УНИВЕРЗИТЕТ У НИШУ

ЕЛЕКТРОНСКИ ФАКУЛТЕТ

Микросервисна архитектура и имплементација софтверског система паметне куће

Завршни рад

Студијски програм: Електротехника и рачунарство

Модул: Рачунарство и информатика

Студент:

Василије Томовић, бр. индекса 18450

Ментор:

Проф. др Драган Х. Стојановић

Ниш, 2025. године

Универзитет у Нишу
Електронски факултет

Микросервисна архитектура и имплементација софтверског система паметне куће

Завршни рад
Студијски програм: Електротехника и рачунарство
Модул: Рачунарство и информатика
Студент: Василије Томовић, бр. индекса 18450
Ментор: Проф. др Драган Х. Стојановић
Задатак: Проучити технологије, методе, софтверске алате и платформе за развој микосервисних архитектура, апликација и система. Развити микросервисну апликацију за обраду и анализу података у паметној кући, детектовање догађаја од интереса и и покретање одговарајућих акција.

Датум пријаве рада: 21.11.2025. г.
Датум предаје рада: 02.12.2025. г.
Датум одбране рада: 04.12.2025. г.

Комисија за оцену и одбрану:

_____

1. Проф. др Драган Х. Стојановић, председник комисије

_____

2. Проф. др Милорад Б. Тошић

_____

3. Проф. др Наталија М. Стојановић

Микросервисна архитектура и имплементација софтверског система паметне куће

АПСТРАКТ

Савремени системи паметних кућа све чешће обухватају велики број сензора и актуатора који раде у реалном времену, што намеће потребу за флексибилним, скалабилним и лако проширивим софтверским решењима. У овом раду представљен је MySweetHome - микросервисни IoT систем за паметну кућу, који омогућава праћење услова у просторији и аутоматизовано управљање уређајима на основу измерених вредности и безбедносних догађаја.

Систем се састоји од Arduino уређаја са MKR IoT Carrier модулом, који мери температуру, влажност и притисак, детектује покрет и омогућава управљање алармом уз PIN аутентикацију, као и скупа микросервиса који се извршавају у Docker окружењу. Комуникација између уређаја и backend дела система заснива се на MQTT протоколу, док микросервиси међусобно размењују податке преко NATS брокера порука. SensorGatewayService врши пријем и агрегирање података, StorageService их трајно смешта у MongoDB базу и излаже HTTP API за визуелизацију у Grafana окружењу, док CommandService на основу правила и догађаја генерише команде за актуаторе. За аналитику на edge-у и обраду stream-ова података коришћен је EKuiper.

Резултати рада показују да микросервисна архитектура добро одговара IoT сценарију паметне куће, јер омогућава независан развој и распоређивање компоненти, лако додавање нових функционалности (нових сензора, сервиса или правила), као и јасно раздвајање одговорности између слоја уређаја, обраде података и аутоматизације.

Microservice architecture and implementation od a smart home software system

ABSTRACT

Modern smart home systems increasingly integrate a large number of sensors and actuators operating in real time, which creates the need for flexible, scalable and easily extensible software solutions. This thesis presents MySweetHome - a microservice-based IoT system for smart homes that enables monitoring of room conditions and automated control of devices based on measured values and security events.

The system consists of an Arduino device with an MKR IoT Carrier module, which measures temperature, humidity and pressure, detects motion, and provides alarm control with PIN-based authentication, as well as a set of microservices running in a Docker environment. Communication between the device and the backend part of the system is based on the MQTT protocol, while the microservices exchange data via the NATS message broker. The SensorGatewayService handles data ingestion and aggregation, the StorageService persistently stores data in a MongoDB database and exposes an HTTP API for visualization in Grafana, while the CommandService generates actuator commands based on rules and events. EKuiper is used for edge analytics and stream processing.

The results demonstrate that the microservice architecture is well suited for smart home IoT scenarios, as it enables independent development and deployment of components, easy addition of new functionalities (sensors, services, or rules), and a clear separation of concerns between the device layer, data processing layer, and automation layer.

САДРЖАЈ

1. Увод

Савремена домаћинства садрже све већи број „паметних“ уређаја (smart-home devices) - сензоре за температуру, влажност и притисак, паметне термостате, расвету, камере и алармне системе. Ови уређаји су често међусобно изоловани или везани за затворене (proprietary) платформе, па корисник нема јединствен преглед стања нити централно место за управљање. Циљ овог рада је да прикаже како се, коришћењем микросервисне архитектуре, може изградити флексибилан систем који прикупља податке са више сензора, доноси одлуке и управља актуаторима, уз могућност лаког проширења функционалности и прилагођавања конкретним потребама домаћинства.

Микросервисна архитектура (microservices architecture) подразумева да се систем састоји од више малих, слабо повезаних сервиса, од којих сваки има јасно дефинисану улогу и може се независно развијати, распоређивати и скалирати. Такав приступ је посебно погодан за Интернет ствари (Internet of Things), у даљем тексту IoT, где велики број уређаја производи континуиране токове (stream) података које је потребно поуздано транспортовати, обрадити и сачувати. Уместо једне монолитне апликације, логика за пријем, обраду, складиштење и управљање актуаторима распоређује се у више специјализованих сервиса који комуницирају преко брокера порука.

Мотивишући сценарио рада је систем паметне куће MySweetHome. У стану се налази IoT уређај заснован на Arduino платформи са MKR IoT Carrier модулом, који мери температуру, влажност и притисак, детектује покрет и омогућава управљање алармом. Подаци се преко Message Queuing Telemetry Transport (MQTT) брокера шаљу ка микросервисима у позадини, који даље користе NATS као брокер порука за међусобну комуникацију. На основу измерених вредности и безбедносних догађаја (детекција уљеза, успешно или неуспешно уношење PIN кода), систем аутоматски активира или деактивира актуаторе као што су клима-уређај, овлаживач ваздуха или вентилатор. На овај начин MySweetHome конкретно илуструје примену микросервисне архитектуре у IoT сценарију паметне куће.

У поглављу 2 даје се преглед микросервисне архитектуре: дефинишу се микросервиси и њихове карактеристике, пореди се микросервисни и монолитни приступ те разматрају предности, недостаци и типичне технологије, са освртом на примену у IoT системима. Поглавље 3 представља IoT микросервисне системе и апликације - дефинише се IoT, описује архитектура и компоненте система, разматра улога брокера порука (MQTT и NATS) и дају се

примери постојећих IoT микросервисних платформи. У поглављу 4 детаљно се описује систем MySweetHome: захтеви, архитектура, модел података и имплементација кључних компоненти система, укључујући микросервисе и оркестрацију помоћу Docker-а. Поглавље 5 приказује функционалности, тест сценарије и евалуацију решења, са примерима рада аларма, нормалног режима и аутоматизације актуатора, уз илустрације помоћу слика екрана.

2. Микросервиси и микросервисна архитектура

Микросервиси су стил архитектуре у ком се сложен систем гради као скуп мањих, аутономних сервиса који комуницирају преко добро дефинисаних интерфејса [1]. Уместо једне велике апликације, логика је подељена на више засебних целина (нпр. сервис за кориснике, сервис за наруџбине, сервис за плаћања), које се могу независно развијати, распоређивати и скалирати [3].

2.1. Микросервиси - дефиниција и основне карактеристике

Sam Newman микросервис описује као мали сервис, фокусиран на једну доменску функцију, који поседује сопствене податке и комуницира са другим сервисима преко лаганих механизама , обично Hypertext Transfer Protocol (HTTP) или брокер порука [1].

Основне карактеристике су [1][3]:

- Мала величина и јасна одговорност - сваки сервис реализује једну уско дефинисану функцију домена (нпр. корисници, рачуноводство, обрада команди);
- Аутономија и независно распоређивање - сваки сервис се може посебно изградити, тестирати и развити (deploy), без утицаја на остатак система;
- Сопствени модел података - сервиси обично имају сопствену базу или шему (database per service), чиме се избегава јака повезаност преко заједничке базе;
- Лабаво повезивање (loose coupling) - комуникација се своди на јавне интерфејсе (API, поруке), без директног позивања унутрашњих класа и табела других сервиса.

2.2. Микросервисна vs монолитна архитектура

У монолитној архитектури све функционалности апликације налазе се у једном извршном пакету (процесу) - један код, једна deploy јединица, једна база података. Овај приступ је једноставнији за старт: лакше је поставити развојно окружење, имати једну апликацију и један deployment канал. Међутим, како систем расте, монолит постаје тешко одржив: мале измене захтевају поновно распоређивање целе апликације, скалирање се врши на нивоу целог монолита, а различите доменске целине су јако повезане.

Код микросервисне архитектуре систем је подељен на више сервиса. Предности у односу на монолит су [1][4]:

- финије скалирање - скалира се само онај сервис који је оптерећен;

- бржи развој и независан рад тимова - различити тимови могу радити на различитим сервисима и бирати технологије које им одговарају;
- боља отпорност - пад једног сервиса не мора да обори цео систем (ако су откази адекватно изоловани).

С друге стране, микросервисни приступ уводи нову сложеност: дистрибуиране трансакције, надзор (observability), верзионисање Application programming interface-a (API) и управљање конфигурацијом за више сервиса. Због тога се микросервиси обично примењују када систем прерасте једноставан монолит и када постоји јасна добит од независног скалирања и развоја [1].

2.3. Технологије за имплементацију микросервиса

Микросервиси се могу развијати у различитим програмским језицима и оквирима - чест избор су Java / Spring Boot, .NET / ASP.NET Core, Node.js, Python (Flask, FastAPI), Go и други. Битан елемент је да сваки сервис има јасно дефинисан интерфејс (REST, gRPC, размена порука) и да се једноставно покреће.

У пракси се микросервиси скоро увек пакују у Docker контејнере, што олакшава изолацију зависности и исти начин покретања у развојном, тест и продукционом окружењу. За оркестрацију више контејнера користе се алати као што су docker-compose (за локални развој и мање системе) и Kubernetes за продукциона cloud-native окружења [1][5].

За комуникацију се, поред HTTP/REST и gRPC, често користе брокери порука (message brokers) као што су Apache Kafka, RabbitMQ, NATS и сл. Они омогућавају асинхрону, поузданију размену порука и боље раздвајање сервиса [2][9]. У контексту IoT-а додају се и протоколи као што је MQTT, који је оптимизован за уређаје са ограниченим ресурсима [8].

2.4. Пројектни обрасци у микросервисима

Микросервисна архитектура уводи типичне проблеме (размена података, конзистентност, откази), па се у литератури и пракси користи низ пројектних образаца (design patterns) [1].

Најчешћи обрасци су [1][6]:

- API Gateway - једна улазна тачка (gateway сервис) која прима захтеве клијената, а затим их прослеђује унутрашњим сервисима. Ово поједностављује клијенте и омогућава централизовано аутентификовање, логовање и сл;

- Database per Service - сваки сервис има сопствену базу/шему, чиме се избегава чврста веза преко заједничких табела и омогућава еволуција модела података по сервису;
- Saga - образац за управљање дистрибуираним пословним процесима који се састоје од више локалних трансакција у различитим сервисима. Уместо глобалне Atomicity, Consistency, Isolation, Durability (ACID) трансакције, користи се секвенца локалних трансакција и компензационих акција;
- Circuit Breaker - штити сервис од вишеструких неуспешних позива према сервису који је у проблему (прекид „кола“ након одређеног броја грешака и прављење паузе пре нових покушаја);
- Command Query Responsibility Segregation (CQRS) и Event Sourcing - раздвајање модела за писање и читање, као и чување стања као низа догађаја, користи се у системима са сложеним доменом и великим бројем уписа.

Ови обрасци нису обавезни, али помажу да се начелни концепт микросервиса претвори у практично одржив систем.

2.5. Предности и недостаци микросервисне архитектуре

Главне предности микросервиса су [1]:

- скалабилност - само најоптерећени сервиси се скалирају;
- агилни развој - тимови могу независно да мењају и деплојују своје сервисе;
- флексибилност технологија - различити сервиси могу користити различите језике и базе, у складу са потребама;
- боља изолација грешака - отказ у једном сервису не мора да обори цео систем, ако је дизајн исправан.

Међутим, постоје и значајни недостаци [1][4]:

- већа оперативна сложеност - уместо једне апликације, треба надгледати, конфигурисати и развити више сервиса, што захтева DevOps алате;
- дистрибуирано дебаговање (debugging) - тешко је пратити проток једног захтева кроз више сервиса без доброг логовања, трасирања (tracing) и метрика;
- конзистентност података - нема једноставних ACID трансакција преко више сервиса, па је неопходно користити евентуалну конзистенцију (eventual consistency) и обрасце као што су Saga;

- прекомерни-инжењеринг (over-engineering) - за мање апликације микросервисни приступ може бити непотребно компликован у односу на једноставан монолит.

Зато се микросервисна архитектура препоручује када систем има довољан ниво сложености и дугорочне захтеве за скалабилност и еволуцију.

2.6. Микросервиси као основа за IoT системе

IoT системи су по природи дистрибуирани и хетерогени - велики број уређаја (сензори, актуатори, edge чворови, cloud сервис) размењују податке преко различитих протокола. Perry Lea наглашава да IoT платформе у пракси често користе микросервисни приступ: посебни сервиси за пријем телеметрије (ingest), складиштење, аналитичку обраду, управљање уређајима и командама, аутентификацију и визуелизацију [2].

Примена микросервиса у IoT-у доноси неколико јасних користи [2][7]:

- раздвајање одговорности - на пример, посебан сервис за gateway (пријем MQTT порука), посебан за складиштење и посебан за обраду и генерисање команди;
- лакше увођење нових типова уређаја - додавањем новог сервиса или проширењем постојећег, без измене целог система;
- скалабилност по компонентама - може се скалирати само ingest сервис када број уређаја порасте, или само аналитички сервис када се додају комплекснији алгоритми;
- комбинација edge и cloud компоненти - део логике може радити ближе уређајима (edge), а део у облаку, уз размену преко брокера порука и API-ја.

Управо такав приступ примењен је и у систему MySweetHome: SensorGatewayService микросервис прима телеметрију са Arduino уређаја, StorageService микросервис чува сирове (raw) и агрегиране (avg) податке, а CommandService микросервис управља актуаторима, чиме се добија јасна подела улога и могућност да се појединачне компоненте развијају и проширују независно.

3. IoT микросервисни системи и апликације

3.1. Дефиниција и архитектура IoT система

IoT представља парадигму у којој су физички уређаји - сензори, актуатори, кућни апарати, индустријске машине и други „паметни“ објекти, повезани на мрежу, могу да размењују податке и да буду даљински надгледани и контролисани [2]. Основна идеја је да се физички свет дигитализује, односно да се стање окружења (температура, влажност, покрет, потрошња енергије итд.) непрекидно мери, шаље ка рачунарским системима, обрађује и користи за доношење аутоматизованих одлука.

Референтна архитектура типичног IoT система може се посматрати кроз следеће слојеве [2][7]:

- слој уређаја обухвата сензоре и актуаторе који прикупљају податке и извршавају команде;
- комуникациони слој обезбеђује пренос података помоћу протокола као што су MQTT, HTTP и слично;
- edge слој (gateway уређаји) врши локалну обраду, филтрирање и агрегацију података;
- cloud или апликативни слој обухвата складиштење података, аналитичке сервисе, интеграцију са спољним системима и корисничке апликације.

Оваква слојевита структура омогућава да се различити делови система независно развијају и скалирају, што је посебно важно када се комбинује са микросервисном архитектуром.

3.2. Технологије за IoT системе

IoT системи користе разноврсне технологије на више нивоа. На нивоу уређаја најчешће се користе микроконтролерске платформе као што су Arduino, ESP32, STM32 и сличне, као и рачунари на једној плочи (single-board computers), као што је Raspberry Pi, који могу да покрећу комплекснији софтвер (нпр. Linux и Docker контејнере) [2].

За повезивање уређаја користе се жичане и бежичне технологије: Ethernet и Wi-Fi у кућним и пословним мрежама, али и специјализоване технологије мале потрошње, као што су Zigbee, Z-Wave, Bluetooth Low Energy и LoRaWAN, које су погодне за батеријски напајане уређаје и велике домете [2].

На апликативном нивоу, размена података се често заснива на моделу објављивања порука на теме (topic) и претплате на те теме (publish/subscribe), уз употребу протокола као што су MQTT и NATS. Поруке се типично представљају у JSON формату, а за синхрону комуникацију између сервиса користе се Representational state transfer (REST) или Google remote procedure call (gRPC) интерфејси. За обраду и складиштење података користе се No Structured Query Language (NoSQL) базе, базе засноване на временским серијама (time-series), као и платформе за обраду података у току (stream processing). У cloud-native окружењима често се користе Docker и Kubernetes за контејнеризацију и оркестрацију микросервиса [2].

3.3. Основне компоненте IoT микросервисног система

Када се IoT систем гради као микросервисни, логика се разлаже на више специјализованих компоненти. На нивоу уређаја налазе се сензори и актуатори који мере физичке величине и извршавају команде. Са њима најчешће комуницира gateway сервис (или више њих), који преко протокола као што је MQTT прима податке, обогаћује их метаподацима (нпр. идентификатор куће, корисника, извора), врши основну валидацију и прослеђује даље ка унутрашњем систему [2].

У позадини се обично налази један или више брокера порука, који чине „кичму“ комуникације између микросервиса. Изнад њих раде сервиси за обраду података, који могу да извршавају агрегације, детектују алармне услове или генеришу команде за актуаторе. Подаци се трајно чувају у специјализованим складиштима (MongoDB, InfluxDB, PostgreSQL и сл.), док сервиси за визуелизацију и API-ји омогућавају даљи приступ подацима, интеграцију са другим системима и приказ резултата у облику табела, графикона и контролних табли.

Оваква подела на више сервиса усклађена је са принципима микросервисне архитектуре: сваки сервис има јасну одговорност (ingest, storage, command, analytics, UI…), може да се засебно развија и скалира, а комуникација се ослања на јасно дефинисане интерфејсе и асинхроне поруке [1][2].

3.4. Брокери порука

Брокер порука представља посредничку компоненту која омогућава publish/subscribe стил комуникације. Уместо да сервиси директно позивају једни друге, они објављују поруке на одређене теме, док се други сервиси претплаћују на те теме и добијају поруке које их интересују.

Овај модел је посебно погодан за IoT, јер стварање података (сензори) и коришћење података (аналитички, командни или сервиси за складиштење) могу да се развијају и скалирају независно. Брокери попут MQTT-а или NATS-а омогућавају ниску латенцију и подршку за велики број истовремених клијената, што доприноси великој флексибилности.

3.4.1. MQTT

MQTT је лаки publish/subscribe протокол дизајниран за уређаје са ограниченим ресурсима и мреже са малим протоком и великом латенцијом (low bandwidth, high latency). Клијенти се повезују на MQTT брокер, а комуникација се организује преко тема, које имају хијерархијску структуру (нпр. home/house_1/sensor/data). Клијент може да објави поруку на тему или да се претплати на једну или више тема уз коришћење маски (wildcard) [8].

MQTT подржава различите нивое квалитета услуге (Quality of Service - QoS), што омогућава да се баланс између поузданости и оптерећења мреже подеси у складу са потребама конкретног система [8]. Због једноставности и компактности, MQTT је постао стандардни протокол за комуникацију IoT уређаја ка облаку.

3.4.2. NATS

NATS је cloud-native систем за размену порука, оријентисан на једноставност, високе перформансе и малу латенцију. Као и MQTT, подржава publish/subscribe модел, али поред тога нуди и затражи/одговори (request/reply), и балансирање порука између више радника (queue groups) [9].

Захваљујући томе што је писан за cloud-native окружења и лако се распоређује у кластерима, NATS је погодан као „унутрашњи“ брокер у микросервисним системима, док MQTT често игра улогу „спољашњег“ протокола између IoT уређаја и backend сервиса. Комбинација MQTT-а и NATS-а омогућава ефикасно повезивање света уређаја и света микросервиса.

3.5. IoT апликације - примери

IoT микросервисне апликације налазе примену у различитим доменима. Паметне куће подразумевају праћење услова у просторијама (температура, влажност, квалитет ваздуха), управљање грејањем и хлађењем, осветљењем и безбедносним системима, уз сценарије аутоматизације (нпр. гашење светла када нема покрета, спуштање ролетни при високој осветљености или активирање аларма при покрету у режиму „одсуства“) [2].

У паметним градовима IoT се користи за надзор јавне расвете, квалитета ваздуха, паркинг система и саобраћаја, док у индустријском IoT-у (IIoT) служи за праћење стања машина, предиктивно одржавање и оптимизацију производних процеса. У пољопривреди IoT системи мере влагу земљишта, температуру и друге параметре како би се аутоматизовало наводњавање и употреба ђубрива. У свим овим примерима, микросервисна архитектура олакшава увођење нових сензора, алгоритама и апликација без нарушавања постојећег система [1][2].

3.6. IoT микросервисне платформе

3.6.1. OpenRemote

OpenRemote je IoT платформа отвореног кода (open-source), намењена изградњи решења за паметне зграде, енергетске системе и паметне градове. Пружа алате за управљање уређајима, дефинисање правила (rule engine), интеграцију са различитим протоколима и израду контролних табли за визуелизацију података. Архитектура платформе је заснована на више сервиса који се типично извршавају у Docker окружењу, што је чини погодном референтном тачком за микросервисни приступ у IoT системима [10].

3.6.2. MainFlux

MainFlux је такође open-source, cloud-native IoT платформа заснована на микросервисима, писана углавном у програмском језику Go. Подржава више протокола (MQTT, HTTP, CoAP, WebSocket), управљање уређајима и идентитетима, као и интеграцију са различитим системима за складиштење података. MainFlux користи брокере порука и контејнеризацију како би омогућио хоризонтално скалирање и флексибилно распоређивање компоненти [11].

3.6.3. Thin-edge.io

Thin-edge.io је лагани edge агент намењен уређајима ограничених ресурса, који обезбеђује стандардизован начин да се уређаји повежу са cloud IoT платформама. Подржава конверзију између протокола (нпр. MQTT ↔ cloud специфични протоколи), локалну обраду и безбедно управљање конфигурацијама [12]. Иако сам по себи није комплетна платформа као MainFlux или OpenRemote, thin-edge.io игра важну улогу у микросервисним IoT архитектурама као edge компонента која повезује уређаје и виши слој система.

3.6.4. SiteWhere

SiteWhere je open-source IoT платформа дизајнирана за индустријске и пословне примене, са фокусом на вишекорисничка (multi-tenant) окружења. Омогућава ingest података из више извора, управљање уређајима, складиштење временских серија и интеграцију са big data алатима и аналитичким системима. Архитектура се заснива на микросервисима распоређеним у Kubernetes кластеру, што омогућава скалабилност и поузданост [13].

3.6.5. EdgeX Foundry

EdgeX Foundry је open-source пројекат фондације “LF Edge”, осмишљен као универзална, микросервисно оријентисана платформа за edge рачунарство у IoT-у. Платформа разлаже функционалности на више микросервиса: ingest података са уређаја, нормализацију и филтрирање, rule engine, експорт података ка другим системима, управљање уређајима и безбедносне компоненте. Сви сервиси се типично пакују као Docker контејнери, што омогућава флексибилно распоређивање на различитим edge уређајима и gateway-има [14].

4. MySweetHome - микросервисни систем паметне куће

У овом поглављу описује се пројекат MySweetHome као конкретна реализација микросервисне архитектуре у домену паметних кућа. Систем комбинује IoT уређај заснован на Arduino платформи, брокере порука MQTT и NATS, више микросервиса за пријем, обраду и складиштење података, као и компоненту за управљање актуаторима и визуелизацију података. Дизајн је усаглашен са принципима микросервисне архитектуре и препорукама за IoT и edge системе описаним у литератури.

4.1. Спецификација захтева

4.1.1. Функционални захтеви

Систем MySweetHome реализује следеће главне функционалне захтеве:

- Мерење параметара окружења - уређај у стану континуирано мери температуру, влажност и притисак помоћу сензора на MKR IoT Carrier плочи и периодично шаље измерене вредности ка backend систему;
- Детекција покрета и алармни систем - уграђени Inertial Measurement Unit (IMU) модул користи се за детекцију померања уређаја. На основу покрета и логике стања (ARMED, AUTH_COUNTDOWN, NORMAL, INTRUDER) систем генерише безбедносне догађаје и активира аларм (звук, LED сигнализација);
- Аутентификација корисника PIN кодом - корисник преко тактилних дугмади на IoT Carrier-у уноси PIN код. У зависности од успешног или неуспешног уноса, систем мења стање аларма и бележи одговарајуће догађаје (AUTH_SUCCESS, AUTH_FAILED, AUTH_TIMEOUT, AUTH_MAX_FAILED);
- Слање телеметрије преко MQTT протокола - Arduino уређај шаље сензорске податке и безбедносне догађаје на MQTT брокер у JSON формату, на тематске канале за телеметрију и догађаје (нпр. home/house_1/sensor/data, home/house_1/sensor/events);
- Пријем телеметрије и обрада у gateway микросервису - SensorGatewayService претплаћује се на MQTT теме, нормализује податке (додаје ts_server, метаподатке о сензору и кући), прослеђује их NATS брокеру и примењује прозоре за рачунање просечних вредности (tumbling window) над температуром, влажношћу и притиском;
- Складиштење података у бази - StorageService прима raw, avg и security поруке преко NATS-а и уписује их у MongoDB базу података у одвојене time-series колекције за сирове податке, агрегате и безбедносне догађаје;

- Генерисање и обрада команди за актуаторе - аналитичка компонента (EKuiper) обрађује токове података и на основу дефинисаних правила (SQL над токовима) генерише команде за актуаторе, које се шаљу као MQTT поруке. CommandService прима те поруке, води стање актуатора и прослеђује само стварне промене стања ка Arduino уређају;
- Визуелизација података и догађаја - StorageService нуди HTTP API над подацима у MongoDB-у, а Grafana користи овај API као извор података (data source) за приказ табела и графикона са телеметријом, агрегатима, безбедносним догађајима и командама.

4.1.2. Нефункционални захтеви

Кључни нефункционални захтеви система су:

- Модуларност и проширивост - систем мора да омогући једноставно додавање нових сервиса (нпр. нови analytics сервис или UI сервис) без измена постојећих компоненти. Ово се постиже микросервисном архитектуром и употребом брокера порука;
- Робустност и толеранција на грешке - компоненте треба да буду слабо повезане, пад једног сервиса не сме да обори цео систем. Асинхрона комуникација преко брокера порука доприноси изолацији грешака и постепеном опоравку сервиса;
- Скалабилност - сервиси који обрађују велики број порука треба да могу независно да се скалирају покретањем више контејнера истог сервиса у Docker окружењу;
- Употреба open-source технологија - услов је коришћење технологија отвореног кода (Arduino, Eclipse Mosquitto, NATS, MongoDB, EKuiper, Grafana, Docker), како би решење било транспарентно, преносиво и без лиценцних ограничења;
- Једноставно развојно и извршно окружење - цео систем треба да се покрене на једној развојној машини помоћу docker compose конфигурације, што олакшава тестирање и демонстрацију решења.

4.2. Архитектура микросервисне апликације

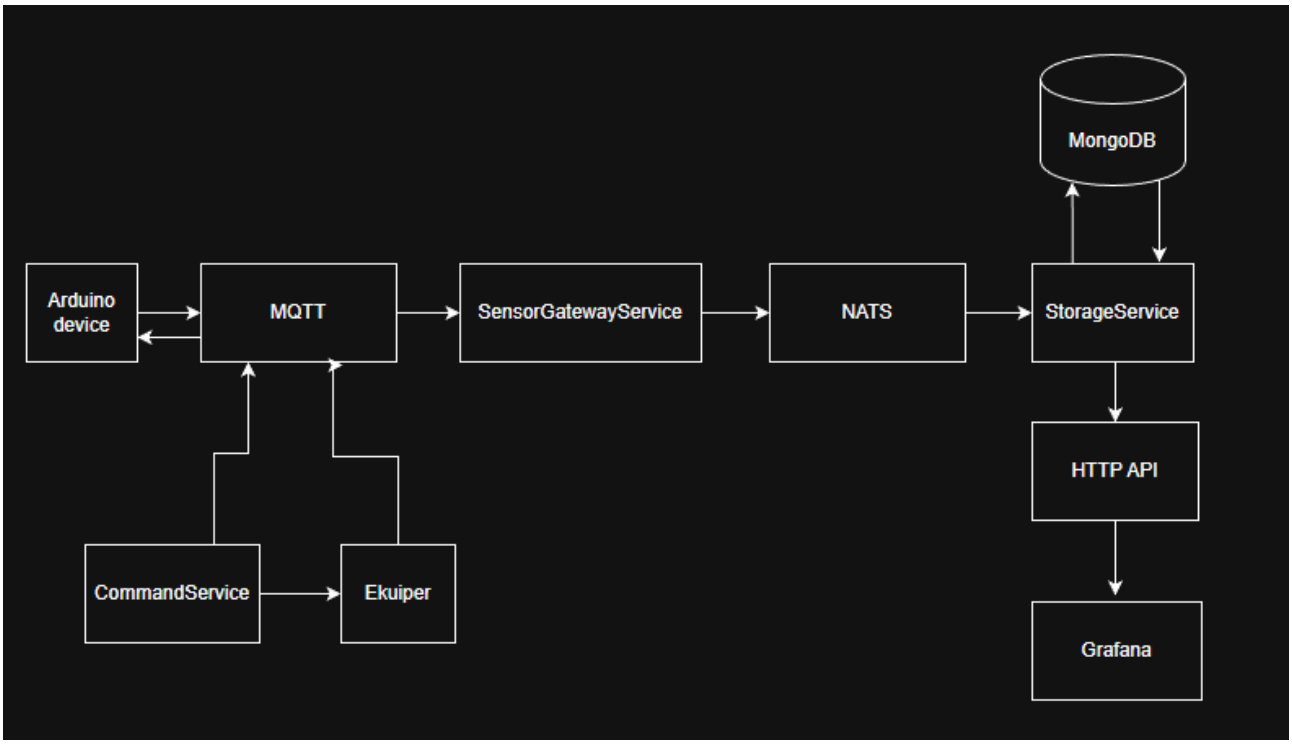
4.2.1. Преглед архитектуре система

Архитектура MySweetHome система састоји се од следећих главних компоненти:

- Arduino уређај са MKR IoT Carrier модулом
- MQTT брокер порука (Eclipse Mosquitto)

- NATS брокер порука
- SensorGatewayService микросервис (Python)
- StorageService микросервис (Node.js + MongoDB)
- CommandService микросервис (.NET)
- EKuiper edge analytics компонента
- Grafana сервис за визуелизацију.

Архитектура је визуелно приказана на слици 4.1.



Слика 4.1. - Архитектура система

4.2.2. Комуникација и размена порука

Комуникација у систему заснива се на два publish/subscribe брокера порука: MQTT и NATS.

- MQTT ниво (између уређаја и backend-а) - Arduino уређај користи MQTT за слање:
  - сензорских података (нпр. тема home/house_1/sensor/data)
  - безбедносних догађаја (нпр. home/house_1/sensor/events)
  - пријем команди за актуаторе преко теме home/house_1/commands/actuators.

На овом нивоу поруке су компактне JSON структуре које садрже идентификатор сензора, куће, корисника (ако постоји), стање аларма и измерене вредности.

- NATS ниво (комуникација између микросервиса) - SensorGatewayService након пријема и нормализације шаље поруке на NATS субјекте (subject):
    - sensor.raw.<sensor_name> за raw податке
    - sensor.avg.<sensor_name> за avg податке
    - sensor.security.<sensor_name> за безбедносне догађаје.

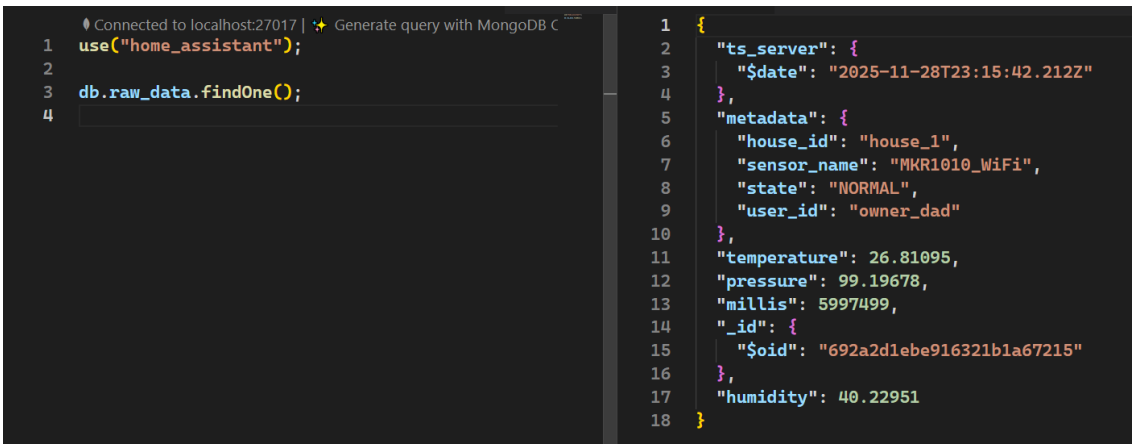
StorageService се претплаћује на шаблоне sensor.raw.>, sensor.avg.> и sensor.security.> и на тај начин добија све поруке за све сензоре. Предност NATS-а је у једноставном моделу субјекта, високој пропусности и могућности лаког додавања нових сервиса који се претплаћују или објављују догађаје.

- Команде и актуатори - EKuiper генерише команде у виду MQTT порука на тему за команде (нпр. home/house_1/commands), које садрже атрибуте као што су device, command, reason, као и сензорске параметре у тренутку доношења одлуке. CommandService те поруке преузима, а након одлуке о промени стања актуатора објављује поруку ка уређају, са пољима device и desired_state. Ова размена показује како се могу комбиновати више сервиса и више нивоа брокера у јединственом IoT микросервисном систему.

4.2.3. Складиштење података и модел података

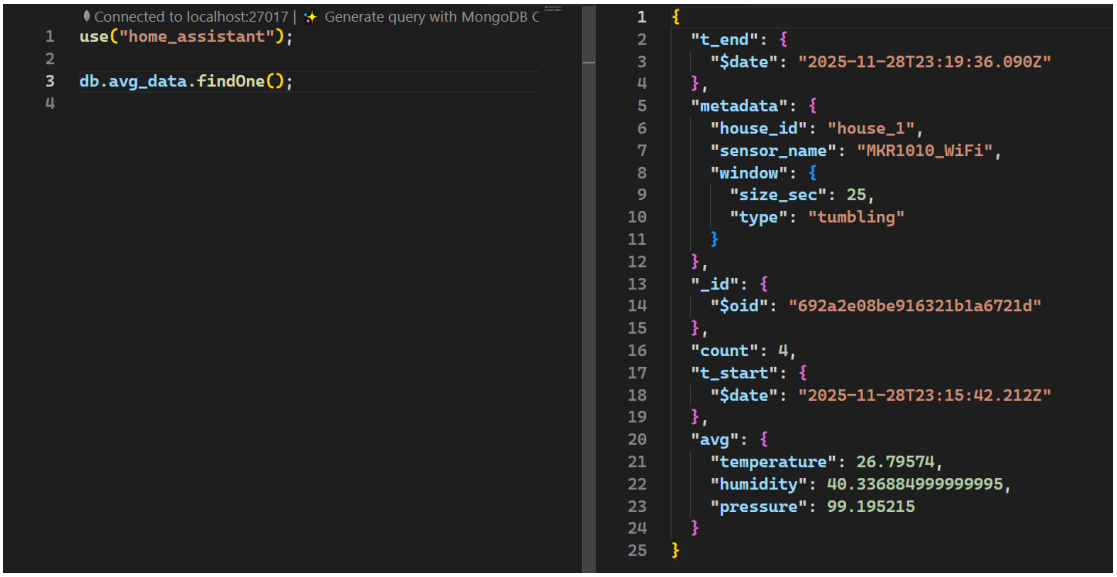
У систему се користи MongoDB база home_assistant са четири главне колекције: raw_data, avg_data, security_events и commands. С обзиром на то да се ради о MongoDB time-series бази података, свака колекција мора да има свој timeField, metaField и подешен granularity који интерно омогућава бази да прави скуп (chunk) података и убрзава касније процес претраживања тих података.

- raw_data - чува сирове телеметријске податке које SensorGatewayService прослеђује преко NATS-а. Сваки документ садржи временски жиг (timestamp), који представља време пријема поруке на backend-у, објекат metadata са пољима sensor_name, house_id, user_id и state, као и мерене вредности temperature, humidity, pressure и millis са уређаја. Структура документа ове колекције илустрована је на слици 4.2;



Слика 4.2. – Модел података у колекцији raw_data

- avg_data - садржи агрегиране (просечне) податке за временске прозоре које израчунава SensorGatewayService. Документ садржи поља t_start и t_end (почетак и крај прозора), објекат metadata са sensor_name, house_id и window (тип прозора и трајање у секундама), под-објекат avg са temperature, humidity и pressure, као и поље count које означава број raw мерења у том прозору. Пример документа ове колекције дат је на слици 4.3;

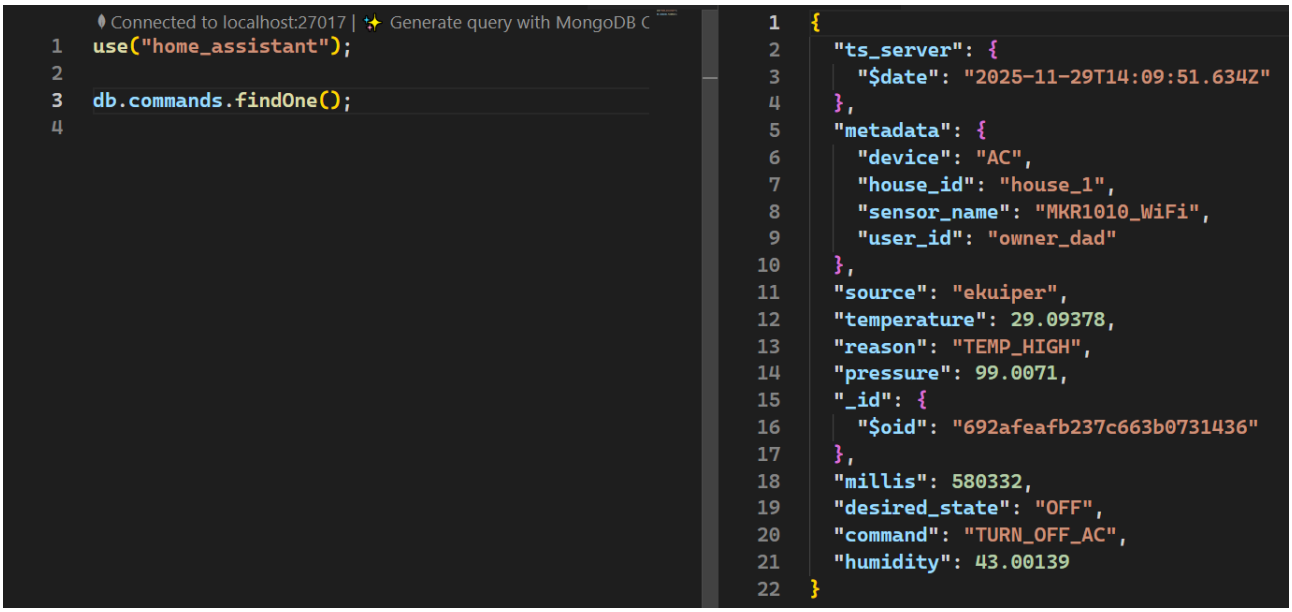


Слика 4.3. – Модел података колекције avg_data

- security_events - бележи све безбедносне догађаје у систему аларма (детекција покрета у режиму ARMED, успешна/неуспешна аутентификација, улазак у INTRUDER стање и друго). Сваки документ садржи timestamp - ts_server, објекат metadata (sensor_name, house_id, user_id), као и поља event_type, prev_state, new_state, reason, failed_attempts и millis. Структура типичног безбедносног догађаја приказана је на слици 4.4;

Слика 4.4. - Модел података колекције security_events

- commands - чува све генерисане команде за актуаторе које производи CommandService (укључивање/искључивање климе, овлаживача, вентилатора и сл.). Документи садрже поље ts_server, објекат metadata са sensor_name, house_id, user_id и device, затим поља command, reason, source, мерене вредности temperature, humidity, pressure, millis, као и поље desired_state које има вредност "ON" или "OFF". Пример документа из колекције commands дат је на слици 4.5.



Слика 4.5. - Модел података колекције commands

Оваква организација података раздваја телеметрију, агрегирана мерења, безбедносне догађаје и команде у посебне колекције, у складу са принципом раздвајања одговорности (separation of concerns) у микросервисним системима. Истовремено, модел омогућава лаку интеграцију са Grafana алатом, који преко HTTP API-ја StorageService-а може да чита податке и приказује их у облику табела и графикона.

4.3. Имплементација система MySweetHome

4.3.1. Arduino уређај

Arduino скица (sketch) реализује више функционалности у једном физичком уређају:

- Мерење параметара окружења помоћу сензора на MKR IoT Carrier плочи (температура, влажност, притисак), као и читање података са IMU модула ради детекције покрета. Код који приказује мерење вредности у одређеном временском интервалу и објављивање на MQTT topic приказан је на слици 4.6;

```
if (now - lastEnvSampleTime >= ENV_SAMPLE_INTERVAL) {
  lastEnvSampleTime = now;

  lastTemp  = carrier.Env.readTemperature();
  lastHum   = carrier.Env.readHumidity();
  lastPress = carrier.Pressure.readPressure();

  publishSensorData(lastTemp, lastHum, lastPress);
```

Слика 4.6. – Очитавање вредности са сензора и слање на MQTT topic

- Управљање стањима аларма: ARMED, AUTH_COUNTDOWN, NORMAL и INTRUDER. За свако стање постоје посебне функције за приказ на екрану, контролу LED диода и понашање бузера (нпр. трептање и сирена у INTRUDER стању). Главна петља уређаја је приказана на слици 4.7;

```
void loop() {
  mqttClient.poll();

  if (actuatorOverlayActive) {
    updateActuatorOverlay();
    delay(20);
    return;
  }

  switch (alarmState) {
    case ARMED:
      if (detectMovement()) {
        publishEventMotionDetectedArmed();
        startAuthCountdown();
      } else {
        delay(20);
      }
      break;

    case AUTH_COUNTDOWN:
      updateAuthCountdown();
      break;

    case NORMAL:
      handleNormalState();
      break;

    case INTRUDER:
      updateIntruderAlarm();
      break;
  }
}
```

Слика 4.7. – Главна петља Arduino уређаја

- PIN аутентификација: у AUTH_COUNTDOWN стању корисник преко капацитивних touch дугмади уноси PIN код; систем води број неуспелих покушаја, шаље одговарајуће MQTT догађаје и прелази у NORMAL или INTRUDER стање. Код који омогућава овакво извршење дат је на слици 4.8;

```
if (enteredLength == PIN_LENGTH) {
  int8_t userIdx = findUserIndexForPin(enteredPin);

  if (userIdx >= 0) {
    currentUserIndex = userIdx;
    currentUserId = USER_IDS[userIdx];

    Serial.print("PIN tacan -> NORMAL stanje, korisnik: ");
    Serial.print("index=");
    Serial.print(currentUserIndex);
    Serial.print(", id=");
    Serial.println(currentUserId);

    publishEventAuthSuccess();

    alarmState = NORMAL;
    normalStateJustEntered = true;
    carrier.Buzzer.noSound();
    playMelody(successMelody, successDurations, SUCCESS_MELODY_LEN);

    showWelcomeScreen();
  } else {
    Serial.println("PIN pogresan!");
    failedAttempts++;

    publishEventAuthFailed();

    playMelody(errorMelody, errorDurations, ERROR_MELODY_LEN);

    Serial.print("Broj neuspesnih pokusaja: ");
    Serial.println(failedAttempts);

    if (failedAttempts >= MAX_FAILED_ATTEMPTS) {
      Serial.println("Previse pogresnih pokusaja -> INTRUDER!");
```

Слика 4.8. – Аутентификација корисника уз помоћ PIN кода

- Мрежна комуникација преко Wi-Fi и MQTT-а: у setup() се успоставља Wi-Fi конекција, затим MQTT конекција ка брокеру, подешава се MQTT client ID и претплата на тему за актуаторе; у loop() се редовно позива mqttClient.poll() ради одржавања конекције;
- Објава телеметрије и догађаја: посебне функције граде JavaScript Object Notation (JSON) документе и шаљу их на MQTT topic-е за податке и догађаје, као што је приказано на слици 4.9.

```
void publishSensorData(float temperature, float humidity, float pressure) {
  StaticJsonDocument<256> doc;

  doc["sensor_name"] = SENSOR_NAME;
  doc["house_id"] = HOUSE_ID;

  if (currentUserId != NULL) {
    doc["user_id"] = currentUserId;
  } else {
    doc["user_id"] = nullptr; // JSON null
  }

  doc["state"] = "NORMAL";
  doc["temperature"] = temperature;
  doc["humidity"] = humidity;
  doc["pressure"] = pressure;
  doc["millis"] = millis();

  publishJsonToMqtt(MQTT_TOPIC_DATA, doc);
}
```

Слика 4.9. – Грађење JSON објекта за слање на MQTT

На тај начин Arduino уређај представља edge чвор који комбинује сензорску, контролно-логичку и комуникациону функционалност, у складу са IoT архитектурама описаним у литератури.

4.3.2. SensorGatewayService

SensorGatewayService је имплементиран у програмском језику Python и састоји се од три главне целине:

- MqttWorker - класа која користи библиотеку paho-mqtt за повезивање на MQTT брокер, претплату на одговарајуће теме и позивање handler функција (on_measure) по пријему сваке JSON поруке (слика 4.10);

```
def _on_message(self, client, userdata, msg):
    try:
        payload = json.loads(msg.payload.decode("utf-8"))

        self.on_measure(payload)

        print(f"[mqtt] Payload from {msg.topic}: {payload}", flush=True)
    except Exception as e:
        print(f"[mqtt] Bad message ({e}). Topic={msg.topic}", flush=True)
```

Слика 4.10. – Рад SensorGatewayService-а са пристиглим порукама на MQTT topic

- NatsPublisher - класа која у посебној асинхроној нити одржава NATS конекцију (nats.aio.client) и прихвата захтеве за објаву порука преко thread-safe реда (queue). Ово раздваја MQTT callback логику од слања порука ка NATS-у и поједностављује руковање асинхроним input/output-ом. Код ове класе се може видети на слици 4.11;

```python
class NatsPublisher:

  def start(self):
    self._thread = threading.Thread(target=self._run_loop, daemon=True)
    self._thread.start()

  def _run_loop(self):
    self._loop = asyncio.new_event_loop()
    asyncio.set_event_loop(self._loop)
    self._loop.run_until_complete(self._async_main())

  async def _async_main(self):
    self._nats_conn = NATS()
    await self._nats_conn.connect(servers=self.servers)
    print("[nats] Connected.", flush=True)

    try:
      while not self._stop_event.is_set():
        try:
          subject, payload = self._queue.get(timeout=0.2)
        except queue.Empty:
          await asyncio.sleep(0.05)
          continue
        data = json.dumps(payload).encode("utf-8")
        await self._nats_conn.publish(subject, data)
    finally:
      await self._nats_conn.drain()
      await self._nats_conn.close()
      print("[nats] Closed.", flush=True)

  def publish(self, subject: str, payload: dict):
    self._queue.put((subject, payload))

  def stop(self):
    self._stop_event.set()
    if self._loop:
      asyncio.run_coroutine_threadsafe(asyncio.sleep(0), self._loop)
      self._thread.join(timeout=3)
```

Слика 4.11. – NatsPublisher класа

- SensorGatewayService - класа која садржи доменску логику:
  - функцију ingest_measure за пријем сирових мерења, обогаћивање метаподацима и слање RAW порука на NATS (слика 4.12);

```python
def ingest_measure(self, measure: dict):
  if not self._running:
    return

  try:
    temp = measure.get("temperature")
    hum = measure.get("humidity")
    press = measure.get("pressure")

    if temp is None or hum is None or press is None:
      print("[gateway] Skipping measure without full env data:", measure, flush=True)
      return

    sensor_name = measure.get("sensor_name", "unknown_sensor")
    house_id = measure.get("house_id", "unknown_house")
    user_id = measure.get("user_id")
    millis = measure.get("millis")

    ts_server = datetime.now(timezone.utc).isoformat()

    raw_payload = {
      "sensor_name": sensor_name,
      "house_id": house_id,
      "user_id": user_id,
      "state": measure.get("state"),
      "temperature": float(temp),
      "humidity": float(hum),
      "pressure": float(press),
      "millis": millis,
      "ts_server": ts_server,
    }

    raw_subject = f"{self.subject_raw_prefix}.{sensor_name}"
    self.nats.publish(raw_subject, raw_payload)
    print(f"[gateway] RAW -> {raw_subject}: {raw_payload}", flush=True)
```

Слика 4.12. – Функција ingest_measure, слање објекта на NATS subject

- одржавање tumbling прозора за рачунање средњих вредности (слика 4.13);

```
now_utc = datetime.now(timezone.utc)

if self._window_started_at is None:
  self._window_started_at = now_utc

elapsed = (now_utc - self._window_started_at).total_seconds()
if elapsed >= self.window_sec and self._bucket:
  self._flush_avg()

self._bucket.append({
  "ts_server": ts_server,
  "sensor_name": sensor_name,
  "house_id": house_id,
  "temperature": float(temp),
  "humidity": float(hum),
  "pressure": float(press),
})
```

Слика 4.13. – Провера да ли је tumbling прозор истекао

- функцију _flush_avg која израчунава просечне вредности и шаље AVG поруку на NATS (слика 4.14);

```
def _flush_avg(self):
  if not self._bucket:
    return

  temps = [m["temperature"] for m in self._bucket]
  hums = [m["humidity"] for m in self._bucket]
  press = [m["pressure"] for m in self._bucket]
  count = len(self._bucket)

  avg_temp = sum(temps) / count if count else None
  avg_hum = sum(hums) / count if count else None
  avg_press = sum(press) / count if count else None

  sensor_name = self._bucket[-1]["sensor_name"]
  house_id = self._bucket[-1]["house_id"]

  t_start = self._window_started_at.isoformat()
  t_end = datetime.now(timezone.utc).isoformat()

  out = {
    "sensor_name": sensor_name,
    "house_id": house_id,
    "window": {
      "type": "tumbling",
      "size_sec": self.window_sec,
    },
    "t_start": t_start,
    "t_end": t_end,
    "avg": {
      "temperature": avg_temp,
      "humidity": avg_hum,
      "pressure": avg_press,
    },
    "count": count,
  }

  subject = f"{self.subject_avg_prefix}.{sensor_name}"
  self.nats.publish(subject, out)
  print(f"[gateway] AVG -> {subject}: {out}", flush=True)

  self._bucket.clear()
  self._window_started_at = datetime.now(timezone.utc)
```

Слика 4.14. - Након истека дефинисаног времена шаље се avg податак на NATS subject

- функцију ingest_security_event која прослеђује безбедносне догађаје као SECURITY поруке (слика 4.15).

```
def ingest_security_event(self, event: dict):
  if not self._running:
    return

  try:
    sensor_name = event.get("sensor_name", "unknown_sensor")
    subject = f"{self.subject_security_prefix}.{sensor_name}"

    payload = dict(event)  # kopija
    payload.setdefault("ts_server", datetime.now(timezone.utc).isoformat())

    self.nats.publish(subject, payload)
    print(f"[gateway] SECURITY -> {subject}: {payload}", flush=True)

  except Exception as e:
    print(f"[gateway] Error in ingest_security_event: {e}. Payload={event}", flush=True)
```

Слика 4.15. – Слање безбедносних догађаја на NATS subject

Главни програм у main.py чита конфигурацију из .env фајла, покреће NatsPublisher, креира инстанцу SensorGatewayService и стартује два MQTT worker-а - један за телеметрију, други за security догађаје - чиме се постиже јасно раздвајање улога и добра тестабилност сервиса. Исечак кода дат је на слици 4.16.

```
nats_pub = NatsPublisher(servers=NATS_SERVER)
nats_pub.start()

gateway = SensorGatewayService(
  window_sec=WINDOW_SEC,
  subject_raw_prefix=SUBJECT_RAW_PREFIX,
  subject_avg_prefix=SUBJECT_AVG_PREFIX,
  subject_security_prefix=SUBJECT_SECURITY_PREFIX,
  nats_publisher=nats_pub
)

mqtt_data = MqttWorker(
  host=MQTT_HOST,
  port=MQTT_PORT,
  topic=MQTT_TOPIC_DATA,
  on_measure=gateway.ingest_measure
)
mqtt_data.start()

mqtt_events = MqttWorker(
  host=MQTT_HOST,
  port=MQTT_PORT,
  topic=MQTT_TOPIC_EVENTS,
  on_measure=gateway.ingest_security_event,
)
mqtt_events.start()
```

Слика 4.16. – Исечак кода из main.py

4.3.3. StorageService

StorageService је Node.js микросервис који повезује NATS и MongoDB и обезбеђује HTTP API за приступ подацима.

- Модул mongo.js креира MongoDB клијента и врши иницијални ping ка бази како би се потврдила повезаност (слика 4.17);

```
import { MongoClient } from "mongodb";

export async function createMongoClient(uri) {
  try {
    const client = new MongoClient(uri, {
      serverSelectionTimeoutMS: 5000,
      connectTimeoutMS: 5000,
    });

    console.log("[mongo] Connecting to MongoDB...");

    await client.connect();
    await client.db("admin").command({ ping: 1 });

    console.log("[mongo] Connected successfully");
    return client;
  } catch (err) {
    console.error("[mongo] Connection FAILED:");
    console.error(err.message);
    process.exit(1);
  }
}
```

Слика 4.17. – Повезивање сервиса са базом

- Модул nats.js обезбеђује функцију createNatsConnection за повезивање на NATS и subscribeNATS за претплату на субјекте, уз асинхроно обрађивање порука у for await петљи (слика 4.18);

```
import { connect as natsConnect } from "nats";

export async function createNatsConnection(servers) {
  const nats_conn = await natsConnect({ servers });
  console.log(`[nats] Connected to ${servers}`);
  return nats_conn;
}

export function subscribeNATS(nats_conn, subject, handler) {
  const sub = nats_conn.subscribe(subject);

  (async () => {
    for await (const m of sub) {
      try {
        const obj = JSON.parse(m.data.toString());
        await handler(obj);
      } catch (err) {
        console.error("[nats] Bad message (not JSON?):", err);
      }
    }
  })();

  console.log(`[nats] Subscribed to ${subject}`);
  return sub;
}
```

Слика 4.18. – Пријављивање сервиса на NATS

- Главни модул index.js чита конфигурацију из .env фајла, успоставља конекције, креира handler-е за упис сирових података у колекцију raw_data, затим упис агрегираних података у колекцију avg_data и упис безбедносних догађаја у колекцију security_events.

Поред тога, StorageService покреће Express веб сервер који нуди REST endpoint-е. Они су неопходни из разлога што Grafana на тај начин долази до података из базе и омогућава њихову визуелизацију креирањем табела и графикона.

Руте ка REST endpoint-има су следеће:

- /api/raw/latest,
- /api/avg/latest,
- /api/security/latest,
- /api/commands/latest,

са параметрима за филтрирање (sensor_name, house_id) и ограничење броја записа (limit). Исечак кода једне руте је приказан на слици 4.19.

```
app.get("/api/raw/latest", async (req, res) => {
  try {
    const limit = parseLimit(req.query.limit, 100);
    const query = buildCommonQuery(req);

    const docs = await rawCol
      .find(query)
      .sort({ ts_server: -1 })
      .limit(limit)
      .toArray();

    res.json(docs);
  } catch (err) {
    console.error("[storage][API][RAW] error:", err);
    res.status(500).json({ error: "Internal server error" });
  }
});
```

Слика 4.19. – HTTP рута за прибављање података из базе

4.3.4. CommandService

CommandService је .NET микросервис који је пријављен на MQTT topic на који Ekuiper објављује команде. Његов задатак је да команде уписује у базу података и прослеђује акцију Arduino уређају у зависности од тога која је команда пристигла.

Модел CommandMessage представља структуру JSON порука које стижу из EKuiper-a. Садржи мерене параметре сензора (температура, влажност, притисак), идентификатор куће и корисника, као и информације о уређају (device), генерисаној команди (command), разлогу (reason) и извору (source) настанка команде. Класа CommandRepository преузима овај модел, конструише одговарајући BsonDocument са пољем metadata (сензор, кућа, корисник, уређај), телеметријским вредностима и пољем desired_state („ON“ или „OFF“), и уписује документ у MongoDB колекцију commands.

Класа MqttCommandWorker је задужена за интеграцију са MQTT брокером. Она успоставља везу ка брокеру, претплаћује се на тему за команде и, по пријему поруке, десеријализује JSON садржај у објекат типа CommandMessage. На основу поља command

одређује се жељено стање актуатора и ажурира се интерни речник тренутних стања по комбинацији (house_id, device), тако да се игноришу команде које не мењају стварно стање уређаја. Само команде које доводе до промене стања уписују се у базу података. То значи да уколико је температура превисока, Ekuiper ће послати команду TURN_AC_OFF, стање ће се у CommandService-у променити, међутим температура неће нагло пасти тако да уколико стигне нова команда TURN_AC_OFF, CommandService ће је игнорисати. За команде које доводе до промене стања се формира и објављује нова MQTT порука ка Arduino уређају, на тему намењену актуаторима, са кључним пољима device и desired_state (слика 4.20).

```
1 reference
private async Task PublishToArduinoAsync(CommandMessage msg, bool desiredOn, CancellationToken ct)
{
    if (_client is null || !_client.IsConnected)
    {
        _logger.LogWarning("[mqtt] Client not connected, cannot publish to Arduino.");
        return;
    }

    var payloadObj = new
    {
        device = msg.Device,
        command = msg.Command,
        desired_state = desiredOn ? "ON" : "OFF",
        reason = msg.Reason,
        house_id = msg.HouseId,
        user_id = msg.UserId,
        ts_server = DateTime.UtcNow
    };

    var json = JsonSerializer.Serialize(payloadObj);
    var message = new MqttApplicationMessageBuilder()
        .WithTopic(_outgoingTopic)
        .WithPayload(json)
        .WithQualityOfServiceLevel(MQTTnet.Protocol.MqttQualityOfServiceLevel.AtMostOnce)
        .Build();

    await _client.PublishAsync(message, ct);
    _logger.LogInformation("[mqtt] Published to Arduino topic {Topic}: {Payload}", _outgoingTopic, json);
}
```

Слика 4.20. – Слање акције на Arduino уређај преко MQTT topic-a

На овај начин CommandService представља спој између аналитике на ивици (edge) - Ekuiper, перзистентног слоја – MongoDB, и извршавања акција на самом уређају.

4.3.5. Ekuiper и аналитика на ивици

LF Edge eKuiper je лагана edge streaming analytics платформа намењена извршавању SQL-подобних правила над токовима података, обично близу извора података [15]. У систему MySweetHome, EKuiper се повезује на MQTT брокер и користи тему са телеметријом као улазни stream, дефинише ток података и правила (rules) која садрже SQL упите над тим током и на основу услова генерише команде и објављује их као MQTT поруке на излазни topic за команде.

На овај начин део логике аутоматизације (нпр. управљање климом или овлаживачем на основу услова у просторији) извршава се на edge-у (у истом окружењу где је и MQTT брокер), што смањује латенцију и оптерећење централног backend-а.

4.3.6. Оркестрација помоћу Docker-а

Сви микросервиси и инфраструктурне компоненте паковани су као Docker контејнери и оркестрирани помоћу docker compose-а.

За сваку компоненту (осим готових image-а као што су MongoDB, Mosquitto, NATS, Grafana, EKuiper) дефинисан је одговарајући Dockerfile (Python, Node.js, .NET) који описује како се апликација пакује и покреће. Compose конфигурација такође дефинише промењиве окружења (на пример, адресу MQTT брокера и NATS сервера за поједине сервисе) и међусобне зависности (depends_on), тако да се цео систем може покренути једном командом docker compose up.

4.3.7. Коришћене технологије и open-source компоненте

Систем MySweetHome заснива се на комбинацији више open-source технологија:

- Arduino MKR WiFi 1010 + MKR IoT Carrier као edge уређај са сензорима, екраном, touch дугмадима, LED диодама и IMU модулом;
- Eclipse Mosquitto као лаки MQTT брокер за комуникацију између уређаја и backend-а;
- NATS као брзи, cloud-native брокер порука за комуникацију између микросервиса;
- MongoDB као NoSQL time-series база података за чување сирових, агрегираних и безбедносних података и команди;
- Python за имплементацију SensorGatewayService-а, Node.js за StorageService, .NET за CommandService, што показује да микросервисни приступ омогућава развој сваког сервиса у различитој технологији;
- EKuiper као edge streaming analytics engine, који реализује правила за аутоматизацију на основу телеметрије;
- Grafana за визуелизацију података и креирање контролних табли и графикона;
- Docker и docker compose за контејнеризацију и оркестрацију комплетног система у развојном окружењу.

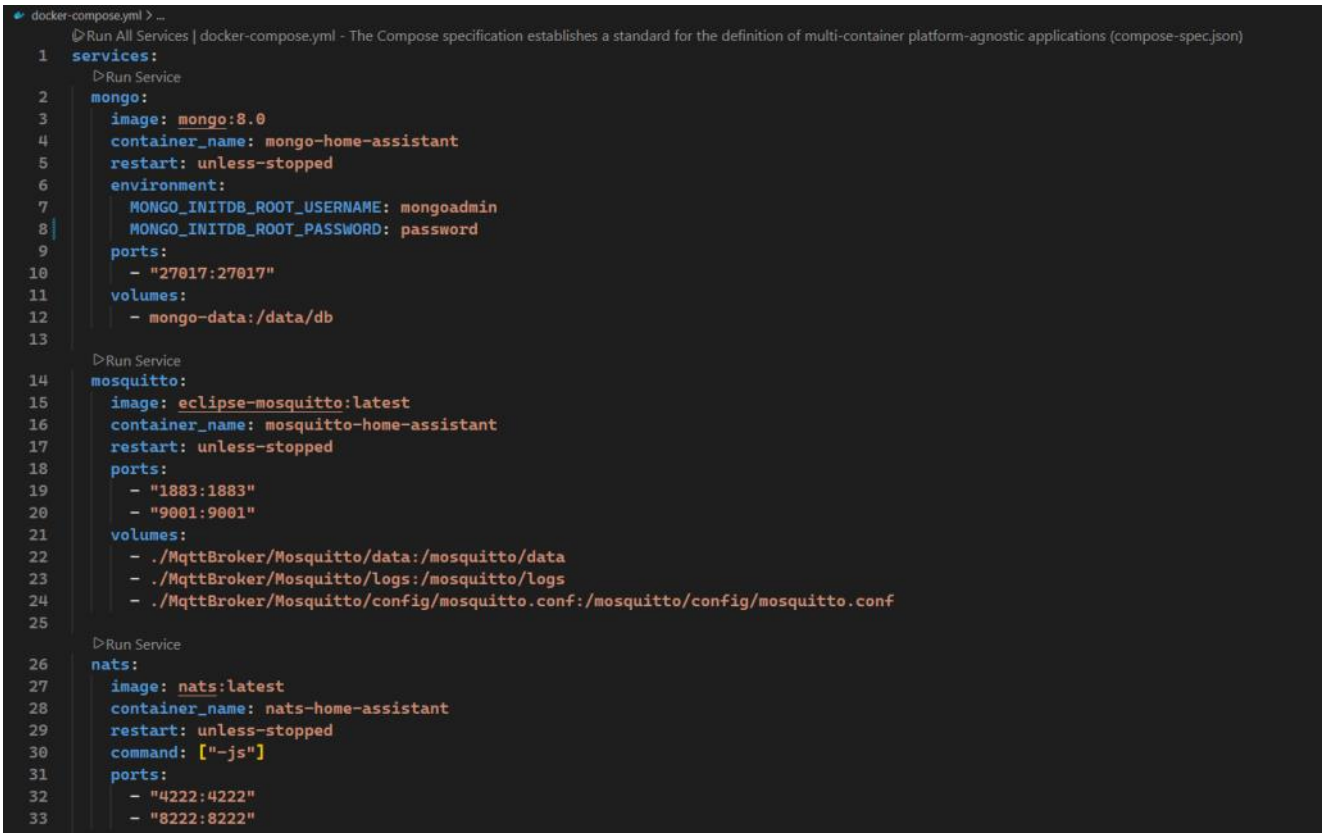
Одабир оваквог скупа технологија показује како се концепти микросервисне архитектуре и IoT система из литературе могу применити у практичном сценарију паметне куће.

5. Опис функционалности, тестирање и евалуација

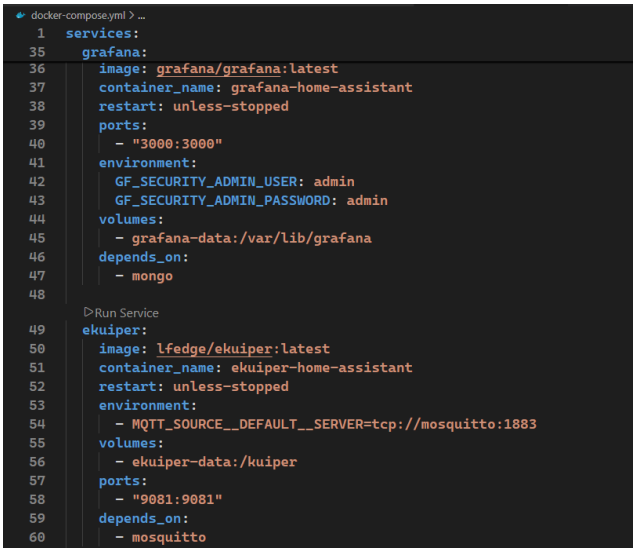
У овом поглављу приказују се функционалности система MySweetHome кроз типичне сценарије извршења, са фокусом на рад аларма, нормалан рад уређаја и аутоматизовано управљање актуаторима. Затим се описују кораци тестирања и анализира се у којој мери реализовано решење испуњава дефинисане функционалне и нефункционалне захтеве.

5.1. Сценарио извршења система

Систем MySweetHome се састоји од Arduino уређаја са MKR IoT Carrier модулом и скупа микросервиса који се извршавају у Docker окружењу, чије је оркестрирање дефинисано у конфигурационој датотеци *docker-compose.yml*, што је приказано на сликама 5.1. до 5.4. Након покретања docker compose стека (stack) стартују се сервиси MongoDB, Mosquitto MQTT брокер, NATS, Grafana, EKuiper, и микросервиси SensorGatewayService, StorageService и CommandService, и то се може видети на слици 5.5.

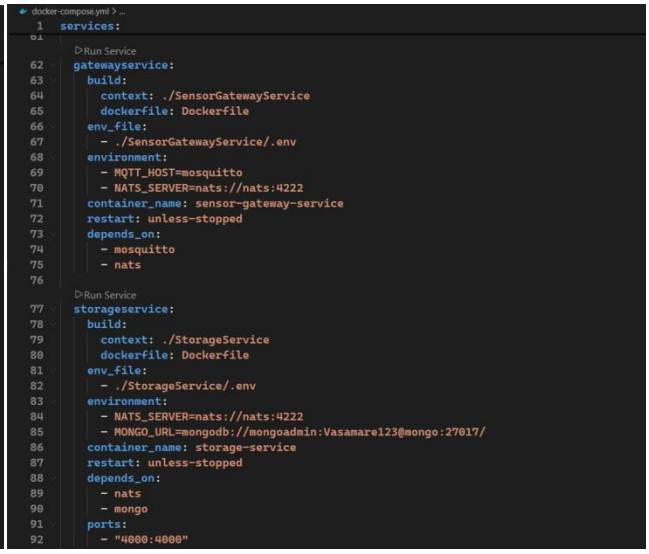


Слика 5.1 - Docker-compose.yml фајл



```
docker-compose.yml > ...
  1  services:
 35    grafana:
 36      image: grafana/grafana:latest
 37      container_name: grafana-home-assistant
 38      restart: unless-stopped
 39      ports:
 40        - "3000:3000"
 41      environment:
 42        GF_SECURITY_ADMIN_USER: admin
 43        GF_SECURITY_ADMIN_PASSWORD: admin
 44      volumes:
 45        - grafana-data:/var/lib/grafana
 46      depends_on:
 47        - mongo
 48
        ▷Run Service
 49    ekuiper:
 50      image: lfedge/ekuiper:latest
 51      container_name: ekuiper-home-assistant
 52      restart: unless-stopped
 53      environment:
 54        - MQTT_SOURCE__DEFAULT__SERVER=tcp://mosquitto:1883
 55      volumes:
 56        - ekuiper-data:/kuiper
 57      ports:
 58        - "9081:9081"
 59      depends_on:
 60        - mosquitto
```

Слика 5.2. - Docker-compose.yml фајл



```
docker-compose.yml > ...
  1  services:
        ▷Run Service
 62    gatewayservice:
 63      build:
 64        context: ./SensorGatewayService
 65        dockerfile: Dockerfile
 66      env_file:
 67        - ./SensorGatewayService/.env
 68      environment:
 69        - MQTT_HOST=mosquitto
 70        - NATS_SERVER=nats://nats:4222
 71      container_name: sensor-gateway-service
 72      restart: unless-stopped
 73      depends_on:
 74        - mosquitto
 75        - nats
 76
        ▷Run Service
 77    storageservice:
 78      build:
 79        context: ./StorageService
 80        dockerfile: Dockerfile
 81      env_file:
 82        - ./StorageService/.env
 83      environment:
 84        - NATS_SERVER=nats://nats:4222
 85        - MONGO_URL=mongodb://mongoadmin:Vasamare123@mongo:27017/
 86      container_name: storage-service
 87      restart: unless-stopped
 88      depends_on:
 89        - nats
 90        - mongo
 91      ports:
 92        - "4000:4000"
```

Слика 5.3. - Docker-compose.yml фајл

```
        ▷Run Service
 94    commandservice:
 95      build:
 96        context: ./CommandService
 97        dockerfile: Dockerfile
 98      environment:
 99        - Mqtt__Host=mosquitto
100        - Mongo__ConnectionString=mongodb://mongoadmin:Vasamare123@mongo:27017/
101      container_name: command-service-home-assistant
102      restart: unless-stopped
103      depends_on:
104        - mosquitto
105        - mongo
106
107  volumes:
108    mongo-data:
109    grafana-data:
110    ekuiper-data:
```

Слика 5.4. - Docker-compose.yml фајл

diplomski-home-assistant	-	-	-	2.1%	1 hour ago	
nats-home-assistant	c94a1304f20d	nats:latest	4222:4222 Show all ports (2)	0.05%	1 hour ago	
mosquitto-home-assistant	b2edaeb570ee	eclipse-mosquitto:latest	1883:1883 Show all ports (2)	0.05%	1 hour ago	
mongo-home-assistant	6bea1039f626	mongo:8.0	27017:27017	0.61%	1 hour ago	
storage-service	94e5963c82ea	diplomski-home-assistant-storageservice	4000:4000	0.12%	1 hour ago	
sensor-gateway-service	f42fc12d01d2	diplomski-home-assistant-gatewayservice		0.11%	1 hour ago	
grafana-home-assistant	874a9d54a45c	grafana/grafana:latest	3000:3000	0.9%	1 hour ago	
ekuiper-home-assistant	4edc79f98297	lfedge/ekuiper:latest	9081:9081	0%	1 hour ago	
command-service-home-assistant	18efd14bf024	diplomski-home-assistant-commandservice		0.26%	1 hour ago	

Слика 5.5. - Docker-compose стек (stack)

Када се и Arduino уређај повеже на бежичну мрежу и MQTT брокер, систем је спреман за рад: уређај шаље телеметрију и безбедносне догађаје, микросервиси их обрађују и чувају, а командни део система по потреби аутоматски активира или деактивира актуаторе, тако што је уређај претплаћен на тему на коју CommandService објављује команду. Део кода који то илуструје приказан је на слици 5.6.

```
Serial.print("Povezivanje na WiFi: ");
Serial.println(WIFI_SSID);
WiFi.begin(WIFI_SSID, WIFI_PASS);

while (WiFi.status() != WL_CONNECTED) {
  Serial.print(".");
  delay(500);
}
Serial.println("\nWiFi povezan!");
Serial.print("IP adresa: ");
Serial.println(WiFi.localIP());

Serial.print("Povezivanje na MQTT broker: ");
Serial.print(MQTT_BROKER);
Serial.print(":");
Serial.println(MQTT_PORT);

if (!mqttClient.connect(MQTT_BROKER, MQTT_PORT)) {
  Serial.print("MQTT konekcija neuspesna! Code: ");
  Serial.println(mqttClient.connectError());
  while (1);
}
Serial.println("MQTT povezan!");
mqttClient.setId(SENSOR_NAME);

mqttClient.onMessage(onMqttMessage);
mqttClient.subscribe(MQTT_TOPIC_ACTUATORS);
```

Слика 5.6. - Повезивање Arduino уређаја на WiFi и MQTT брокер

5.2. Сценарио рада аларма

Рад аларма заснива се на моделу стања дефинисаном у Arduino скици. Систем почиње у стању ARMED, што значи да се још увек нико није аутентификовао и уређај је у стању у ком очекује детекцију покрета преко свог IMU сензора на IoT Carrier модулу. На екрану уређаја приказује се натпис ARMED, а LED диоде су угашене. На слици 5.7. је приказано како то изгледа.

Слика 5.7. - Arduino уређај у стању ARMED

Када IMU сензор покрета детектује кретање изнад дефинисаног прага, систем прелази у стање AUTH_COUNTDOWN. У овом стању се на екрану приказује одбројавање преосталих

секунди за унос PIN кода, праћено кратким звучним сигналом сваке секунде, као што се види на сликама 5.8 и 5.9.

Слике 5.8. и 5.9. - Приказ одбројавања уређаја у стању AUTH_COUNTDOWN

Корисник преко капацитивних дугмади уноси PIN, за сваку унету цифру има визуелни приказ *, а број преосталих покушаја такође је приказан на екрану. Слике које то илуструју су 5.10 и 5.11.

Слике 5.10 и 5.11. - Визуелни приказ унетог PIN кода и преостали број покушаја

За сваки детектован покрет у ARMED стању, односно улазак у AUTH_COUNTDOWN, на MQTT events topic шаље се одговарајући безбедносни догађај (слика 5.12.), који се преко

SensorGatewayService-a и StorageService-a на крају уписује у колекцију security_events и то се може видети на слици 5.13.

```
Gyroscope:      X: 57.80        Y: 0.43 Z: -6.96
IMU: POMERANJE DETEKTOVANO
----------------------
MQTT PUBLISH [home/house_1/security/events]: {"se
Prelaz u AUTH_COUNTDOWN
```

Слика 5.12. - Уређај је детектовао померање

```
[storage][SEC] inserted: {
  sensor: 'MKR1010_WiFi',
  ts: '2025-12-01T17:02:45.404Z',
  type: 'MOTION_DETECTED_ARMED',
  from: 'ARMED',
  to: 'AUTH_COUNTDOWN'
}
```

Слика 5.13. - Евидентирање догађаја у бази

Уколико корисник у оквиру временског прозора унесе исправан PIN, систем прелази у стање NORMAL. У том тренутку се бележи догађај успешне аутентификације, а на уређају се кратко приказује екран „WELCOME“ са именом корисника, уз одговарајуће звучне и светлосне ефекте треперења зелених лед диода. Уређај подржава рад са више различитих корисника, сваки од њих има свој PIN код, и на основу тога који корисник се аутентификовао одређен је и праг вредности за задавање команди актуаторима за укључивање кућних уређаја. Успешна аутентификацију корисника приказана је на сликама 5.14 до 5.16.

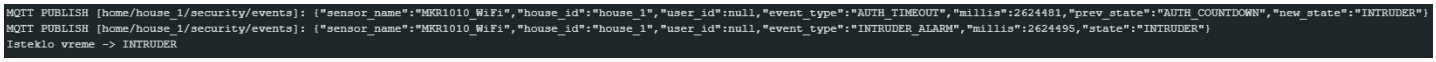
Слике 5.14, 5.15. и 5.16. - Успешна аутентификација корисника

У случају погрешног PIN-а, број неуспелих покушаја се увећава и бележи у догађају AUTH_FAILED. Након што број погрешних покушаја достигне дозвољени максимум или

протекне целокупни AUTH_COUNTDOWN интервал, систем прелази у стање INTRUDER. Тада уређај почиње да емитује звучни аларм и црвено трептање LED диода, док се на екрану приказује порука о детектованом уљезу (INTRUDER DETECTED!) . То је приказано на слици 5.17. Такође се шаље порука преко MQTT брокера и то се може видети на сликама 5.18 и 5.19.

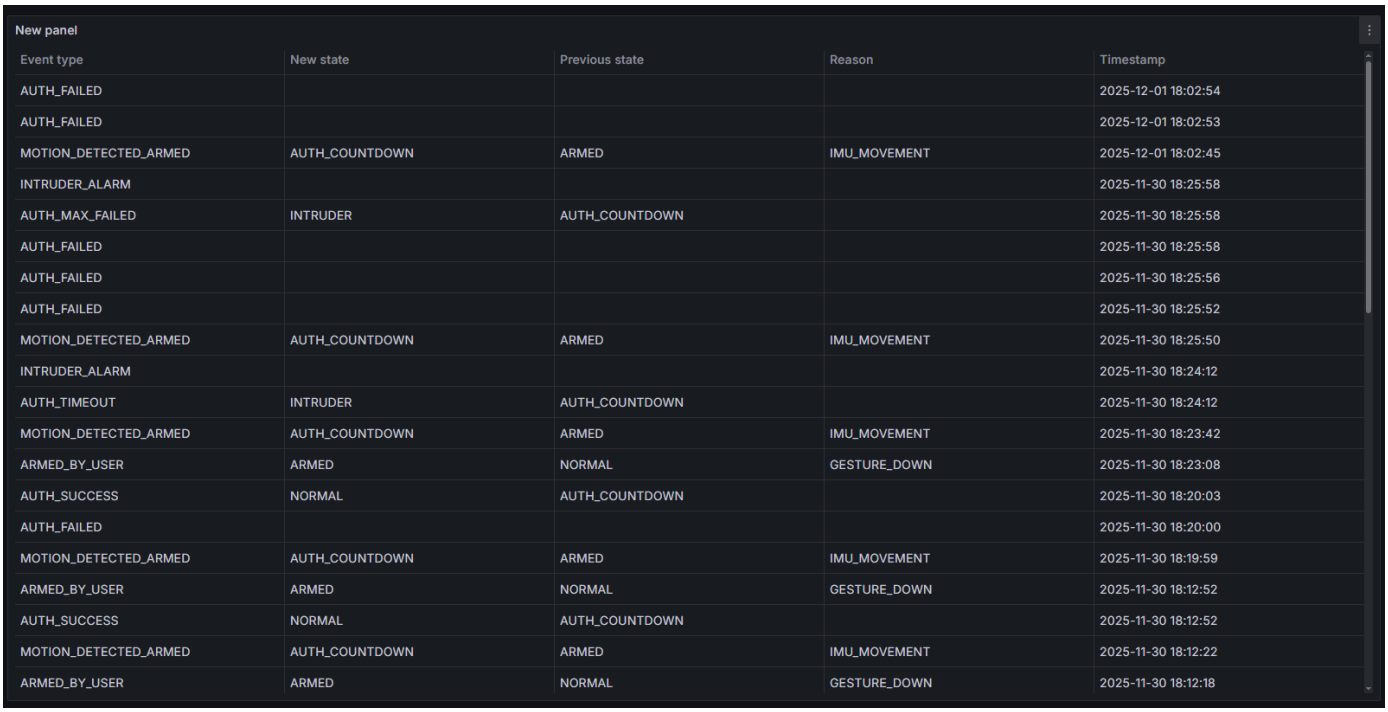
Слика 5.17. - Детекција уљеза

Слика 5.18. - Превелики број неуспешних покушаја аутентификације



Слика 5.19. - Време за унос PIN кода је истекло

У бази се ово одражава као редослед догађаја AUTH_TIMEOUT или AUTH_MAX_FAILED и INTRUDER_ALARM, који се могу пратити у Grafana табли заснованој на колекцији security_events и то је приказано на слици 5.20.



New panel

Event type	New state	Previous state	Reason	Timestamp
AUTH_FAILED				2025-12-01 18:02:54
AUTH_FAILED				2025-12-01 18:02:53
MOTION_DETECTED_ARMED	AUTH_COUNTDOWN	ARMED	IMU_MOVEMENT	2025-12-01 18:02:45
INTRUDER_ALARM				2025-11-30 18:25:58
AUTH_MAX_FAILED	INTRUDER	AUTH_COUNTDOWN		2025-11-30 18:25:58
AUTH_FAILED				2025-11-30 18:25:58
AUTH_FAILED				2025-11-30 18:25:56
AUTH_FAILED				2025-11-30 18:25:52
MOTION_DETECTED_ARMED	AUTH_COUNTDOWN	ARMED	IMU_MOVEMENT	2025-11-30 18:25:50
INTRUDER_ALARM				2025-11-30 18:24:12
AUTH_TIMEOUT	INTRUDER	AUTH_COUNTDOWN		2025-11-30 18:24:12
MOTION_DETECTED_ARMED	AUTH_COUNTDOWN	ARMED	IMU_MOVEMENT	2025-11-30 18:23:42
ARMED_BY_USER	ARMED	NORMAL	GESTURE_DOWN	2025-11-30 18:23:08
AUTH_SUCCESS	NORMAL	AUTH_COUNTDOWN		2025-11-30 18:20:03
AUTH_FAILED				2025-11-30 18:20:00
MOTION_DETECTED_ARMED	AUTH_COUNTDOWN	ARMED	IMU_MOVEMENT	2025-11-30 18:19:59
ARMED_BY_USER	ARMED	NORMAL	GESTURE_DOWN	2025-11-30 18:12:52
AUTH_SUCCESS	NORMAL	AUTH_COUNTDOWN		2025-11-30 18:12:52
MOTION_DETECTED_ARMED	AUTH_COUNTDOWN	ARMED	IMU_MOVEMENT	2025-11-30 18:12:22
ARMED_BY_USER	ARMED	NORMAL	GESTURE_DOWN	2025-11-30 18:12:18

Слика 5.20. - Визуелизација сигурносних догађаја уз помоћ Grafana табле

5.3. Сценарио нормалног рада

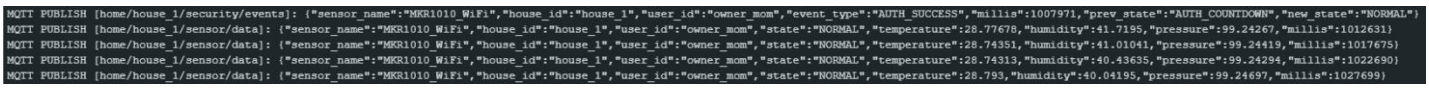
Након успешне аутентификације корисника систем улази у стање NORMAL, у коме се уређај понаша као мултифункционални уређај за праћење услова у просторији.. У овом режиму се периодично, на дефинисани интервал, очитавају вредности температуре, влажности и притиска са сензора MKR IoT Carrier модула. На екрану уређаја приказују се три странице: температура, влажност и притисак, а корисник између њих прелази помоћу гестова ЛЕВО/ДЕСНО преко сензора амбијенталног светла. На свакој страници се приказује назив величине која се тренутно мери, лого те мерене величине и вредност мерења, тако да корисник може да види на пример која је тренутна температура у просторији. Слике 5.21 до 5.23 приказују како то изгледа у реалности.

Слика 5.21. - Визуелни приказ измерене вредности температуре

Слика 5.22. - Визуелни приказ измерене вредности влажности ваздуха

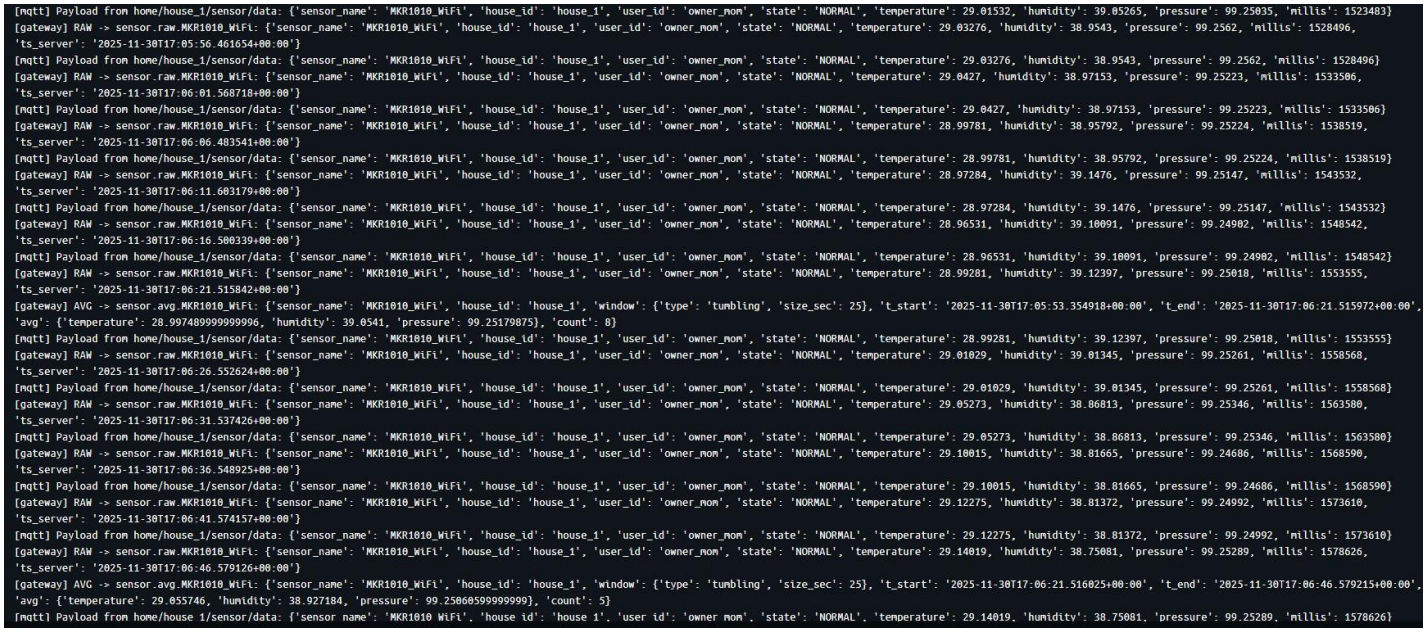
Слика 5.23. - Визуелни приказ измерене вредности притиска

Сваки пут када се очитају нове вредности, Arduino формира JSON поруку са пољима sensor_name, house_id, user_id, state, temperature, humidity, pressure и millis и шаље је на MQTT тему за податке (слика 5.24.).



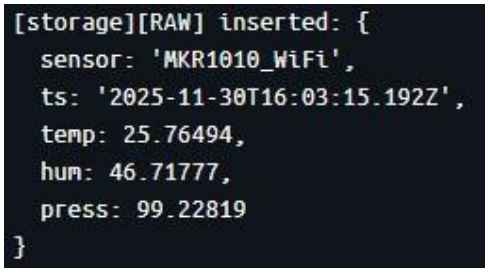
Слика 5.24. - Објављивање порука на MQTT тему од стране Arduino уређаја

SensorGatewayService прима те поруке, додаје timestamp ts_server и прослеђује их преко NATS брокера на тему sensor.raw.<sensor_name> (слика 5.25.).



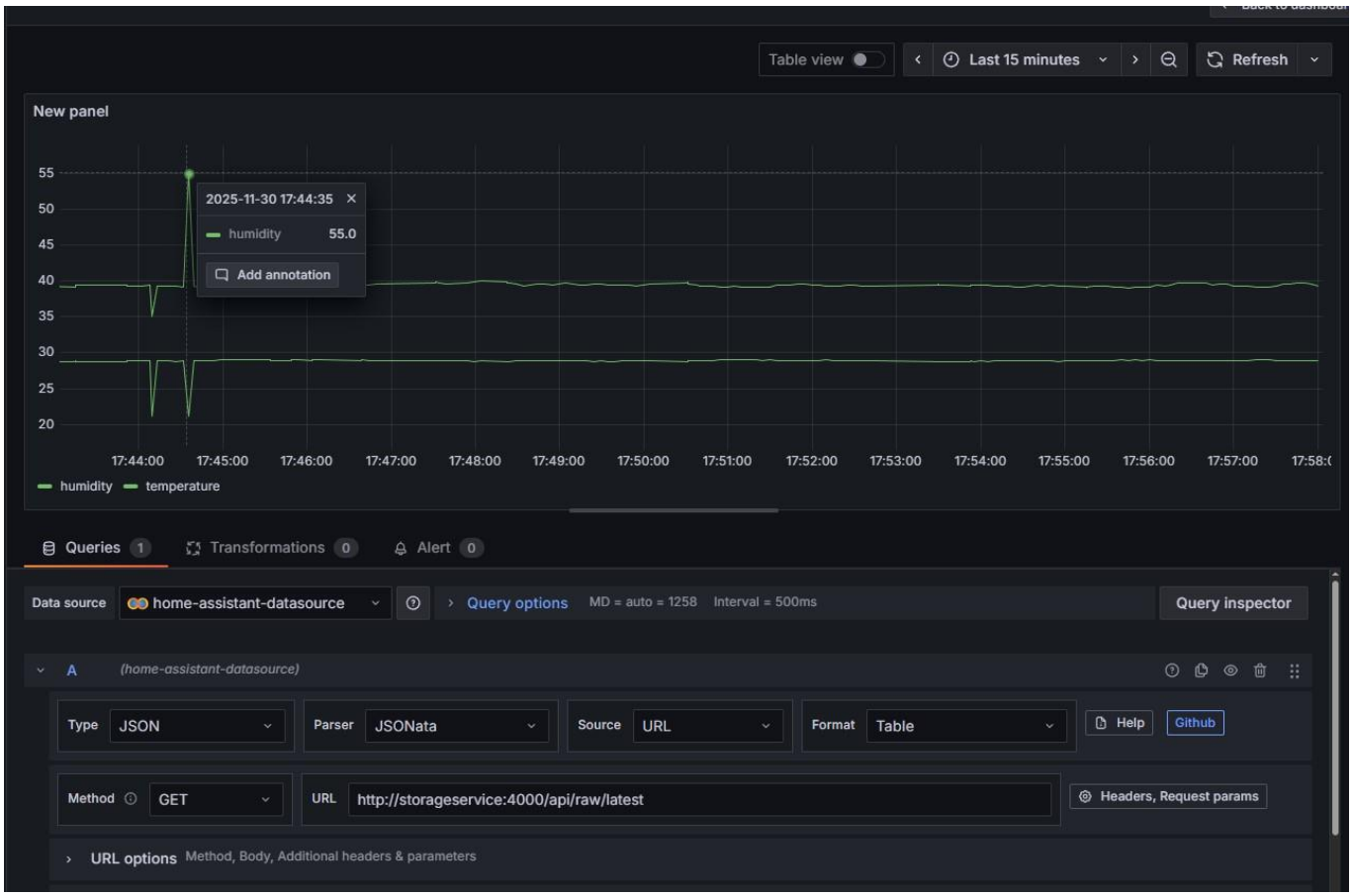
Слика 5.25. - SensorGatewayService прима и прослеђује податке са MQTT теме

StorageService затим уписује записе у временску серију raw_data (слика 5.26), а паралелно се у SensorGatewayService-у формирају и просечне вредности у одређеном временском прозору, које се шаљу на sensor.avg.<sensor_name> и чувају у колекцији avg_data (слика 5.27). Овако прикупљени подаци се у Grafana-и приказују у виду графикон, што омогућава праћење услова у просторијама кроз време, као што је приказано на сликама 5.28 и 2.29.



Слика 5.26. - Упис чистог податка у базу

Слика 5.27. - Упис просечне вредности у базу



Слика 5.28. - Grafana графикон чистих података

Слика 5.29. - Grafana графикон просечних вредности података

Корисник се из NORMAL режима може вратити у стање ARMED, на пример када напушта кућу, и тако поново активира аларм, помоћу геста ДОЛЕ преко сензора амбијенталног светла. Ток рада Arduino уређаја ARMED - NORMAL - ARMED је приказан на слици 5.30.

Слика 5.30. - Ток рада Arduino уређаја ARMED - NORMAL - ARMED

5.4. Сценарио аутоматизације актуатора

Сценарио аутоматизације актуатора показује како систем MySweetHome користи измерене параметре да би аутоматски управљао климатским условима у просторији. Први корак је дефинисање тока података које ће Ekuiper користити (слика 5.31). Затим се креирају правила у EKuiper-у, који се повезује на MQTT data topic и посматра ток мерења. Пример креираног правила приказан је на слици 5.32. Правила су креирана уз помоћ Postman алата. Правила могу, на пример, да укључе клима-уређај када температура падне испод одређеног прага, да активирају овлаживач ваздуха при ниској влажности или да покрену вентилациони вентилатор при одређеном притиску или комбинацији услова.

Слика 5.31. - Креирање тока података у Ekuiper-у

Слика 5.32. - Креирање правила у Ekuiper-у

Када услов правила буде испуњен, EKuiper емитује JSON поруку која садржи поља као што су sensor_name, house_id, device (нпр. "AC", "HUMIDIFIER", "VENT_FAN"), команду (command), нпр. "TURN_ON_AC", разлог команде (reason) и одговарајуће мерене параметре. Ова порука се шаље на MQTT тему за команде. На ту тему је претплаћен CommandService и он преузима пристигле поруке. Компонента CommandService на основу вредности поља command одређује жељено стање уређаја (укључено или искључено) и пореди га са последње познатим стањем за дати house_id и device. Ако нема промене, команда се игнорише; у супротном се ажурира локално стање и запис команде уписује у колекцију commands у MongoDB-у (слика 5.33).

Слика 5.33. - Пријем команде преко MQTT теме

Након тога CommandService формира „актуаторску“ поруку која садржи поља device, desired_state, reason, house_id, user_id и ts_server и објављује је на MQTT „actuators“ topic. На у тему је претплаћен Arduino уређај, који при пријему поруке приказује кратку поруку на екрану (нпр. „AC ON“, „HUM ON“, „WND OFF“) и истовремено плавим осветљењем LED траке сигнализира да је примљена команда. То се може видети на сликама 5.34 до 5.37. Овај визуелни повратни канал омогућава кориснику да одмах види да је систем реаговао на услове у окружењу и да је одговарајући актуатор активиран или деактивиран.

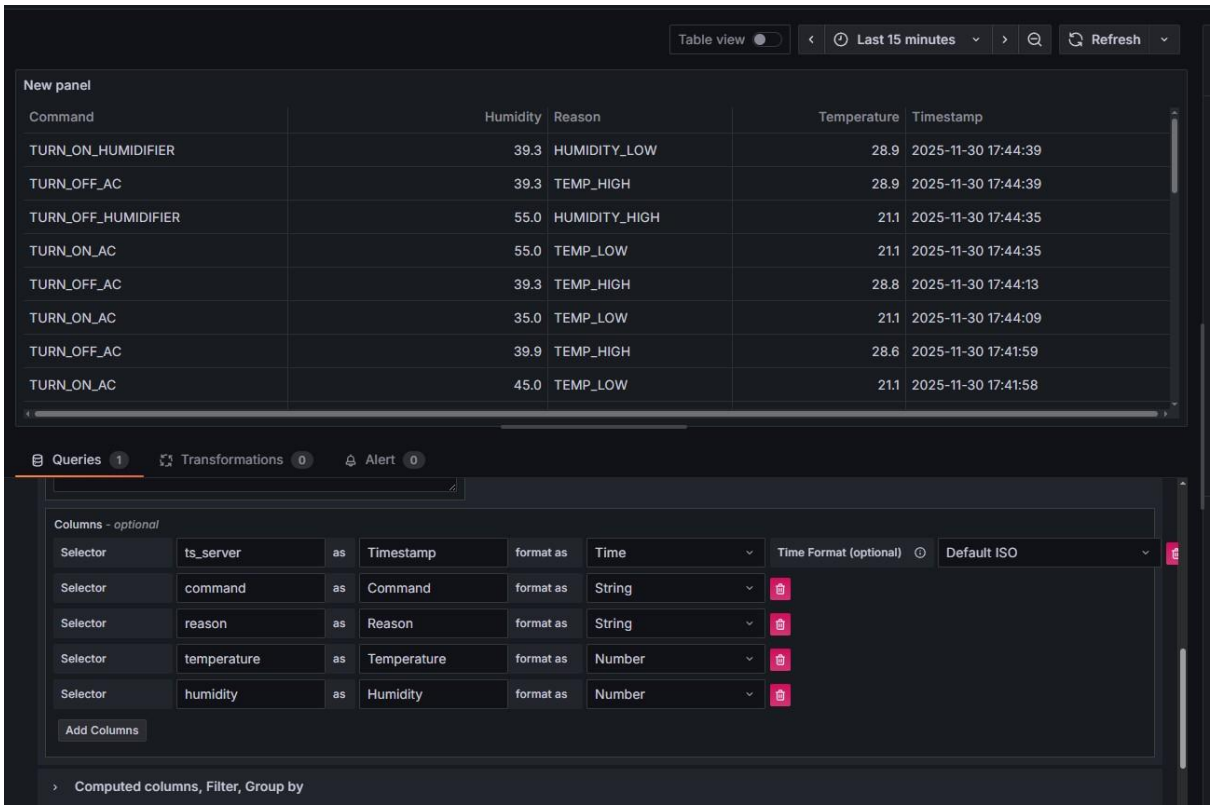
Слика 5.34. - Укључивање климе

Слика 5.35. Искључивање климе

Слика 5.36. - Укључивање овлаживача ваздуха

Слика 5.37. - Искључивање овлаживача ваздуха

Такође, корисник може помоћу Grafana табле да испрати које су команде и када биле сигнализиране од стране Ekuiper-а и CommandService-а ка Arduino уређају. То је приказано на слици 5.38.



Слика 5.38. - Приказ команди уз помоћ Grafana табле

5.5. Тестирање система

Тестирање система MySweetHome спроведено је у локалном развојном окружењу, у коме су сви микросервиси и инфраструктурне компоненте (MongoDB, MQTT брокер, NATS, EKuiper, Grafana) покретани помоћу docker compose конфигурације. Arduino уређај је био повезан на исту мрежу као и Docker окружење, а комуникација се одвијала преко MQTT брокер.

Функционално тестирање обухватило је више сценарија. У домену рада аларма проверено је: да ли се у ARMED стању детектује покрет и иницира AUTH_COUNTDOWN; да ли се унос исправног PIN кода завршава преласком у NORMAL стање; да ли се након више погрешних покушаја или истека времена систем прелази у INTRUDER стање и да ли се у свим овим случајевима генеришу одговарајући безбедносни догађаји у колекцији security_events. У домену нормалног рада проверавано је да ли се телеметрија (температура, влажност, притисак) редовно записује у raw_data и avg_data, да ли се вредности исправно приказују на екрану уређаја и да ли графикони у Grafana-и одговарају очекиваним трендовима (нпр. намерно загревање или хлађење сензора).

За сценарио аутоматизације актуатора тестирање је обухватило вештачко изазивање услова за активирање правила у EKuiper-у (нпр. снижавање температуре испод прага), уз провере да ли се генеришу одговарајуће команде у колекцији commands, да ли CommandService ажурира стање уређаја и да ли Arduino приказује поруку са називом актуатора и жељеним стањем. Поред тога, спроведени су и основни тестови отпорности: привремено заустављање StorageService-а или CommandService-а и провера да ли се систем након поновног покретања сервиса стабилизује и наставља рад без оштећења постојећих података.

5.6. Евалуација решења

Евалуација решења врши се у односу на функционалне и нефункционалне захтеве дефинисане у поглављу 4. Са функционалне стране, систем успешно реализује сценарио паметне куће са алармним механизмом: подржана су стања ARMED, AUTH_COUNTDOWN, NORMAL и INTRUDER, заједно са одговарајућим реакцијама уређаја и генерисањем безбедносних догађаја. Поред тога, реализовано је периодично прикупљање података о температури, влажности и притиску, њихово чување и визуелизација, као и аутоматско генерисање и извршавање команда за актуаторе на основу правила задатих у EKuiper-у.

У погледу нефункционалних захтева, систем показује добру модуларност и проширивост. Микросервисна архитектура омогућава да се сваки сервис (SensorGatewayService, StorageService, CommandService, EKuiper, Grafana) развија и распоређује независно, а docker compose конфигурација поједностављује покретање комплетног окружења. Комуникација преко MQTT и NATS брокера обезбеђује слабу спрегу између компоненти, што олакшава замену или додавање нових сервиса у будућности (нпр. додатни аналитички сервис или веб кориснички интерфејс). Чување података у MongoDB-у, заједно са јасно дефинисаним моделом колекција (raw_data, avg_data, security_events, commands), омогућава флексибилну каснију анализу и проширење шеме.

Ипак, постоје и ограничења решења. Систем је тестиран у кућном окружењу, са једним Arduino уређајем и једном инстанцом сервиса, без реалног оптерећења које би постојало у већим инсталацијама. Правила у EKuiper-у заснивају се на једноставним праговима и не обухватају напредније технике као што су машинско учење или адаптивне шеме управљања. Такође, кориснички интерфејс се у великој мери ослања на Grafana табле, док би у продукционом систему највероватније постојала посебна веб или мобилна апликација. Упркос овим ограничењима, MySweetHome адекватно демонстрира примену микросервисне архитектуре у IoT сценарију паметне куће и поставља добру основу за даља проширења и унапређења.

6. Закључак

У овом раду разматрана је примена микросервисне архитектуре у домену система паметних кућа, кроз конкретан пример система MySweetHome. Полазиште је био проблем све већег броја хетерогених уређаја у савременим домаћинствима и потреба да се они интегришу у јединствен, флексибилан и проширив систем. Анализом микросервисне архитектуре и IoT концепата показано је да микросервиси, уз употребу брокера порука и контејнеризације, представљају погодан модел за изградњу оваквих система, посебно када се захтева модуларност, скалабилност и јасно раздвајање одговорности између компоненти.

У теоријском делу рада дати су основни појмови микросервиса, њихове карактеристике, предности и ограничења, као и упоредни приказ са монолитном архитектуром. Затим је описана архитектура IoT система, типичне технологије и основне компоненте једног IoT микросервисног решења, уз посебан нагласак на брокере порука MQTT и NATS и на постојеће IoT платформе засноване на микросервисима. На тај начин постављен је концептуални оквир у који се природно уклапа реализовано решење MySweetHome.

Практични допринос рада је дизајн и имплементација система MySweetHome као примера микросервисног система паметне куће. Систем обухвата Arduino уређај са MKR IoT Carrier модулом, који реализује алармни механизам и прикупља телеметрију, као и три микросервиса у позадини: SensorGatewayService за пријем и агрегацију података, StorageService за трајно складиштење и излагање података према Grafana-и и CommandService за обраду команди и управљање актуаторима. EKuiper је коришћен за обраду тока података и имплементацију правила на edge-у, док Docker и docker compose обезбеђују једноставно покретање комплетног система. Реализовано је више функционалних сценарија - рад аларма, нормалан рад уређаја и аутоматизација актуатора - чиме је показано да архитектура задовољава дефинисане захтеве.

Евалуација решења показала је да систем испуњава кључне функционалне захтеве: поуздано прикупљање и чување података, генерисање и евидентирање безбедносних догађаја, као и аутоматизовано управљање актуаторима на основу правила. Са нефункционалне стране, микросервисна архитектура и употреба брокера порука обезбеђују добру модуларност, флексибилност и могућност проширења - нови сервиси (нпр. напреднија аналитика, веб или мобилна апликација) могу се додати без измена постојећих компоненти. Истовремено, уочена су и ограничења: систем је тестиран у кућним условима, на једном

уређају, а правила за аутоматизацију су релативно једноставна и не обухватају сложеније алгоритме или машинско учење.

Као будући правци развоја могу се истакнути проширење система на више сензорских чворова и више кућа, интеграција напредних аналитичких техника за предикцију и оптимизацију потрошње енергије, као и развој прилагођеног корисничког интерфејса (веб или мобилна апликација) за конфигурацију правила и надзор система. Такође, систем се може интегрисати са другим IoT платформама или cloud услугама, чиме би се додатно потврдила интероперабилност предложене архитектуре. Упркос наведеним ограничењима, реализовано решење MySweetHome успешно демонстрира како се микросервисна архитектура, у комбинацији са IoT технологијама, може применити за изградњу савременог система паметне куће који је истовремено функционалан, модуларан и спреман за даља унапређења.

Литература

[1] Sam Newman, Building Microservices (година издања 2015.)

[2] Perry Lea, IoT and Edge Computing for Architects 2nd Edition (година издања 2020.)

[3] Wikipedia, „Microservices“ (https://en.wikipedia.org/wiki/Microservices, последњи приступ 30.11.2025.).

[4] Omar Al-Debagy & Peter Martinek, „A comparative review of microservices and monolithic architecture“ (година издања 2018.)

[5] Microsoft Docs / .NET microservices guidance (https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/microservices-architecture, последњи приступ 30.11.2025.).

[6] „5 Essential Microservices Design Patterns“ (https://www.osohq.com/learn/microservices-design-patterns, последњи приступ 30.11.2025.).

[7] Edalab, „Using Microservices Architecture for IoT Solutions“ (https://edalab.it/en/microservizi-soluzioni-iot/, последњи приступ 30.11.2025.).

[8] MQTT званична документација (https://mqtt.org/, последњи приступ 30.11.2025.)

[9] NATS званична документација (https://docs.nats.io/, последњи приступ 30.11.2025.)

[10] OpenRemote званична документација (https://www.openremote.io/, последњи приступ 30.11.2025.)

[11] MainFlux званична документација (https://mainflux.com/, последњи приступ 30.11.2025.)

[12] thin-edge.io званична документација (https://thin-edge.io/, последњи приступ 30.11.2025.)

[13] SiteWhere званична документација (https://sitewhere.io/, последњи приступ 30.11.2025.)

[14] EdgeX Foundry званична документација (https://www.edgexfoundry.org/, последњи приступ 30.11.2025.)

[15] eKuiper званична документација (https://ekuiper.org/, последњи приступ 30.11.2025.)