|  |  |  |
| --- | --- | --- |
|  | 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 решење према израђеној спецификацији.  5. Тестирати имплементирано софтверско решење.  6. Документовати (1), (2), (3), (4) и (5). |

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

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

**Садржај**

[1 Увод 5](#_Toc174501996)

[2 Oснове IoT система 6](#_Toc174501997)

[2.1 Увод у IoT системе 6](#_Toc174501998)

[2.2 Архитектура IoT система 6](#_Toc174501999)

[2.3 Комуникациони протоколи и стандарди 6](#_Toc174502000)

[2.4 Сигурност у IoT системима 6](#_Toc174502001)

[2.5 Примене IoT система 7](#_Toc174502002)

[3 Технологије и концепти 8](#_Toc174502003)

[3.1 Git 8](#_Toc174502004)

[3.2 PostgreSQL 9](#_Toc174502005)

[3.3 TimescaleDB 10](#_Toc174502006)

[3.4 Python 12](#_Toc174502007)

[3.4.1 REST (Representational State Transfer) 13](#_Toc174502008)

[3.4.2 JWT (JSON Web Token) 14](#_Toc174502009)

[3.4.3 WebSocket 16](#_Toc174502010)

[3.4.4 mDNS (Multicast DNS) 17](#_Toc174502011)

[3.4.5 Pydantic 18](#_Toc174502012)

[3.4.6 SQLAlchemy 18](#_Toc174502013)

[3.4.7 FastAPI 19](#_Toc174502014)

[3.5 Arduino IDE 20](#_Toc174502015)

[3.6 ESP модули 21](#_Toc174502016)

[3.6.1 ESP32 22](#_Toc174502017)

[3.6.2 ESP8266 22](#_Toc174502018)

[3.7 Flutter 23](#_Toc174502019)

[3.8 Docker 24](#_Toc174502020)

[4 Мотивација 26](#_Toc174502021)

[5 Спецификација 27](#_Toc174502022)

[5.1 Спецификација захтева 28](#_Toc174502023)

[5.1.1 Функциконални захтеви 28](#_Toc174502024)

[5.1.2 Нефункционални захтеви 32](#_Toc174502025)

[5.2 Спецификација система 32](#_Toc174502026)

[5.2.1 Модел података 32](#_Toc174502027)

[5.2.2 Архитектура система 32](#_Toc174502028)

[6 Имплементација система 33](#_Toc174502029)

[7 Закључак 34](#_Toc174502030)

[8 Литература 35](#_Toc174502031)

[9 Биографија 36](#_Toc174502032)

# 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

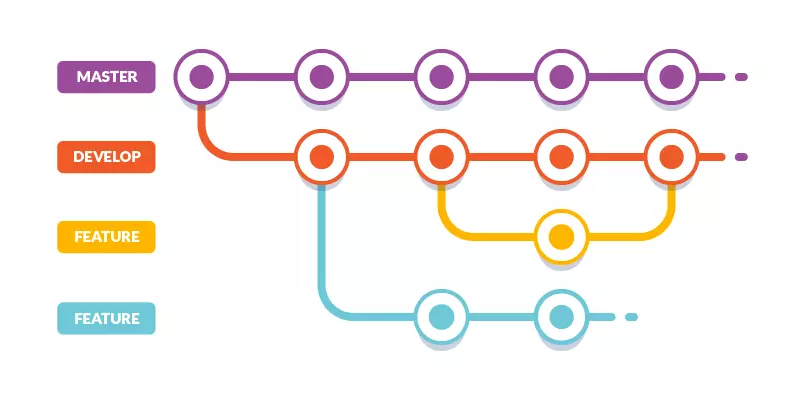
Ово поглавље ће се фокусирати на **технолошке компоненте** и **концепте** који су коришћени у развоју **IoT система** описаног у овом раду. За реализацију система за праћење **еколошких уређаја**, примењене су различите **напредне технологије** и **алати**. У наставку су детаљно представљене **кључне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

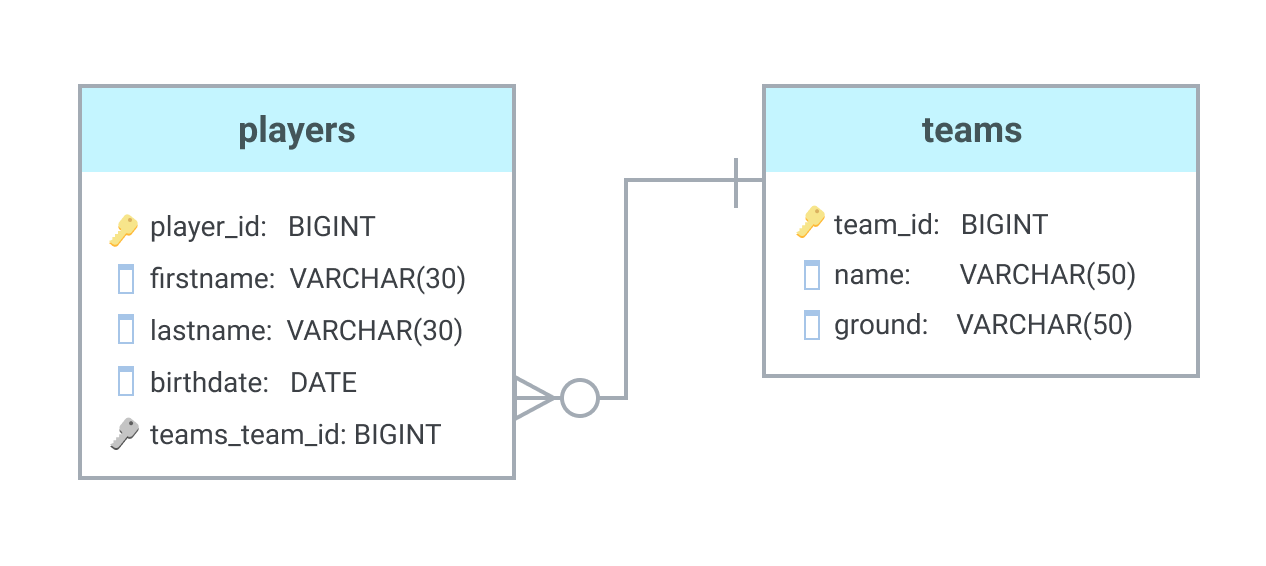
**Git** се интегрише са различитим алатима и платформама за управљање пројектима, као што су **CI/CD системи**, алати за преглед кода и менаџмент пројеката, што га чини основним алатом у савременом развоју софтвера.

## 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.

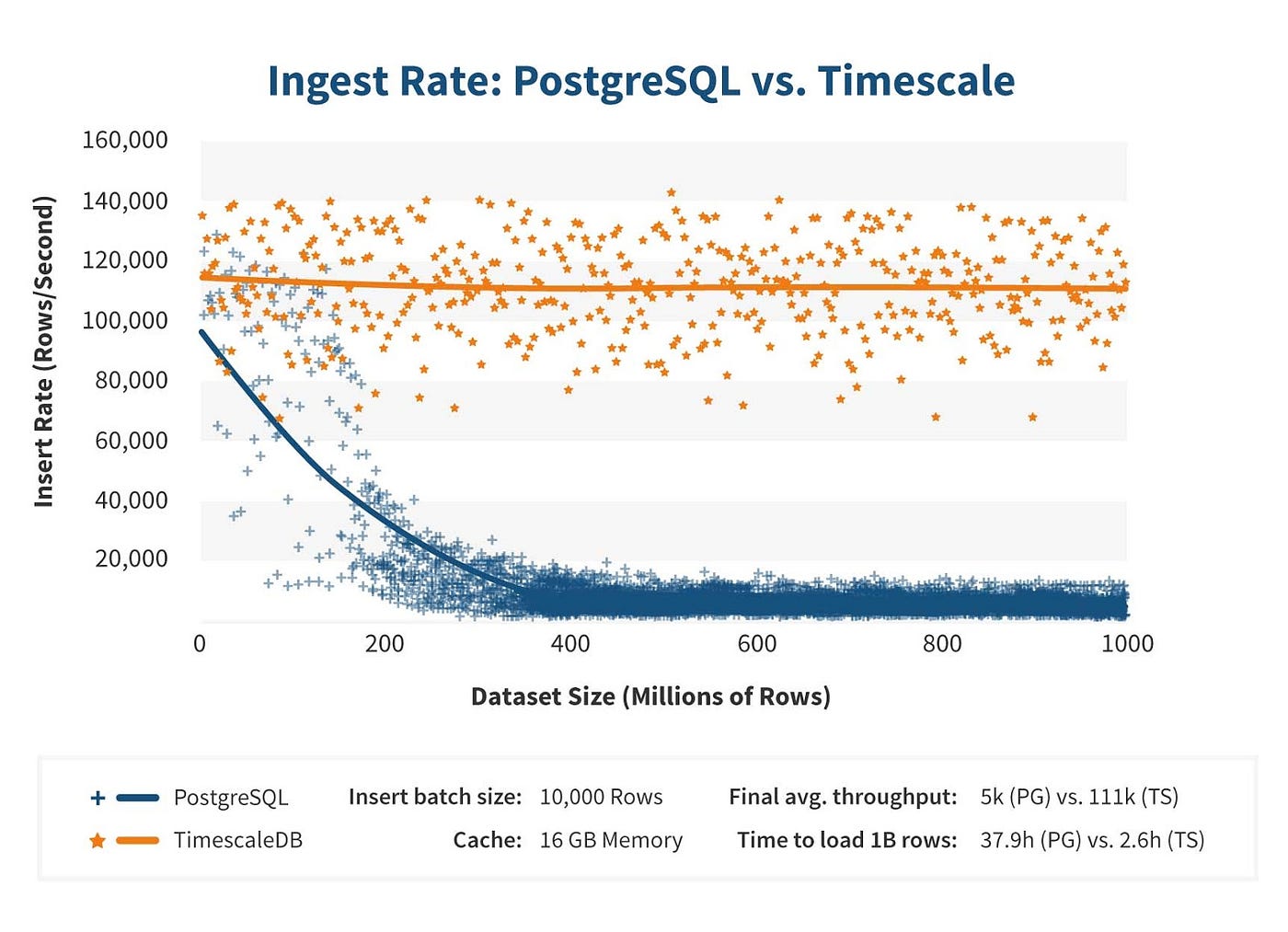


Слика 2 Пример релације две табеле[[2]](#footnote-2)

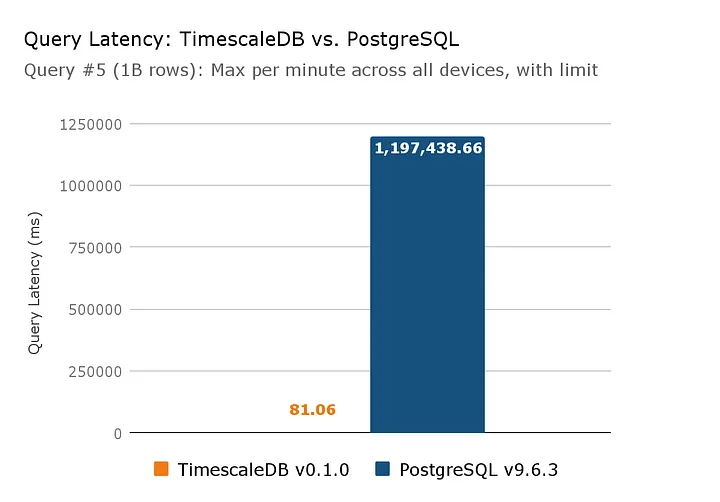
## TimescaleDB

**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

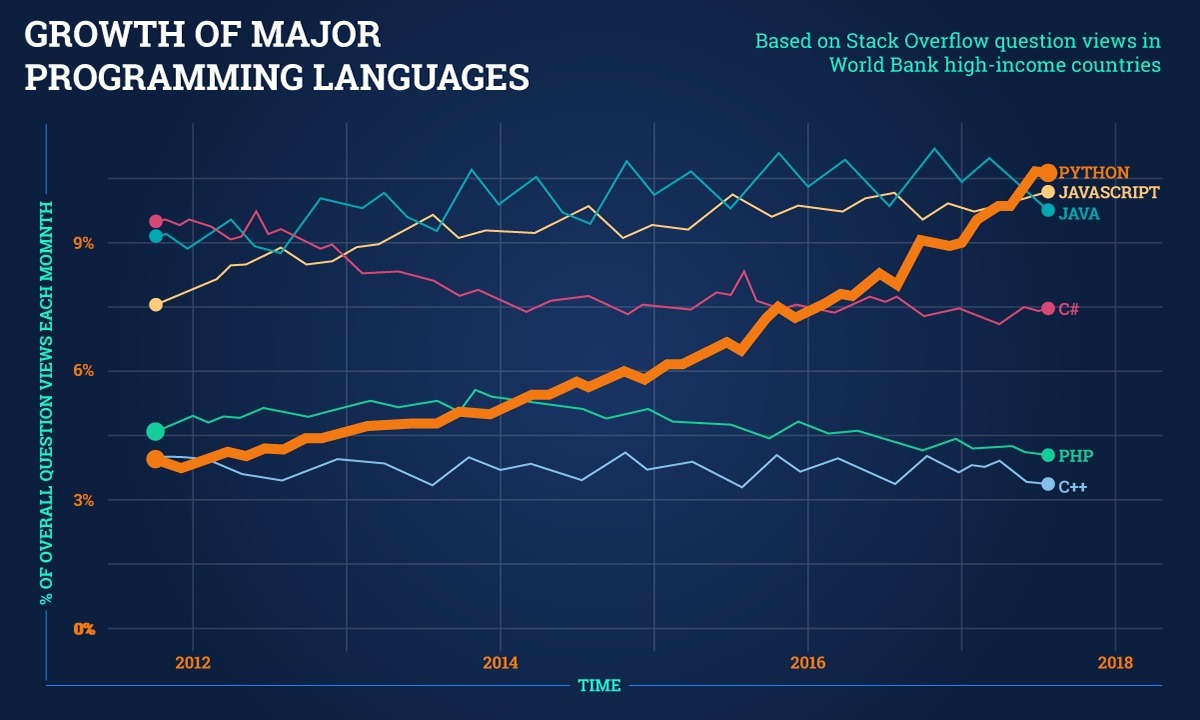
**TimescaleDB** нуди напредне функције за управљање временским серијама, укључујући **агрегацију података по временским интервалима**, **израду временских прозора** и **претраживање података у реалном времену**. Ове функције су интегрисане у **PostgreSQL-ову** SQL синтаксу, што омогућава корисницима 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.

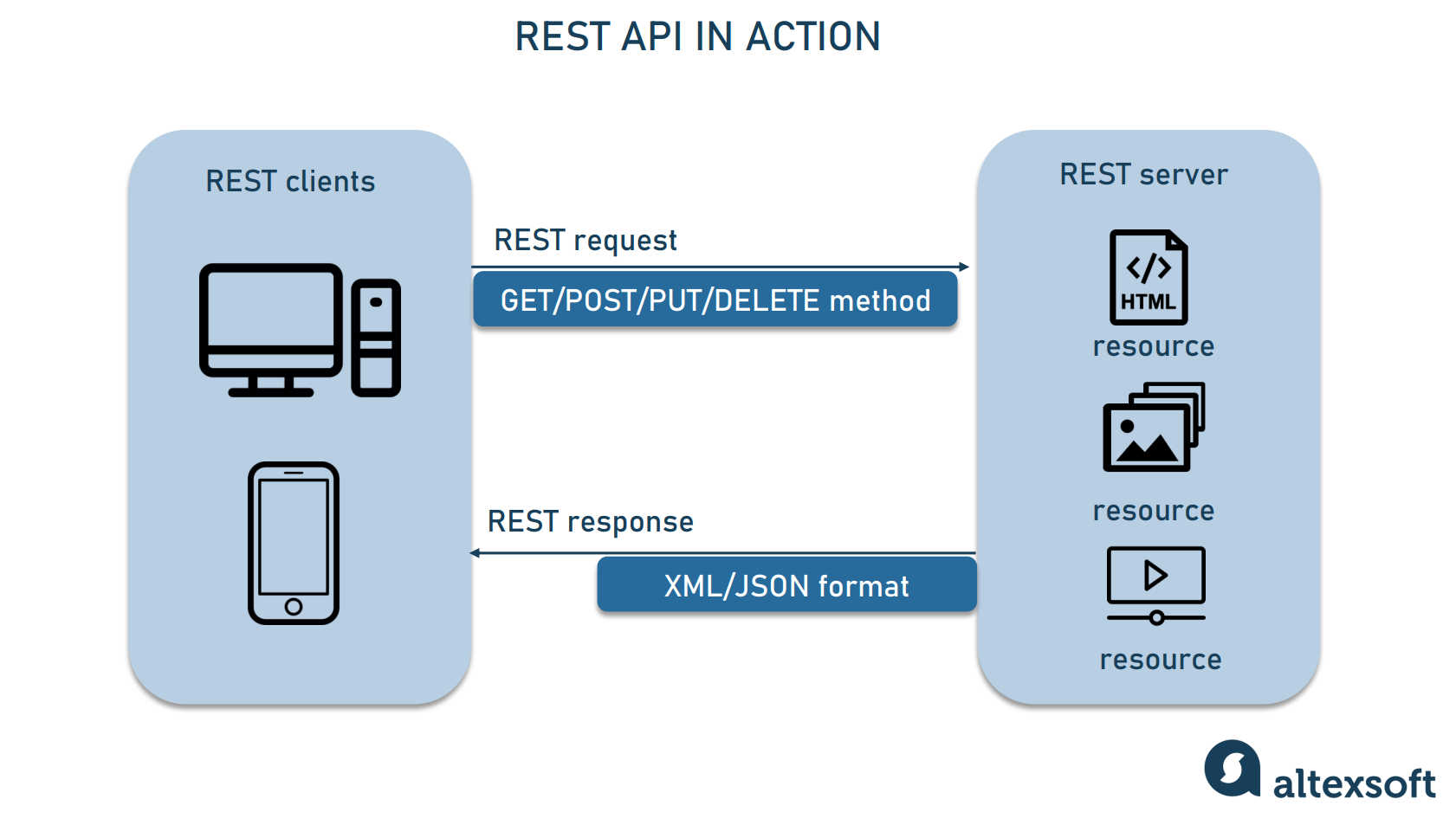
Са подршком за разне **парадигме програмирања**, укључујући **објектно-оријентисано** и **функционално програмирање**, **Python** омогућава програмерима да изаберу најприкладнији приступ за решавање конкретних проблема. Његова једноставност и моћни алати чине га идеалним избором за развој софтвера и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

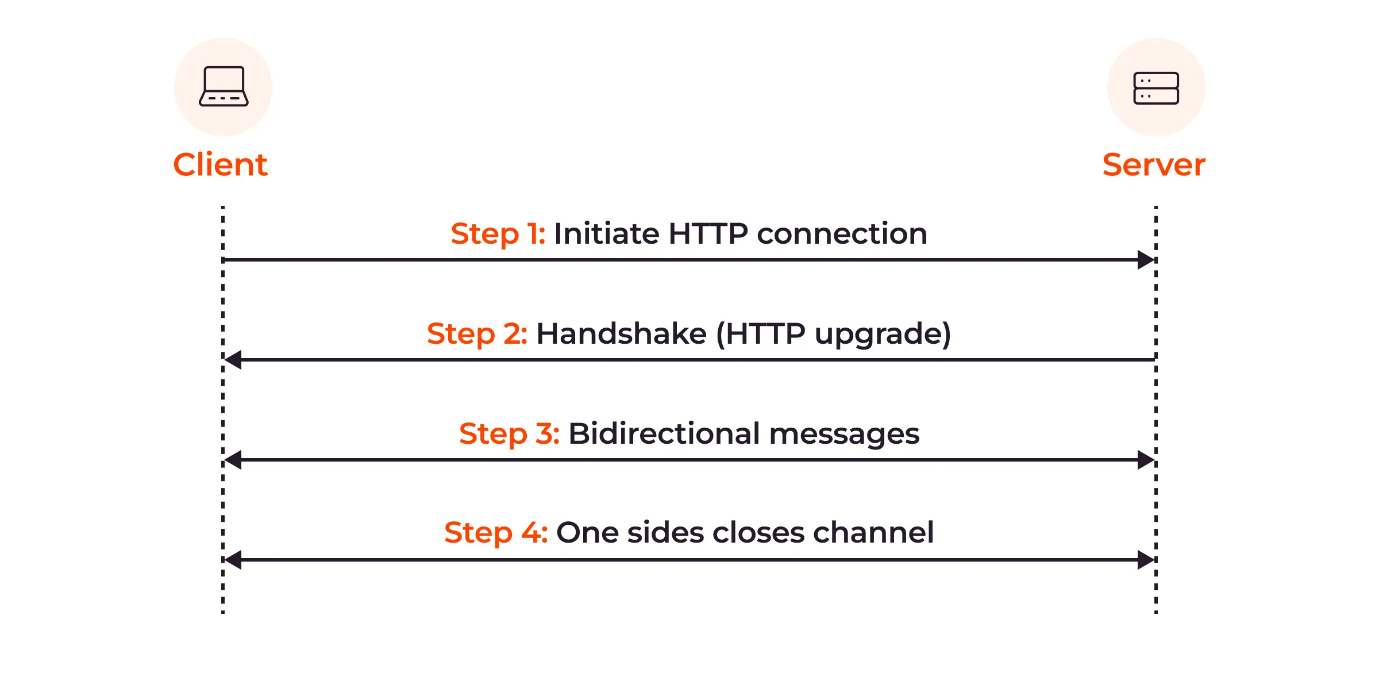
* **Безбедност**: Једна од главних предности је сигурност. Користећи стандардне криптографске алгоритме, JWT осигурава да су подаци унутар токена аутентични и непромењени. Потписивање токена обезбеђује да се подаци не могу модификовати без откривања.
* **Рок трајања и обнављање**: Подешавање истека токена 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.

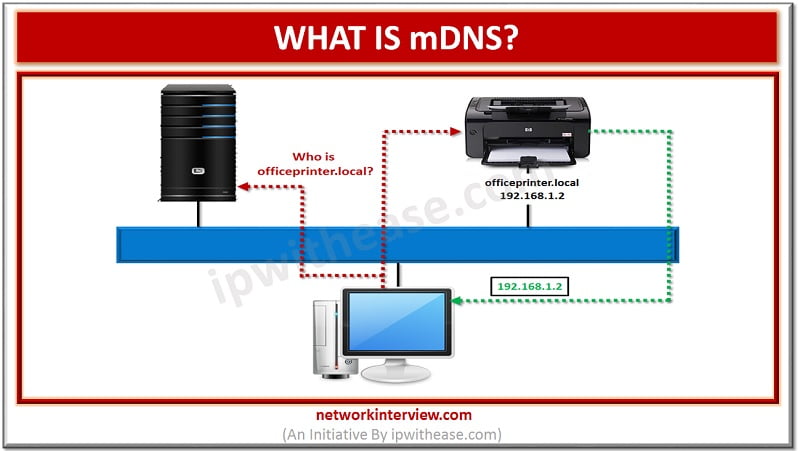
### mDNS (Multicast **DNS**)

**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 у исечку кода 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

Исечак кода 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, исечци 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  
 }

Исечак кода 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()

Исечак кода 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 исечку 4.

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

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

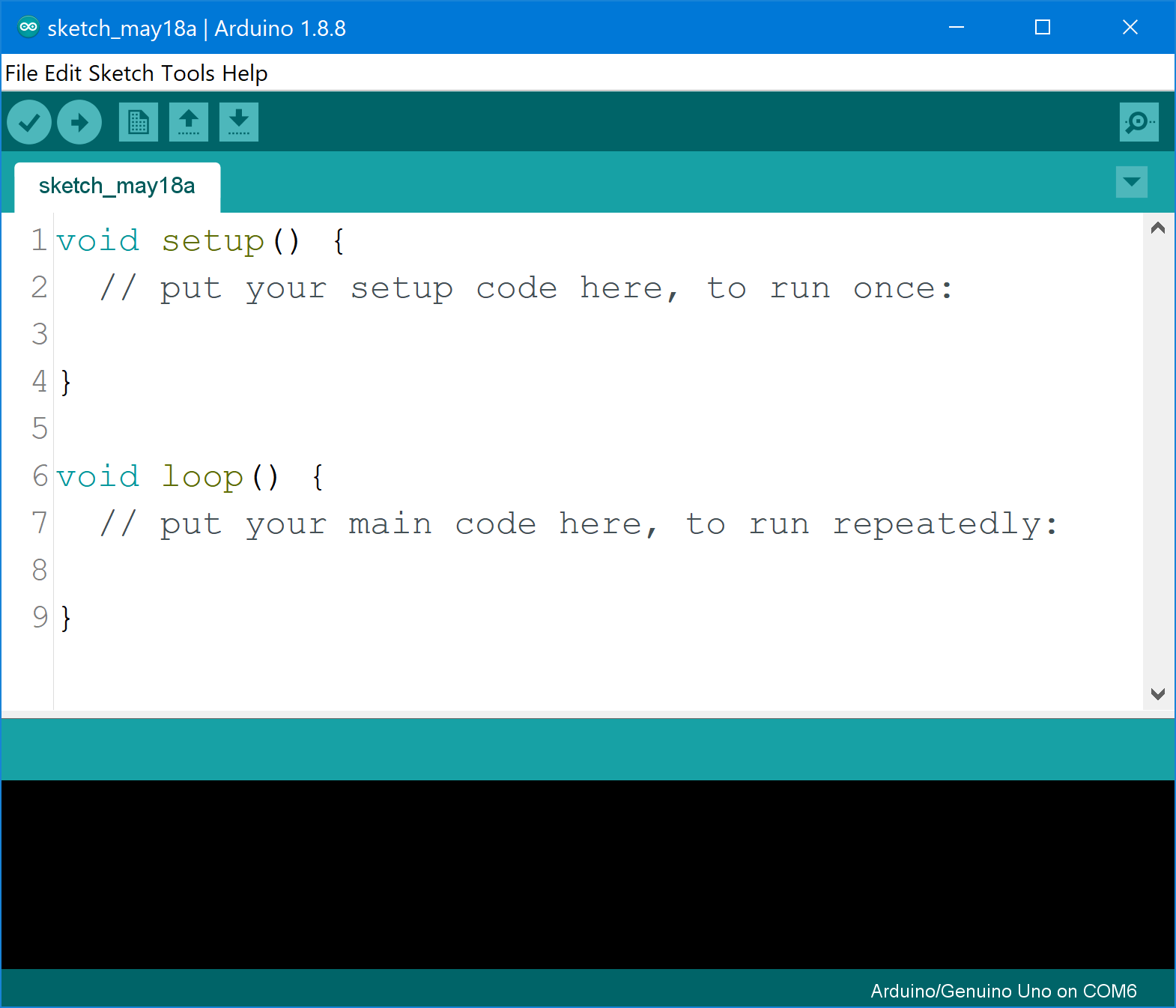
## Arduino IDE

**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)

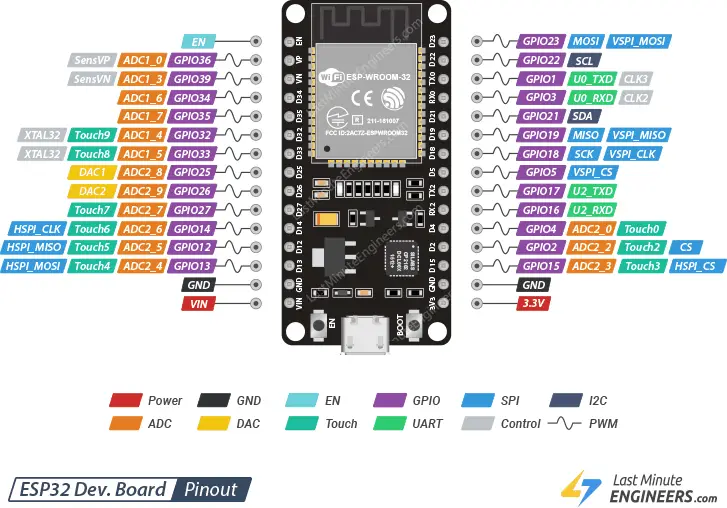
Arduino IDE је основни алат за развој и имплементацију пројеката на Arduino платформи, чинећи га важним делом за све кориснике Arduino хардвера.

## 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

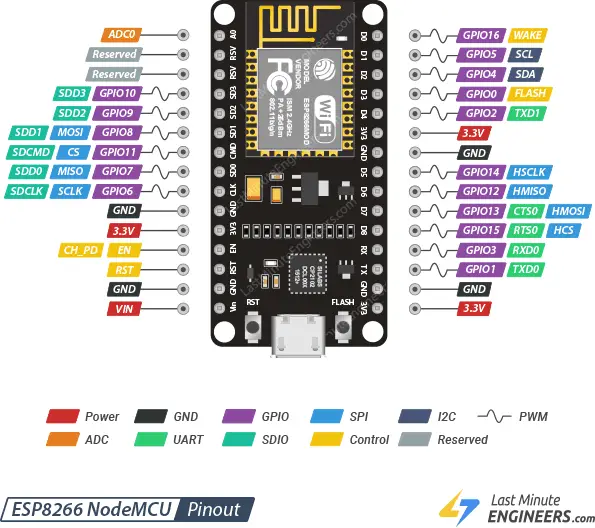
**ESP32** је напредни микроконтролер чип који подржава **Wi-Fi** и **Bluetooth** (Classic и BLE). Са **двојезграним процесором** и великим бројем **I/O пинова**, ESP32 пружа велику флексибилност у развоју пројеката. Његове карактеристике укључују интегрисани Wi-Fi модул, **Bluetooth комуникацију**, и 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

### 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.



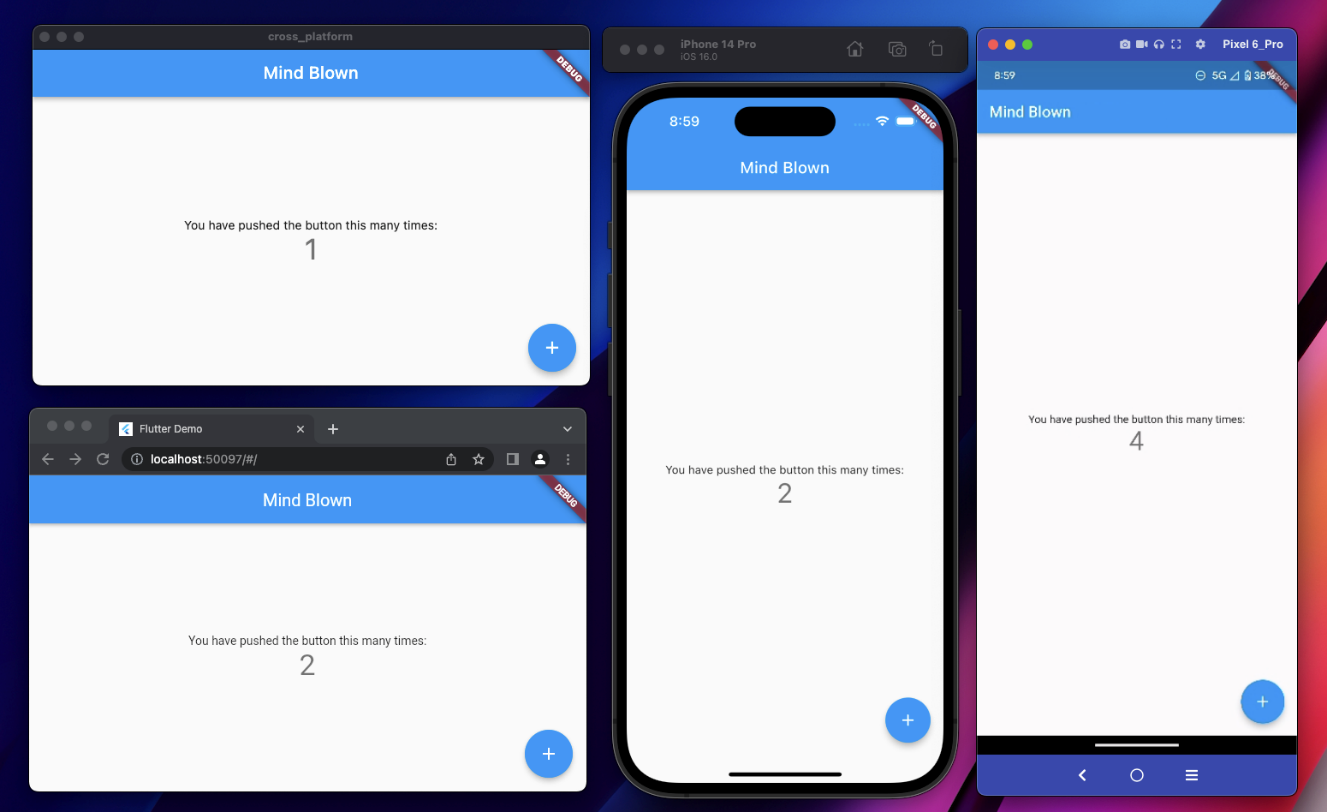
Слика 12 Приказ ESP8226 заједно са његовим пиновима као и подржаним протоколима за сваки пин

## 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**.



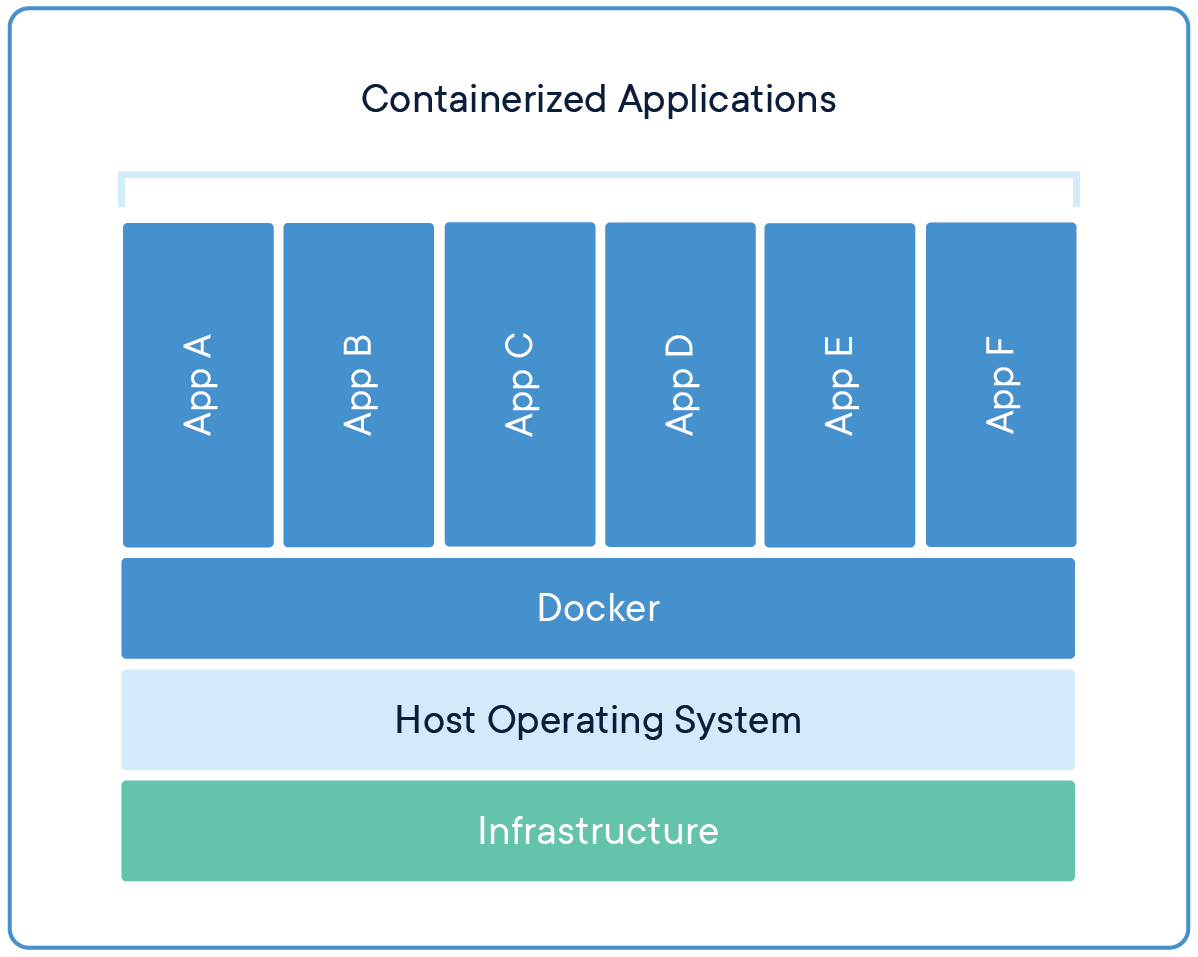
Слика 13 Flutter комајлован за iOS, Android i Web[[10]](#footnote-10)

## 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[[11]](#footnote-11)

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.

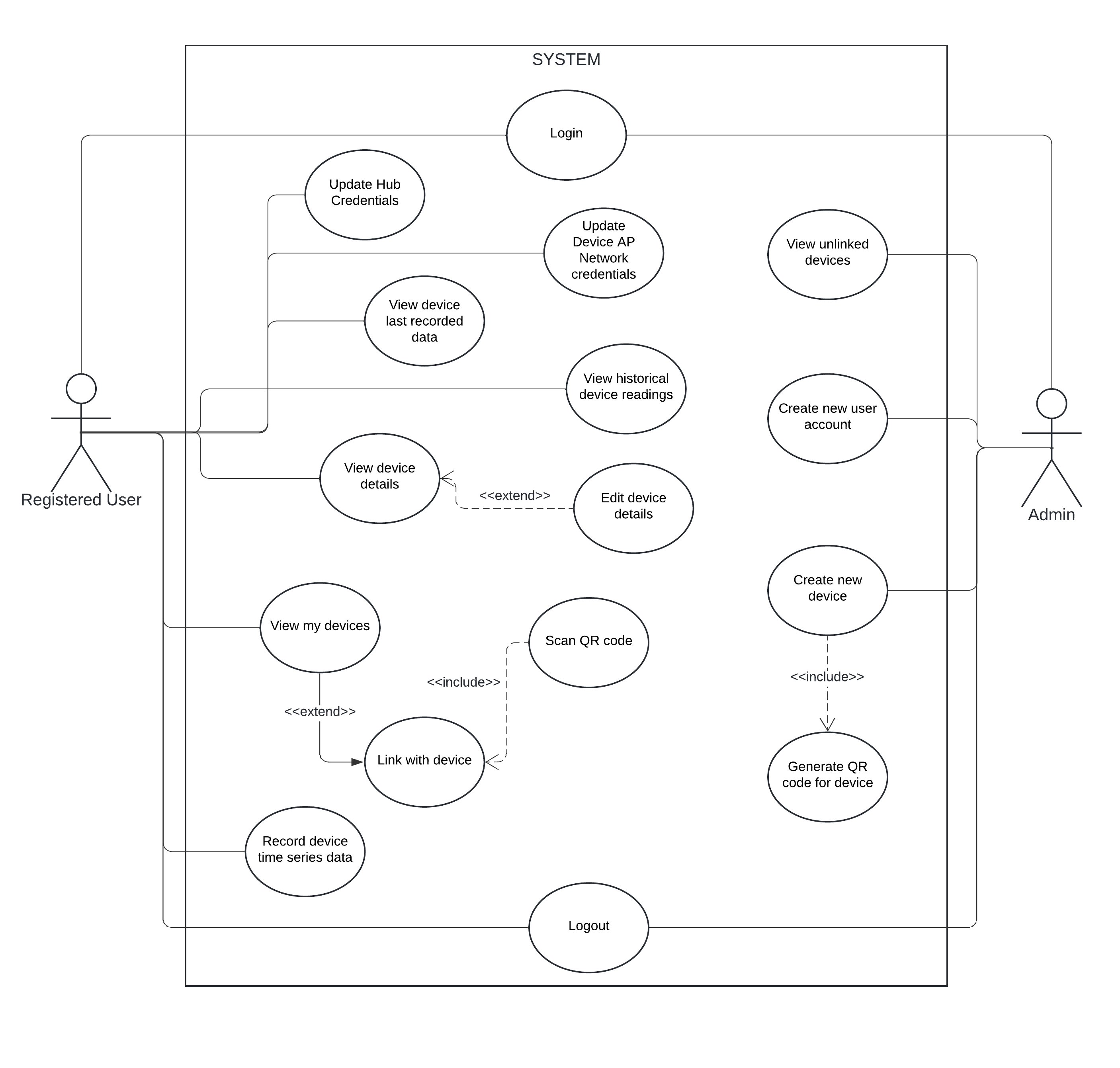
## Спецификација захтева

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.

### Модел података

### Архитектура система

# Имплементација система

Сети се хипертабеле код timescaleDB

# Закључак

Асд

# Литература

Асд

# Биографија

Асд

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 [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-10)
11. Слика преузета са [What is a Container? | Docker](https://www.docker.com/resources/what-container/) [↑](#footnote-ref-11)