footnote-ref-2)
3. Слике преузете са [TimescaleDB vs. Postgres for time-series: 20x higher inserts, 2000x faster deletes, 1.2x-14,000x faster queries | by Rob Kiefer | Timescale | Medium](https://medium.com/timescale/timescaledb-vs-6a696248104e) [↑](#footnote-ref-3)
4. Слика преузета са [Migrating to SaaS - Baytech Consulting](https://www.baytechconsulting.com/blog/why-python-development-is-hot-in-2019-current-trends) [↑](#footnote-ref-4)
5. Слика преузета са [What is a REST API? Beginner's Guide (altexsoft.com)](https://www.altexsoft.com/blog/rest-api-design/) [↑](#footnote-ref-5)
6. Слика преузета са [What is a JWT? Understanding JSON Web Tokens (supertokens.com)](https://supertokens.com/blog/what-is-jwt) [↑](#footnote-ref-6)
7. Слика преузета са [What Is WebSocket? How Does It Work? | Gcore](https://gcore.com/learning/what-is-websocket/) [↑](#footnote-ref-7)
8. Слика преузета са [What is mDNS(Multicast DNS)? » Network Interview](https://networkinterview.com/what-is-mdnsmulticast-dns/) [↑](#footnote-ref-8)
9. Слика преузета са [How to Install and Configure the Arduino IDE - Circuit Basics](https://www.circuitbasics.com/how-to-install-and-configure-the-arduino-ide/) [↑](#footnote-ref-9)
10. Слика преузета са [ESP32 Pinout Reference - Last Minute Engineers](https://lastminuteengineers.com/esp32-pinout-reference/) [↑](#footnote-ref-10)
11. Слика преузета са [ESP8266 Pinout Reference - Last Minute Engineers](https://lastminuteengineers.com/esp8266-pinout-reference/) [↑](#footnote-ref-11)
12. Слика преузета са [Creating a Cross-Platform App with Flutter — Adapptor - Perth App Developers](https://www.adapptor.com.au/blog/creating-a-cross-platform-app-with-flutter) [↑](#footnote-ref-12)
13. Слика преузета са [What is a Container? | Docker](https://www.docker.com/resources/what-container/) [↑](#footnote-ref-13)
14. [lucidchart.com)](https://www.lucidchart.com/) [↑](#footnote-ref-14)
15. [adafruit/Adafruit\_BusIO: Arduino library for I2C & SPI abstractions (github.com)](https://github.com/adafruit/Adafruit_BusIO) [↑](#footnote-ref-15)
16. [adafruit/Adafruit-GFX-Library: Adafruit GFX graphics core Arduino library, this is the 'core' class that all our other graphics libraries derive from (github.com)](https://github.com/adafruit/Adafruit-GFX-Library) [↑](#footnote-ref-16)
17. [adafruit/Adafruit\_SSD1306: Arduino library for SSD1306 monochrome 128x64 and 128x32 OLEDs (github.com)](https://github.com/adafruit/Adafruit_SSD1306) [↑](#footnote-ref-17)
18. [adafruit/Adafruit\_Sensor: Common sensor library (github.com)](https://github.com/adafruit/Adafruit_Sensor) [↑](#footnote-ref-18)
19. [adafruit/DHT-sensor-library: Arduino library for DHT11, DHT22, etc Temperature & Humidity Sensors (github.com)](https://github.com/adafruit/DHT-sensor-library) [↑](#footnote-ref-19)
20. [jkb-git/ESP32Servo: Arduino-compatible servo library for the ESP32 (github.com)](https://github.com/jkb-git/ESP32Servo) [↑](#footnote-ref-20)
21. [gamegine/HCSR04-ultrasonic-sensor-lib: Arduino lib for HCSR04 ultrasonic sensor (github.com)](https://github.com/gamegine/HCSR04-ultrasonic-sensor-lib) [↑](#footnote-ref-21)
22. [PN532/NDEF at PN532\_HSU · elechouse/PN532 (github.com)](https://github.com/elechouse/PN532/tree/PN532_HSU/NDEF) [↑](#footnote-ref-22)
23. [PN532/PN532 at PN532\_HSU · elechouse/PN532 (github.com)](https://github.com/elechouse/PN532/tree/PN532_HSU/PN532) [↑](#footnote-ref-23)
24. [PN532/PN532\_I2C at PN532\_HSU · elechouse/PN532 (github.com)](https://github.com/elechouse/PN532/tree/PN532_HSU/PN532_I2C) [↑](#footnote-ref-24)
25. [knolleary/pubsubclient: A client library for the Arduino Ethernet Shield that provides support for MQTT. (github.com)](https://github.com/knolleary/pubsubclient) [↑](#footnote-ref-25)
26. Слика преузета са [Amazon.com: HiLetgo 1PC ESP8266 NodeMCU CP2102 ESP-12E Development Board Open Source Serial Module Works Great for Arduino IDE/Micropython (Small) : Electronics](https://www.amazon.com/HiLetgo-Internet-Development-Wireless-Micropython/dp/B010O1G1ES) [↑](#footnote-ref-26)
27. Слика преузета са [DHT22 Sensor Pinout, Specs, Equivalents, Circuit & Datasheet (components101.com)](https://components101.com/sensors/dht22-pinout-specs-datasheet) [↑](#footnote-ref-27)
28. Слика преузета са [ESP32 ESP-WROOM-32 Bluetooth i WIFI modul CP2102 Type C - Elektroleum](https://elektroleum.rs/product/esp-wroom-32-esp32-bluetooth-i-wifi-dual-core-cpu-cp2102-type-c/) [↑](#footnote-ref-28)
29. Слика преузета са [Using the PN532 RFID reader with Arduino • AranaCorp](https://www.aranacorp.com/en/using-the-pn532-rfid-reader-with-arduino/) [↑](#footnote-ref-29)
30. Слика преузета са [How the HC-SR04 Sensor Works and Example Programs with Arduino - NN Digital | Learn Arduino, ESP8266 / NodeMCU, STM32, Raspberry Pi, Microcontroller and Other Information Technology (nn-digital.com)](https://www.nn-digital.com/en/blog/2019/08/07/how-the-hc-sr04-sensor-works-and-example-programs-with-arduino/) [↑](#footnote-ref-30)
31. Слика преузета са [Micro Servo MG90S 2.5Kg (naylampmechatronics.com)](https://naylampmechatronics.com/servomotores/246-micro-servo-mg90s.html) [↑](#footnote-ref-31)
32. Слика преузета са [SSD1306 OLED Display — ESPHome](https://esphome.io/components/display/ssd1306.html) [↑](#footnote-ref-32)
33. Слика преузета са [NTAG 213 Wet Inlay - Round 25mm | Universal Smart Cards Ltd (usmartcards.co.uk)](https://www.usmartcards.co.uk/ntag-213-wet-inlay-round-25mm) [↑](#footnote-ref-33)
34. [Eclipse Mosquitto](https://mosquitto.org/) [↑](#footnote-ref-34)
35. [FastAPI (tiangolo.com)](https://fastapi.tiangolo.com/) [↑](#footnote-ref-35)
36. [zeroconf · PyPI](https://pypi.org/project/zeroconf/) [↑](#footnote-ref-36)
37. [Raspberry Pi](https://www.raspberrypi.com/) [↑](#footnote-ref-37)
38. Слика преузета са [Raspberry PI 5 model B 4GB RAM - Elektroleum](https://elektroleum.rs/product/raspberry-pi-5-model-b-4gb-ram/) [↑](#footnote-ref-38)
39. Слика преузета са [Raspberry Pi for home – Raspberry Pi](https://www.raspberrypi.com/for-home/) [↑](#footnote-ref-39)