ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

Факултет по Приложна математика и информатика

Д И П Л О М Н А Р А Б О Т А

Софтуер за "Сензора като услуга" в екосистемата на IoT базирана на технологията OSGi

Дипломант:

/ Десислава Емилова Милушева /

Научен ръководител:

/ проф. дн инж. Пламенка Боровска /

София, 2023г.

Утвърдено дипломно задание

Съдържание

Увод **3**
Глава 1 ..... **4**
Обзор на проблемите и съществуващите решения **5**
1.1. Съществуващи решения за реализация на софтуер за "Сензора като услуга" 5
1.1.1. Технологични решения на база типа устройствата и сензорите 5
1.1.2. Технологични решения на база начина на комуникация 7
1.1.3. Технологични решения на база архитектурата на системата 8
Глава 2 ..... **9**
Проектиране на софтуерната архитектура **10**
2.1. Цели и задачи 10
2.2.1. Избор на хардуер 11
SONOFF ZBDongle-P Zigbee 3.0 11
Сензор за движение SONOFF SNZB-03 12
Сензор за контакт SONOFF SNZB-04 13
2.2.2. Избор на софтуер 15
Операционна система 15
OSGi (Open Service Gateway Initiative) 16
Spring Framework 17
Angular 17
Zigbee протокол 18
MQTT протокол (Message Queuing Telemetry Transport) 19
Zigbee2MQTT 19
Mosqutto 20
Docker 21
Глава 3 ..... **24**
Развитие на софтуера **24**
3.1. Подготовка на средата 24
Стартиране на SONOFF ZBDongle-P Zigbee 3.0 24
Стартиране на сензорите 27
3.2. Изграждане и разработка 28
3.3. Стартиране на OSGi и Angular 41
3.4. Автоматизиране на процеса по стартиране 42
Глава 4 ..... **44**
Изследване на характеристиките на реализираната система **44**
Заключение **45**
Декларация за авторство **47**
Използвани източници **48**

Увод

В последните години интернет на нещата (Internet of Things) се превърна в ключова технология за свързване на различни устройства и системи. Сензорите са едни от най-важните елементи на IoT, тъй като те позволяват събирането на данни от физическия свят и тяхното трансформиране в цифрови данни. От ключово значение за ефективното използване на IoT е възможността за свързване и управление на голям брой устройства и сензори. В този контекст, концепцията "Сензора като услуга" може да бъде използвана за предоставяне на услуги, свързани със събирането и анализа на данни от сензори.

Съществуват различни софтуерни имплементации целящи предоставне на възможност за по-ефективно и гъвкаво управление на IoT хардуерни решения, но в тази дипломна разработка фокусът ще бъде технологията OSGi (Open Services Gateway Initiative), която предоставя архитектурен модел за изграждане на софтуерни приложения, които могат да бъдат динамично променяни и ъпдейтвани без да се налага да се прекомпилират или рестартират цели приложения. Това прави OSGi идеална за разработване на приложения в IoT екосистемата, където сензорите и устройствата могат да се добавят и премахват по всяко време.

Целта на тази дипломна работа е да разработи софтуер за "Сензора като услуга" в екосистемата на IoT базирана на технологията OSGi. Работата ще се фокусира върху проектирането и разработката на софтуерната архитектура, както и върху изследването на характеристиките на реализираната система, която ще позволява гъвкаво добавяне и управление на нови сензори и устройства. В процеса на разработка на софтуера ще бъдат използвани принципите на OSGi за динамично управление на компонентите.

Този софтуер цели да покаже как може да реализира един реални сценарии на IoT, като подобри функционалността и управлението на сензорите и устройствата в екосистемата. Като резултат, това ще допринесе за увеличаването на ефективността и удобството на приложенията, използващи данни от IoT, както и за улесняване на процесите в различни сфери на живота.

Глава 1

Обзор на проблемите и съществуващите решения

1.1. Съществуващи решения за реализация на софтуер за "Сензора като услуга"

В областта на IoT има множество решения за събиране и управление на данни от различни сензори и устройства. Тези решения могат да бъдат класифицирани на базата на различни критерии, като например:

- типа на устройствата и сензорите
- начина на комуникация
- архитектурата на системата
- методите за анализ на данните

Тази част на проучване цели да оформи общ поглед върху технологичните решения за създаване на системи за контрол на устройства в екосистемата на IoT.

1.1.1. Технологични решения на база типа устройствата и сензорите

Един от основните типове устройства в IoT са сензорите за събиране на данни. Те са проектирани да събират данни от физическия свят, като например температура, влажност, осветеност, движение, звук и много други. Сензорите могат да бъдат различни типове, като например термометри, влагомери, светлинни сензори, акселерометри, гироскопи и други. В зависимост от типа устройство или сензор, използван в системата, се изисква различна технологична интеграция и обработка на данните. Например, могат да бъдат използвани следните типове устройства и сензори:

- Устройства за измерване на температура и влажност - тези устройства се използват за събиране на данни за температурата и влажността на околната среда. Те са важни за мониторинга на климатичните условия и съхранението на хранителни продукти. Пример за такова устройство е DHT11 сензор.

- Устройства за измерване на осветеност - тези устройства са важни за мониторинга на светлината в сгради и открити пространства. Те се използват за контрол на осветлението във вътрешни помещения, за да се намали енергийната консумация. Пример за такова устройство е BH1750FVI сензор.

- Устройства за измерване на движение - тези устройства са важни за мониторинг на дейността в сгради и други места. Те се използват за контрол на движението на хора и животни в обекти и извън тях. Пример за такова устройство е PIR сензор.

- Устройства за измерване на звук - тези устройства са важни за мониторинг на шумовата замърсеност в сгради и други места. Те се използват за контрол на нивото на шума, който може да доведе до здравословни проблеми за хората и животните. Пример за такова устройство е микрофонен сензор.

- Устройства за измерване на качеството на въздуха - тези устройства могат да измерват нивата на замърсявания във въздуха, като например вредните газове и твърди частици. Те са полезни за мониторинг на качеството на въздуха в градовете, промишлени зони и други места. Пример за такова устройство е SDS011 сензор.

- Устройства за измерване на движение на вода - тези устройства са важни за мониторинг на водата в реки, язовири, съдове и други места. Те се използват за контрол на водните източници, за да се следят нивата на водата, както и за прогнозиране на наводнения и други катастрофални събития. Пример за такова устройство е ултразвуковият сензор HC-SR04, който използва ултразвукови вълни, за да измерва разстоянието до повърхността на водата.

Сензорните устройства, са устройства, които преобразуват физически величини, като например температура, налягане, светлина, звук и други, в електрически сигнали, които могат да бъдат обработени от електронни устройства. В зависимост от типа си, сензорите могат да използват различни технологии за преобразуване на физическите величини в електрически сигнали.

Основните компоненти на сензорите включват:

- Измервателен елемент - това е частта от сензора, която реагира на измерваната физическа величина и генерира електрически сигнал. Например, при термометър

това може да бъде термочувствителен резистор, който променя своята съпротивление в зависимост от температурата.

- Усилвател - това е електронен компонент, който усилва слабия електрически сигнал от измервателния елемент, за да може да бъде обработен от други електронни устройства. Усилвателът може да бъде вграден в сензора или да се намира отделно от него.

- Филтър - това е компонент, който филтрира нежеланите смущения в електрическия сигнал и го пречиства, за да може да бъде обработен от електронните устройства.

- Преобразувател - това е електронен компонент, който преобразува аналоговия електрически сигнал от сензора в цифров формат, който може да бъде обработван от микроконтролер или друг компютърен система. Преобразувателят може да бъде вграден в сензора или да се намира отделно от него.

- Комуникационен интерфейс - това е частта от сензора, която позволява на сензора да комуникира с други устройства, като например компютъри, микроконтролери или IoT системи. Интерфейсът може да бъде проводен или безжичен и може да използва различни протоколи за комуникация.

Принципите на работа на сензорите и тяхното устройство могат да варират в зависимост от типа на сензора и измерваната величина. В обобщение, сензорите работят на принципа на преобразуване на енергия от една форма в друга, като това може да бъде електрическа, механична, термална, оптична или друга форма на енергия. Те използват различни физически променливи, като температура, налягане, осветеност, вибрации, силови въздействия и други, за да събират информация от околната среда и да я преобразуват в електрически сигнали.

1.1.2. Технологични решения на база начина на комуникация

Комуникацията между устройствата е от съществено значение. Различните устройства са в състояние да комуникират по различни начини, като използват различни протоколи и стандарти за обмен на данни. В този контекст, технологични решения на база начина на комуникация на сензорите предлагат множество възможности за събиране на данни и изграждане на интелигентни системи.

Безжични мрежи са най-често използвания начин за комуникация между сензорите в IoT. Те използват радиовълни за прехвърляне на данни от един край до друг. Сред тези

безжични технологии могат да се посочат Zigbee, Wi-Fi, Bluetooth, LoRaWAN, NB-IoT и др.

- Zigbee е стандарт за безжична мрежа, който се използва в IoT за малки мрежи с ниска консумация на енергия. Той е идеален за сензори, които изпращат малки количества от данни на голяма дистанция. С Zigbee мрежите могат да се свързват до 65,000 устройства.

- Wi-Fi е популярна безжична технология за достъп до интернет. Тя може да се използва за събиране на данни от устройства в домове, офиси и магазини. С Wi-Fi мрежи могат да се свързват до 250 устройства.

- Bluetooth е мрежа за краткодистанционна комуникация между устройства. Той е често използван за прехвърляне на данни между телефони и други персонални устройства, като слушалки и часовници. С Bluetooth мрежи могат да се свързват до 7 устройства.

- LoRaWAN е дългодистанционна безжична мрежа за IoT устройства. Той може да покрие големи области, като градове и дори страни. Той е идеален за използване в устройства за мониторинг на околната среда, като температура, влажност, качество на въздуха и т.н.

- NB-IoT е технология за безжична мрежа за IoT, която използва мобилните оператори. Той предоставя широка покритие и дълга батерия на устройствата, което го прави подходящ за дългосрочни IoT приложения.

Освен безжичните, има и проводни протоколи, които могат да бъдат използвани за комуникация на сензорите. Например, Modbus е протокол за серийна комуникация, който се използва за измерване и контрол на променливи в промишлеността. С Modbus сензорите могат да бъдат свързани посредством RS-485 или Ethernet интерфейси.

Технологичните решения на база начина на комуникация на сензорите имат голямо приложение в различни области като индустрия, селското стопанство, транспорт, здравеопазване, умни градове, умни домове и други. Тези решения позволяват бързо и ефективно събиране на данни, което дава възможност за по-добро разбиране и управление на околната среда и процесите в различните сфери на живота.

1.1.3. Технологични решения на база архитектурата на системата

Технологичните решения на база архитектурата на системата в IoT са от съществено значение за ефективното функциониране и взаимодействие между различните компоненти и устройства. Архитектурата на системата определя начина, по който данните се събират, обработват и прехвърлят, както и начина, по който се контролират устройствата и изпълняват действия на база получената информация.

Едно от популярните технологични решения в IoT е облачният модел на архитектурата. Това включва използването на облачни услуги и платформи за съхранение и обработка на данни. В IoT, сензорите и устройствата събират данни от околната среда и прехвърлят информацията към облачни сървъри, където данните се обработват и анализират. Този модел предоставя гъвкавост, мащабируемост и лесен достъп до данни от различни устройства и местоположения. Пример за това решение е Amazon Web Services (AWS) IoT Core и Microsoft Azure IoT Hub, които предоставят интегрирани услуги за управление и обработка на данни в IoT системи.

Освен облачния модел, едно от новите и нарастващи технологични решения е решението "Edge Computing". Това се базира на обработката на данни непосредствено в близост до устройствата и сензорите, вместо да се изпращат всички данни към централния облак. Това намалява натоварването на мрежата и ускорява обработката на данните. Пример за това решение е Google Cloud IoT Edge, което предоставя възможност за обработка на данни на устройствата или в локални области преди да бъдат прехвърлени към облака.

В някои приложения може да се използва децентрализиран модел на архитектурата, където сензорите и устройствата работят независимо и са в състояние да комуникират директно помежду си. Този модел е особено полезен в области, където мрежовото свързване е ограничено или несигурно. Пример за това решение са мрежи на базата на Zigbee или Z-Wave протоколите, които позволяват на устройствата да комуникират директно помежду си в домашни или индустриални среди.

Други технологични решения включват използването на контейнерни технологии като Docker и Kubernetes, които позволяват лесно разпределение и управление на приложения и услуги в IoT системи. Също така, се използват протоколи за мрежово взаимодействие като MQTT (Message Queuing Telemetry Transport) и CoAP (Constrained Application Protocol), които се основават на леки и ефикасни комуникационни протоколи за IoT устройства с ограничени ресурси.

Всички тези технологични решения имат своите предимства и недостатъци и се използват в зависимост от специфичните изисквания и приложения на IoT системата. Най-важното е да бъде избрана правилната архитектура на системата, която отговаря на нуждите и целите на конкретното IoT приложение.

Глава 2

Проектиране на софтуерната архитектура

2.1. Цели и задачи

Целта на настоящата дипломна работа е да се създаде и разработи хардуерно-софтуерна система, която демонстрира потенциала на "Сензора като услуга" в интегрирането със софтуерна платформа, базирана на IoT и технологията OSGi. Този проект има за цел да предложи едно иновативно решение, което да улесни събирането, управлението и анализа на данни от различни типове сензори в IoT средата. Ще бъде изградена софтуерна платформа, която в синхронизация с различни сензори поставени в сградово помещение, като дом, офис, учебен кабинет и т.н., превръща подобен тип пространства в умни домове и сгради, като предоставя различен вид инфрмация на база на използваните сензори, като например, влажност на въздуха, температура, осветеност, отчитане на движение, отваряне на врата или прозорец, сигнализация за наводнение или пожар и други.

Проектирането на софтуерната архитектура е от решаващо значение за подобен тип системи. То изисква комбиниране на физическия свят на сензорите с виртуалния свят на софтуера, като важен фактор е лесната адаптация към различни устройства и сензори, и осигуряването на ефективна комуникация между тях.

За реализацията на проекта, ще бъде използвана OSGi (Open Service Gateway Initiative) като ключова технология за управление на модулите и компонентите. Заедно с нея, ще бъде използвана и платформата Spring, която слага още едно помощно ниво, относно бързата и лесна разработка на уеб приложения. Това ще осигури гъвкавост и лесно добавяне или премахване на функционалности, без да нарушава работата на системата. При добавяне на нови сензори или устройства, системата ще може лесно да се надгради и да интегрира нова фънкционалност без нужда от прекомпилиране или рестартиране.

Освен OSGi, за разработката на уеб интерфейса ще използваме технологията Angular, която предоставя модерни и интерактивни уеб интерфейси. Това ще даде възможност на потребителите да визуализират и управляват данните и настройките на системата чрез лесен за използване интерфейс.

С цел да улесним мениджмънта и развитието на проекта, ще използваме и контейнеризация с Docker. Това позволява да изолираме и разгърнем лесно отделните компоненти на системата, както и да гарантираме съвместимостта между тях.

Съчетавайки различни технологии и компоненти, ще бъде изградена модулна и гъвкава система, която може да се адаптира към разнообразни сценарии и изисквания. Такава архитектура осигурява висока производителност, лесен мениджмънт и готовност за бъдещо разширение и развитие на проекта.

2.2. Описание на използваните технологии и хардуер

В процеса на проектиране на архитектурата и съобразно изискванията на проекта за създаване на система, която е в състояние да се справи с различни видове сензори и устройства ще бъдат използавни различни технологии и инструменти, както софтуерни така и хардуерни.

2.2.1. Избор на хардуер

Една от първите стъпки в проектирането на архитектурата е изборът на подходящ хардуер, който да бъде използван за функционирането на системата. За успешната реализация на проекта, ще бъде използван компютър, с процесор с добра производителност, като например Intel Core i5 или i7, и с достатъчно RAM памет, например 16GB или повече. Той ще влезе в ролята на контролер, което ще позволи управлението на сензорите.

За да се осъществи връзката и комуникацията между контролера (компютъра) и сензорите, има нужда от устройство посредник. За тази цел ще бъде използван Zigbee dongle. Това устройство предоставя възможност за свързване на хост системата (например компютър или хъб) с Zigbee мрежата и комуникация със съвместимите Zigbee устройства.

SONOFF ZBDongle-P Zigbee 3.0

SONOFF ZBDongle-P Zigbee 3.0 е висококачествен Zigbee dongle, предоставящ възможност за свързване и управление на Zigbee устройства в умни домове и IoT системи. Този устройство е създадено с цел да предостави на потребителите надеждна и стабилна връзка между разнообразни Zigbee устройства и контролиращата платформа. По-долу ще разгледаме подробно характеристиките, параметрите и спецификациите на SONOFF ZBDongle-P Zigbee 3.0:

Характеристики:

- Zigbee 3.0 съвместимост: SONOFF ZBDongle-P е съвместим с Zigbee 3.0 протокола, който предоставя поддръжка на разнообразни Zigbee устройства от различни производители.
- Безжична връзка: Този dongle осигурява безжична връзка със Zigbee устройствата в умния дом, което ги свързва към контролиращата система.
- Лесна интеграция: Устройството е лесно за инсталиране и интегриране в съществуващите IoT инфраструктури и системи за управление.
- Поддръжка на мрежи с голям обхват: Zigbee мрежата може да се разпростира на голям обхват, като дава възможност за покритие на цялата домашна площ.

Параметри и спецификации:

- Работна честота: Устройството работи на честота около 2.4 GHz, като използва Zigbee 3.0 комуникационен протокол.
- Поддържани устройства: SONOFF ZBDongle-P е съвместим с разнообразни Zigbee устройства като сензори, осветление, контролери и други.
- Съвместимост: Устройството може да се интегрира със съвместими платформи за управление на умния дом.
- Размери: SONOFF ZBDongle-P има компактни размери, което го прави лесен за поставяне и използване в различни среди.
- Интерфейс: Устройството разполага с USB интерфейс за свързване към контролиращата платформа.
- Захранване: Захранва се чрез USB порт, което осигурява удобство при включване.

SONOFF ZBDongle-P Zigbee 3.0 предоставя на потребителите мощен инструмент за свързване и управление на Zigbee устройства. Той гарантира надеждна безжична връзка и съвместимост с разнообразни IoT решения. Този dongle от SONOFF предоставя възможност за разширяване на функционалността на умния дом.

Последния необходим хардуерен компонент са самите сензори. Важно условие, което трябва да се съобрази при изборът им е, че използвайки Zigbee dongle за връзка с контролера, сензорите също трябва да използват и поддържат Zigbee протокол за комуникация. За целите на проекта и нагледното демонстриране на реализацията, ще бъдат използвани следните сензори:

Сензор за движение SONOFF SNZB-03

Сензорът за движение SONOFF SNZB-03 представлява висококачествен компонент, предназначен за интеграция в системи за умни домове и IoT приложения. Този сензор е проектиран да разпознава движение и да предава релевантни данни за по-нататъшна обработка. В следващите редове ще ви предоставя подробно описание на характеристиките, параметрите и спецификациите на Сензора за движение SONOFF SNZB-03.

Параметри и спецификации:

- Работна честота: Сензорът работи на Zigbee честота, която около 2.4 GHz.
- Дистанция на разпознаване: Специфичната дистанция на разпознаване може да варира, но обикновено е в рамките на няколко метра.
- Работна температура: SNZB-03 е проектиран да работи в различни температурни условия, като голям диапазон осигурява надеждност на устройството.
- Батерия: Сензорът се захранва от CR2450 батерия, която обикновено предоставя продължително време на работа.
- Размери: Размерите на сензора са компактни и го правят подходящ за инсталация на различни места.
- Съвместимост: Устройството е съвместимо със системи, базирани на Zigbee протокола.

Сензорът за движение SONOFF SNZB-03 предоставя надеждно и ефективно разпознаване на движение, което го прави подходящ за приложения в умни домове, сигурностни системи и други IoT среди. Неговите характеристики, параметри и спецификации го правят подходящ избор за интегриране в различни IoT решения.

Сензор за контакт SONOFF SNZB-04

Сензорът за контакт SONOFF SNZB-04 е интелигентно устройство, предназначено за мониторинг и управление на състоянието на врати, прозорци, шкафове и други обекти в IoT системата. Този сензор предоставя възможност за мониторинг на статуса на отворени и затворени обекти и подаване на информация за техния статус към контролиращата платформа.

Параметри и спецификации:

- Работна честота: Използва Zigbee 3.0 протокол за безжична комуникация.
- Работен обхват: Сензорът има работен обхват, който позволява далечно мониториране на обекти.
- Съвместимост: Съвместим е с други Zigbee устройства и Zigbee контролиращи платформи.
- Сензорът е с компактни размери, което го прави подходящ за различни места и обекти.
- Индикация за статус: Сензорът може да изпраща сигнали за отворен или затворен статус чрез контролиращата платформа.
- Захранване: Работи на батерии, което предоставя удобство за монтаж на различни места без нужда от захранващ кабел.

Сензорът за контакт SONOFF SNZB-04 е изключително полезен инструмент за мониторинг на отворени и затворени обекти в умния дом. Той предоставя надежден начин за получаване на информация за състоянието на обектите и взаимодействие с тях чрез контролиращата платформа. С този сензор потребителите могат да подобрят сигурността, контрола и удобството на своя умни дом или IoT система.

Сензор за температура и влажност на въздуха TuYa WSD500A

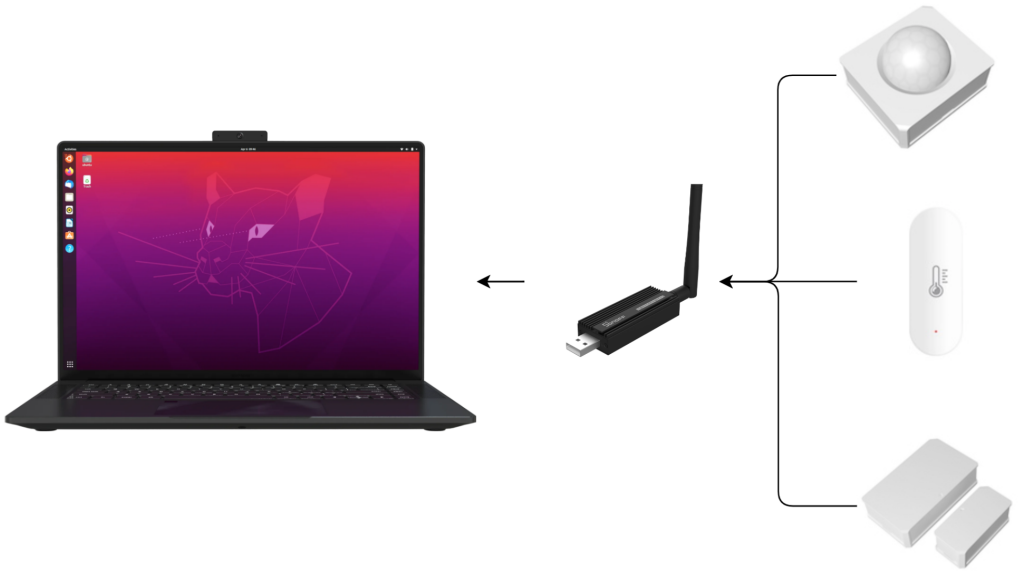
Сензорът за температура и влажност на въздуха TuYa WSD500A е интелигентно устройство, което предоставя информация за околната температура и влажност. Този сензор е нарочно разработен, за да осигури детайлни данни за климатичните условия в околния въздух, което може да бъде използвано за подобряване на комфорта, контрола на влажността и оптимизиране на енергийната ефективност на дома или работното пространство.

Параметри и спецификации:

- Измерване на температурата: Покрива широк диапазон на температури, който позволява измерване на различни климатични условия.
- Измерване на влажността: Предоставя информация за влажността на въздуха, която е ключов фактор за комфорт и качество на въздуха.
- Размери и дизайн: Компактни размери и елегантен дизайн, които позволяват сензорът да се интегрира хармонично в различни интериори.

- Захранване: Работи на батерии, което осигурява гъвкавост при поставянето му в различни места.
- Съвместимост: Съвместим със смарт устройства и приложения, което позволява лесна интеграция в умните системи.

Сензорът за температура и влажност на въздуха TuYa WSD500A предоставя ценни данни за климатичните условия и околната среда. Този сензор може да бъде използван както за персонално комфортно обслужване, така и за управление на климатичната система, за поддържане на оптимални условия във вашия дом или офис. Благодарение на неговите точни измервания и безжична свързаност, сензорът предоставя ценна информация, която може да бъде използвана за подобряване на животния стил и ефективността на работните пространства.



2.2.2. Избор на софтуер

Изборът на подходящ софтуер играе ключова роля в успешната реализация на проекта за "Сензора като услуга" в екосистемата на IoT, базиран на технологията OSGi. Този етап е от съществено значение, тъй като софтуерът осигурява необходимите инструменти и инфраструктура за управление, комуникация и анализ на данните от сензорите. В контекста на проекта, като основни цели се залага разработването на ефективна и гъвкава система, която може да се адаптира към различни видове сензори и устройства.

Операционна система

Изборът на операционна система (ОС) е критична стъпка в процеса на проектиране и реализация на системата за "Сензора като услуга" в екосистемата на IoT, базирана на технологията OSGi. В този контекст, изборът на ОС има дълбоко влияние върху функционалността, сигурността, поддръжката и гъвкавостта на системата. Операционната система Linux Ubuntu 22.04 LTS (Long-Term Support) се явява обещаващ избор за реализацията на този проект.

Ubuntu 22.04 LTS представлява последната версия на една от най-популярните дистрибуции на операционната система Linux. LTS версиите (Long-Term Support) се отличават със стабилността и дълъг срок на поддръжка, което ги прави предпочитани за интеграция в IoT проекти. В случая на проекта, операционната система Ubuntu 22.04 LTS предоставя няколко ключови предимства, тя е стабилна и надеждна платформа за разработка и изпълнение на софтуерни приложения. Като част от семейството на Linux дистрибуциите се отличава с много предимства, които я правят предпочитан избор за различни типове проекти, включително IoT приложения.

В съчетание с технологията OSGi, която осигурява модулна архитектура на приложенията, Ubuntu 22.04 LTS предоставя основата за разработване на система, която е в състояние да се справи със сложните изисквания на проекта.

OSGi (Open Service Gateway Initiative)

OSGi (Open Service Gateway Initiative) е отворена стандартизирана платформа, предоставяща инфраструктура за разработка и управление на приложения в модулен и динамичен начин. Основната цел на OSGi е да улесни създаването на сложни и разнообразни софтуерни системи, като позволява на различни модули и компоненти да работят в хармония, без да се налага прекомпилация на цялото приложение.

Основният принцип на OSGi е модулността. Приложението се състои от множество малки и независими модули, наречени "бандли", които могат да бъдат инсталирани, активирани и деактивирани по време на работа на системата. Този модулен подход позволява на разработчиците да създават отделни компоненти, които могат да бъдат поддържани и обновявани независимо, без да се нарушава функционалността на цялата система.

OSGi предоставя механизми за управление на зависимостите между модулите, което гарантира, че всички необходими ресурси са налице преди инсталацията на даден модул. Това спомага за предотвратяване на конфликти и прекъсвания в работата на системата.

Една от ключовите характеристики на OSGi е динамичната управляемост. Модулите могат да бъдат активирани, деактивирани и обновявани по време на работа на системата,

без да се налага прекомпилиране или рестартиране на цялото приложение. Това позволява на системите да бъдат по-гъвкави и лесни за управление.

В контекста на проекта, OSGi играе важна роля в създаването на софтуерната архитектура. Той позволява лесната интеграция и управление на различни модули, включително сензори, устройства и логически компоненти. OSGi осигурява възможността за създаване на модулни приложения, които се адаптират към нуждите на системата и позволяват лесното добавяне на нови функционалности и разширения.

OSGi играе ключова роля в IoT проектите, където гъвкавостта, динамичното управление на компонентите и лесната интеграция с различни устройства и сензори са от изключително значение. Този стандарт подкрепя създаването на софтуерни решения, които могат да бъдат непрекъснато актуализирани и развивани, в съответствие с променящите се нужди и изисквания на приложението.

Spring Framework

Spring Spring Framework е широко използван и мощен Java фреймуърк, предназначен за създаване на скалабилни и ефективни приложения. Този фреймуърк предоставя набор от модули и инструменти, които облекчават процеса на разработка и поддържка на приложения с различни изисквания. Ето някои от ключовите характеристики и предимства на Spring:

Spring предоставя инверсия на контрола и внедряване на зависимости (IoC / DI), което допринася за по-слабо свързани и по-тестваеми компоненти. Това подпомага разделянето на отговорностите и лесното разширяване на системата.

Фреймуъркът предоставя богата библиотека от готови модули и компоненти, които ускоряват разработката. Spring Boot, например, позволява бързо създаване на стартови приложения с минимална конфигурация.

С оглед на разработката на системата "Сензора като услуга" в IoT, Spring Framework би бил идеален избор за осигуряване на надеждна и скалабилна архитектура на приложението. Този фреймуърк предоставя интеграция с различни технологии и лесно управление на компонентите, което допринася за успешната реализация на проекта.

Angular

Angular е съвременен фронтенд фреймуърк, създаден от екипа на Google, предоставящ мощни инструменти за създаване на динамични уеб приложения. Този фреймуърк е изключително подходящ за проекта "Сензора като услуга" в екосистемата на IoT.

Angular разчита на компонентна архитектура, която допринася за яснотата и поддръжката на кода. Приложението се състои от множество независими компоненти, които могат да бъдат лесно преизползвани и разширявани.

Този фреймуърк поддържа двустранна връзка на данните, което означава, че промените в модела автоматично се отразяват върху потребителския интерфейс и обратно. Това улеснява работата с данни и осигурява динамичен потребителски опит. Angular предоставя механизми за ефективно управление на формите и валидацията. Това е от съществено значение за обработката на данни и взаимодействието с потребителите. Инструментът Angular CLI предоставя лесен и бърз начин за създаване на нови компоненти, модули и услуги. Това значително ускорява разработката и осигурява консистентна структура на приложението.

С Angular може да се изграждат динамични рутиращи системи, което подпомага навигацията между различните страници и компоненти на приложението. Фреймуъркът осигурява ефективно управление на състоянието на приложението и обмена на данни със сървър чрез HTTP заявки. Angular се грижи за съвместимостта на приложението с различни браузъри и устройства, което осигурява семпло и непрекъснато потребителско изживяване.

С обширната си общност и богат набор от ресурси, Angular предоставя множество документации, ръководства и примери, които помагат на разработчиците да се ориентират и да решават предизвикателствата по време на проекта.

Zigbee протокол

Zigbee е безжичен комуникационен протокол, разработен с оглед на осигуряване на ниска консумация на енергия, дълъг живот на батериите и надеждна безжична връзка между IoT устройствата. Този протокол е идеален за сценарии, които изискват дългосрочна автономност на устройствата, като например умни домове, индустриални системи и здравеопазване.

Основната сила на Zigbee се крие в неговата способност да осигурява ниска консумация на енергия, което позволява на устройствата да работят за дълго време с една батерия или дори соларни клетки. Този аспект го прави подходящ за приложения, където достъпът до устройствата е ограничен или труден, като например датчици разположени на труднодостъпни места.

Zigbee използва радиочестотните обхвати от 2.4 GHz, 868 MHz и 915 MHz, което му позволява да предоставя стабилна и надеждна комуникация дори през препятствия и на дълги разстояния. Протоколът използва модел на мрежова архитектура, където устройствата се групират във "клъстери" и образуват мрежи от тези клъстери, което

позволява на устройствата да комуникират помежду си и създават разнообразни сценарии за взаимодействие.

Този протокол предоставя сигурност чрез криптиране на комуникацията и идентификация на устройствата, което го прави подходящ за приложения, които изискват повишена защита на данните, като например системи за управление на достъпа или мониторинг на здравни данни.

MQTT протокол (Message Queuing Telemetry Transport)

MQTT (Message Queuing Telemetry Transport) е лек и ефективен протокол за комуникация между устройства и приложения в Интернет на нещата (IoT) екосистемата. Той е проектиран с оглед на ниската консумация на бандова ширина и ниска латентност на съобщенията, което го прави подходящ за условия с ограничени ресурси, като вградени системи и мобилни устройства.

MQTT използва модел на публикуване-абониране, където устройствата, наречени публикатори, публикуват съобщения на теми (topics), а други устройства, наречени абонати, се абонират за тези теми, за да получават съобщенията. Този модел позволява ефективно разпространение на информация и удобство в мащабиращите се IoT системи.

Основните характеристики и предимства на MQTT включват лек и ефективен дизайн, нисколатентна комуникация, модел на публикуване-абониране, контрол на качеството на услугата (QoS), сигурност чрез TLS/SSL протоколи, интеграция в различни мрежови архитектури.

MQTT е широко използван в IoT проекти, където се изисква надеждна и ефективна комуникация между разнообразни устройства и компоненти. Той е подходящ за мониторинг, управление, контрол и събиране на данни от различни IoT устройства, като сензори, актуатори и други.

Zigbee2MQTT

Zigbee2MQTT е важен софтуерен компонент, който играе ключова роля в реализацията на IoT приложения, основани на Zigbee протокола. Този софтуер позволява на различни устройства и сензори, използващи Zigbee комуникация, да бъдат интегрирани в една обща система и да се управляват чрез MQTT (Message Queuing Telemetry Transport) протокола.

Zigbee2MQTT действа като мост между Zigbee устройствата и MQTT брокера. Той преобразува съобщенията, получени от Zigbee устройствата, в MQTT съобщения, които могат да бъдат публикувани в теми и абонирани от други устройства или системи. Това позволява на различни устройства да обменят информация и команди в единния IoT екосистема.

Една от основните предимства на Zigbee2MQTT е, че той предоставя възможност за използване на голям брой различни Zigbee устройства от различни производители. Този софтуер поддържа широк спектър от устройства, като сензори за движение, температура, осветеност, сензори за контакт и други. Това прави възможно създаването на комплексни IoT приложения, които включват множество типове сензори и устройства.

Софтуерът също така предоставя възможности за конфигурация и управление на устройствата чрез удобен интерфейс. Потребителите могат да настройват параметрите на всяко устройство, както и да изпращат команди за контрол и манипулация. Това прави управлението на IoT системата по-лесно и достъпно.

Сигурността е също важен аспект на Zigbee2MQTT. Софтуерът използва различни мерки за сигурност, за да осигури защита на данните и комуникацията между устройствата. Това е от съществено значение, особено при IoT приложения, където се обработват чувствителни данни.

Zigbee2MQTT е мощен софтуерен инструмент, който позволява интеграцията на разнообразни Zigbee устройства в IoT приложения. Неговата способност да свързва различни устройства и да ги управлява чрез MQTT комуникация предоставя гъвкавост и възможности за разработка на разнообразни IoT сценарии.

Mosqutto

Mosquitto представлява мощен брокер за съобщения, проектиран да обслужва комуникацията в IoT (Интернет на нещата) среда. Той поддържа протокола MQTT (Message Queuing Telemetry Transport), който осигурява лек, но надежден начин за обмен на данни между устройства, сензори и приложения. Този брокер е от съществено значение за изграждането на устойчиви и ефективни IoT системи.

Mosquitto предоставя два ключови модела за комуникация - модела на публикуване-абониране и директната точка-точка комуникация. Този гъвкав подход позволява на различни устройства и приложения да обменят данни по интересуващи ги теми, като се поддържа лесна скалируемост на системата. Брокерът работи с множество клиенти, които изпращат и получават съобщения, докато същевременно осигурява ниска латентност и ефективност на комуникацията.

Важен аспект на Mosquitto е сигурността. Той поддържа механизми за аутентикация и упълномощаване, които осигуряват контролиран достъп до данните. Също така, Mosquitto предоставя възможности за шифроване на данните, което осигурява поверителност и неразривност на информацията при преноса й. Тези функционалности са от съществено значение, особено когато става дума за IoT системи, които обработват лични и чувствителни данни.

В реализирания проект Mosquitto играе ключова роля в комуникацията между различни компоненти на системата - сензори, устройства и приложения. Този брокер предоставя

надеждна и ефективна инфраструктура за обмяна на данни, като осигурява гъвкавост и мащабируемост. Сигурността, предоставяна от Mosquitto, гарантира защитата на данните и поверителността, което е от критично значение за успешната работа на IoT системи, особено в контекста на нарастващите заплахи в онлайн средата.

Docker

Docker Docker е популярна технология за контейнеризация, която предоставя среда за изолиране и управление на приложения и техните зависимости. Тя играе важна роля в разработката и доставката на софтуерни приложения, включително и в IoT проекти.

С Docker, приложенията и техните компоненти се опаковат в контейнери, които включват всичко необходимо за тяхното изпълнение - операционна система, библиотеки, код и настройки. Това позволява лесно преносимост на приложенията между различни среди - от разработка до тестване, продукция и обратно.

В контекста на IoT проекти, Docker може да бъде използван за опаковане на софтуерни компоненти, които се използват на ресурсно ограничени устройства или в разпределени системи. Например, IoT устройства могат да работят с по-леки и оптимизирани версии на операционни системи, които са опаковани в Docker контейнери. Това улеснява управлението и обновлението на устройствата, без да се нарушава функционалността им.

Друга полза на Docker е възможността за създаване на микросервизни архитектури, където различни компоненти на приложението се изолират в контейнери. Това позволява лесна мащабируемост, управление и поддръжка на системата. В IoT проекти този подход може да се използва, например, за създаване на модулни системи, в които различни функционалности се изпълняват в отделни контейнери.

Също така, Docker предоставя инструменти за автоматизация на развойния цикъл на приложението. Това включва дефиниране на инфраструктурата и настройките в код (Infrastructure as Code), както и автоматизирано тестване, сборка и доставка. Това улеснява процеса на разработка и интеграция на нови функционалности в IoT приложения.

Използвайки тези технологии и инструменти, проектирахме гъвкава и мащабируема архитектура, която да улесни събирането, управлението и анализа на данни от разнообразни сензори и устройства. Системата може да се интегрира с множество различни сензори, включително устройства за измерване на движение на вода, осветеност, температура и други.

Проектираната архитектура позволява на потребителите да лесно контролират и управляват своите сензори чрез потребителския интерфейс, изграден с помощта на Angular. MQTT протоколът осигурява ефективното изпращане и получаване на данни от

сензорите, като моста Zigbee2MQTT допълнително предоставя възможност за комуникация с Zigbee устройствата. OSGi архитектурата позволява на системата да бъде динамично разширяема и приспособима към бъдещи нужди и развития.

2.3. Архитектурен план на проектираната система

Проектирането на архитектурния план на система е съществено за осигуряването на функционалността, производителността и устойчивостта й. Този процес включва анализ на взаимодействията между различните компоненти, определяне на техните роли и отговорности, както и изграждането на подходящи структури и модели.

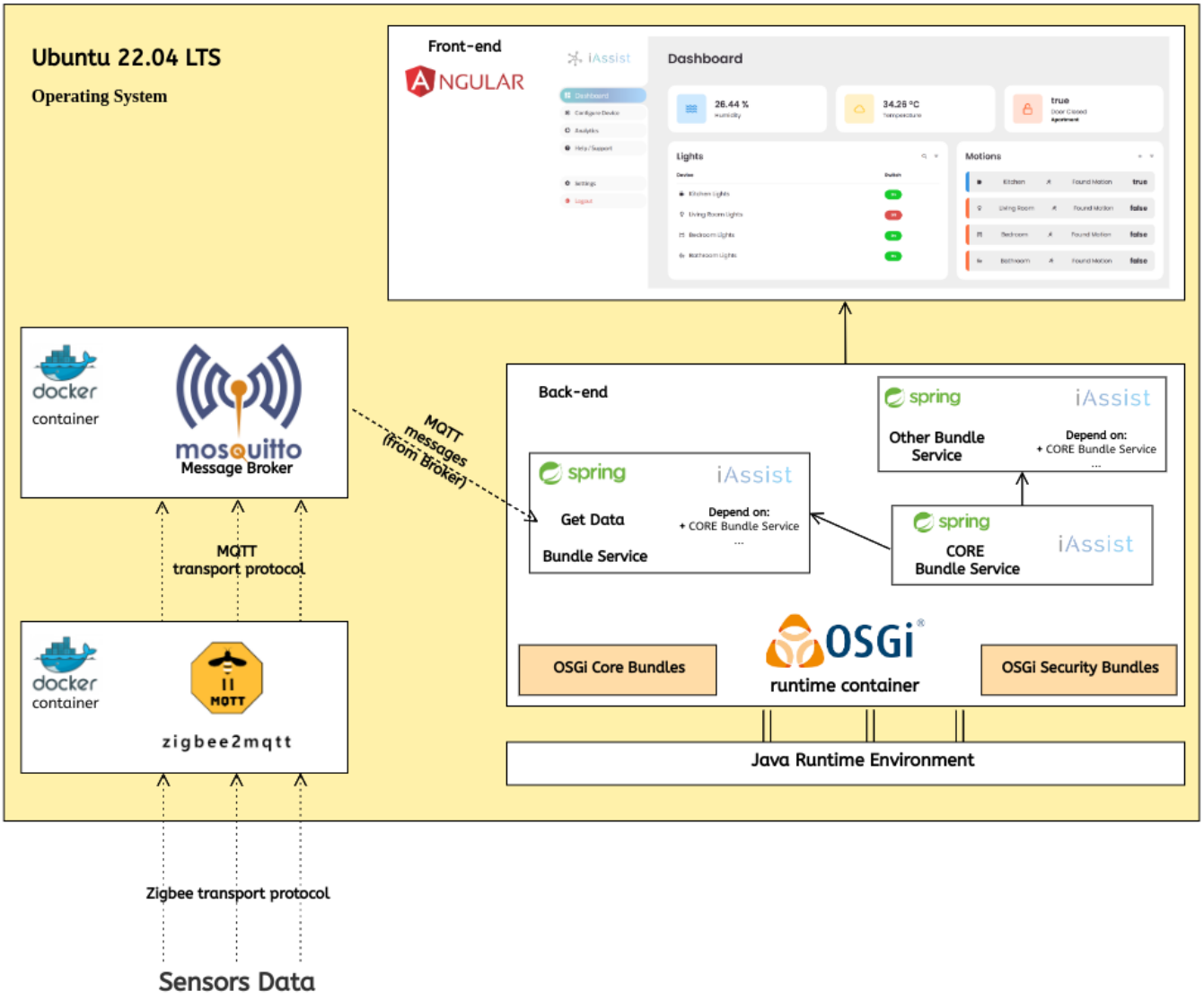
Проектът за "Сензора като услуга" в екосистемата на IoT изисква добре структурирана архитектура, която да поддържа различни видове сензори и устройства, да осигурява комуникацията между тях и да предоставя анализ и визуализация на данните. Целта на архитектурния план е да гарантира ефективно и надеждно функциониране на системата, като същевременно я прави гъвкава и разширяема за бъдещи изисквания.

Със съчетаването на OSGi, Spring и Angular, ние изграждаме модерна и интегрирана инфраструктура, която може да управлява различни аспекти на системата. OSGi ни предоставя средствата за модулност и динамичност, позволявайки ни да добавяме нови функционалности или да модифицираме съществуващите без да се налага спиране на цялата система.

Spring Framework подпомага разработката на бизнес логиката и услугите на системата, като предоставя инверсия на управлението и спрягане на компонентите. Този фреймуърк улеснява интеграцията на външни услуги и бази данни, както и изграждането на стабилни и мащабируеми приложения.

Angular, от своя страна, предоставя средства за създаване на потребителски интерфейси, които да предоставят богат опит на потребителите. Този съвременен фреймуърк позволява създаването на динамични и атрактивни уеб приложения, които да работят ефективно на различни устройства и браузъри.

Интеграцията на MQTT и Zigbee2MQTT позволява създаването на мощна система за комуникация между устройствата и сензорите. Този протокол е основен за предаване на данни в реално време и осигурява надеждна и сигурна връзка между компонентите.



С общата им работа, тези технологии и инструменти създават архитектурен план, който е съобразен с изискванията на проекта за "Сензора като услуга" в IoT екосистемата. Този план поддържа баланс между функционалност, производителност и разширяемост, осигурявайки стабилна основа за реализацията на системата.

Глава 3

Развитие на софтуера

3.1. Подготовка на средата

Първата стъпка преди самата разработка на проекта, е да се подготви средата, която ще позволи лесно създаване, управление и тестване на системата. За целта, трябва да се конфугурира и стартира SONOFF ZBDongle-P Zigbee 3.0, а след това да се свържш интеграцията между сензорите, ZBDongle-a и контролера, т.е. компютъра.

Стартиране на SONOFF ZBDongle-P Zigbee 3.0

За конфигурирането и стартирането на ZBDongle-a, може да бъдат следвани стъпките:

- Включва се ZBDongle устройството, чрез USB, в компютъра.
- Създава се нова папка в папката '**/opt**', която да носи името zigbee2mqtt, в която ще бъде инсталиран софтуерният компонент Zigbee2MQTT:

```
~$ cd /opt
~$ sudo mkdir zigbee2mqtt
~$ cd zigbee2mqtt
```

- Сваляне на нужните конфигурации за Zigbee2MQTT, които ще бъдат записани във файла '**configuration.yaml**':

```
~$ sudo wget https://raw.githubusercontent.com/Koenkk/zigbee2mqtt/master/data/
configuration.yaml -P /opt/zigbe2mqtt
```

- Отваря се отново конфигурационния файл '**configuration.yaml**' на Zigbee2MQTT компонента и се променя неговото съдържание по посочения по-долу начин:

```
~$ cd /opt/zigbee2mqtt
~$ sudo vi configuration.yaml
```

Съдържание на '**/opt/zigbee2mqtt/configuration.yaml**':

```
# Enable Ziggbee2MQTT frontend to run on http://localhost:8082/
frontend: true

# Allow new devices to join
permit_join: true

# MQTT settings
mqtt:
  # MQTT base topic for zigbee2mqtt MQTT messages
  base_topic: zigbee2mqtt
  # MQTT server URL
  server: mqtt://<IP-address-of-the-computer>
  # MQTT server authentication, uncomment if required:
  # user: my_user
  # password: my_password

# Serial settings
serial:
  # Location of CC2531 USB sniffer
  port: /dev/ttyACM0
```

IP адреса на компятътра, който трябва да бъде вписан в '**<IP-address-of-the-computer>**', може да бъде изведен със следната команда:

```
~$ ifconfig
```

Като неговата стойност, може да бъде намерена под '**wlo1:**', отбелязана с '**inet**' таг пред самият адрес.

- На компютъра се отваря папката '**/dev/serial/by-id/**' и се копира името на вече свързаното ZBDongle устройството, което има формата:
  usb-ITEAD_SONOFF_Zigbee_3.0_USB_Dongle_Plus_V2_20221130094342-if00

- Стартиране на софтуерният компонент Zigbee2MQTT, като docker контейнер:
  Забележка: '**--device**' параметърът трябва да сочи към името на ZBDongle устройството от предходната инструкция, а '**/dev/ttyACM0**' е местоположение на CC2531 USB снифера, конфугъруран в '**/opt/zigbee2mqtt/configuration.yaml**'

```
~$ cd /opt/zigbee2mqtt
~$ docker run \
   --name zigbee2mqtt \
   --restart=unless-stopped \
```

```
--device=/dev/serial/by-id/usb-ITEAD_SONOFF_Zigbee_3.0_USB_Dongle_Plus_V2_20221130094342
-if00:/dev/ttyACM0 \
   -p 8080:8080 \
   -v $(pwd)/data:/app/data \
   -v /run/udev:/run/udev:ro \
   -e TZ=Europe/Amsterdam \
   koenkk/zigbee2mqtt
```

След успешно стартиране на Zigbee2MQTT docker контейнера, ще бъде визуализирана следната информация:

```
root@dess:/opt/zigbee2mqtt#  docker run \
>    --name zigbee2mqtt \
>    --restart=unless-stopped \
>    --device=/dev/serial/by-id/usb-ITEAD_SONOFF_Zigbee_3.0_USB_Dongle_Plus_V2_20221130094342-if00:/dev/ttyACM0 \
>    -p 8080:8080 \
>    -v $(pwd)/data:/app/data \
>    -v /run/udev:/run/udev:ro \
>    -e TZ=Europe/Amsterdam \
>    koenkk/zigbee2mqtt
Using '/app/data' as data directory
Zigbee2MQTT:info  2023-05-26 07:17:20: Logging to console and directory: '/app/data/log/2023-05-26.07-17-20' filename: log.txt
Zigbee2MQTT:info  2023-05-26 07:17:20: Starting Zigbee2MQTT version 1.30.4 (commit #b2dd21e)
Zigbee2MQTT:info  2023-05-26 07:17:20: Starting zigbee-herdsman (0.14.111)
Zigbee2MQTT:info  2023-05-26 07:17:25: zigbee-herdsman started (resumed)
Zigbee2MQTT:info  2023-05-26 07:17:25: Coordinator firmware version: '{"meta":{"maintrel":"3 ","majorrel":"6","minorrel":"10",
"product":8,"revision":"6.10.3.0 build 297"},"type":"EZSP v8"}'
Zigbee2MQTT:info  2023-05-26 07:17:25: Currently 0 devices are joined:
Zigbee2MQTT:warn  2023-05-26 07:17:25: `permit_join` set to  `true` in configuration.yaml.
Zigbee2MQTT:warn  2023-05-26 07:17:25: Allowing new devices to join.
Zigbee2MQTT:warn  2023-05-26 07:17:25: Set `permit_join` to `false` once you joined all devices.
Zigbee2MQTT:info  2023-05-26 07:17:25: Zigbee: allowing new devices to join.
Zigbee2MQTT:info  2023-05-26 07:17:26: Connecting to MQTT server at mqtt:// <IP-address-to-computer>
```

Забележка: При проблем със стартирането на контейнера, може да е е необходима промяна по конфигурациите на firewall-а на машината или неговото спиране.

- Стартиране на софтуерният компонент Mosquitto, като docker контейнер:

```
~$ mkdir /opt/mosquitto
~$ cd /opt/mosquitto
~$ docker run -it -p 1883:1883 -v /opt/mosquitto:/mosquitto/ -v
/opt/mosquitto/config/mosquitto.conf:/mosquitto/config/mosquitto.conf -v
/opt/mosquitto/log:/mosquitto/log -v /opt/mosquitto/data:/mosquitto/data
eclipse-mosquitto mosquitto -c /mosquitto-no-auth.conf
```

Следва да променим и съдържанието на '**mosquitto.conf**' файла на Mosquitto компонента по посочения по-долу начин:

```
~$ cd /opt/mosquitto/config
~$ sudo vi mosquitto.conf
```

Съдържание на '**/opt/mosquitto/config/mosquitto.conf**':

```
# Config file for mosquitto
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log
log_type all
listener 1883
allow_anonymous true
```

Стартиране на сензорите

След успешното стартиране и конфигуриране на ZBDongle-a, втората стъпка по подготовката на средата е да се свържат и самите сензори. В документацията на всеки от производителите на сензорите, е обозначено как се стартират самите сензори. В повечето случаи, сензорите имат малки бутончета, които тряба да се натиснат и да се задържат за 5 секунди, след това в самият сензор започва да мига червена лампичка, като индикатор, че сензорът е готов да се свърже към подготвената сред. Тогава на конзолата къде работи Mosquitto контейнера, трябва да се видят логове, подобни на тези:

```
root@dess:/opt/mosquitto/config# docker run -it -p 1883:1883 -v /opt/mosquitto:/mosquitto/ -v /opt/mosquitto/config/mosquitto.conf:/mosquitto/config/
mosquitto.conf -v /opt/mosquitto/log:/mosquitto/log -v /opt/mosquitto/data:/mosquitto/data  eclipse-mosquitto mosquitto -c /mosquitto-no-auth.conf
1685170406: mosquitto version 2.0.15 starting
1685170406: Config loaded from /mosquitto-no-auth.conf.
1685170406: Opening ipv4 listen socket on port 1883.
1685170406: Opening ipv6 listen socket on port 1883.
1685170406: mosquitto version 2.0.15 running
1685170407: New connection from 172.17.0.1:34354 on port 1883.
1685170407: New client connected from 172.17.0.1:34354 as auto-5A1F8E48-D7C1-131C-5BF6-3D9225E6CB28 (p2, c1, k60).
1685170434: New connection from 172.17.0.1:43696 on port 1883.
1685170434: New client connected from 172.17.0.1:43696 as auto-9EB896BE-6506-3898-BE2C-C7A279548510 (p2, c1, k60).
```

Следва да променим и съдържанието на '**configuration.yaml**' на Zigbee2MQTT компонента по посочения по-долу начин:

```
~$ cd /opt/zigbee2mqtt
~$ sudo vi configuration.yaml
```

Съдържание на '**/opt/zigbee2mqtt/configuration.yaml**':

```
# Enable Ziggbee2MQTT frontend to run on http://localhost:8082/
frontend: true

# Allow new devices to join
permit_join: true

# MQTT settings
mqtt:
```

```
  # MQTT base topic for zigbee2mqtt MQTT messages
  base_topic: zigbee2mqtt
  # MQTT server URL
  server: mqtt://<IP-address-of-the-computer>
  # MQTT server authentication, uncomment if required:
  user: user
  password: pass

# Serial settings
serial:
  # Location of CC2531 USB sniffer
  port: /dev/ttyACM0
device_options:
  legacy: false
devices:
  '0xa4c138a11f3bb89e':
    friendly_name: Tuya Temperature Sensor
  '0x00124b002934f4c4':
    friendly_name: Sonoff Contact Sensor
  '0x00124b0029323515':
    friendly_name: Sonoff Motion Sensor
```

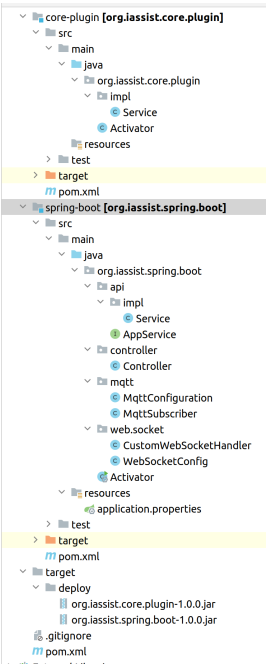
На конзолата къде работи Zigbee2MQTT контейнера, трябва да се видят логове, подобни на тези, които показват, че сензорите са успшено свързани:

```
zigbee2mqtt  | Zigbee2MQTT:info  2023-08-20 16:40:00: Currently 3 devices are joined:
zigbee2mqtt  | Zigbee2MQTT:info  2023-08-20 16:40:00: Tuya Temperature Sensor (0xa4c138a11f3bb89e): WSD500A - TuYa Temperature & humidity sensor (EndDevice)
zigbee2mqtt  | Zigbee2MQTT:info  2023-08-20 16:40:00: Sonoff Contact Sensor (0x00124b002934f4c4): SNZB-04 - SONOFF Contact sensor (EndDevice)
zigbee2mqtt  | Zigbee2MQTT:info  2023-08-20 16:40:00: Sonoff Motion Sensor (0x00124b0029323515): SNZB-03 - SONOFF Motion sensor (EndDevice)
```

3.2. Изграждане и разработка

В този раздел ще се фокусираме върху процеса на изграждане и разработка на реализираната система, който е етап от ключово значение за постигане на целите на проекта. Контейнеризацията на OSGi платформата позволява разработването на отделните части на софтуера, като ги изолира по такъв начин, че при ъпдейт на версия на дадена функционалност, или добавянето на нова такава, т.н. бъндали (bundle), не е необходимо рестартирането на цялата система.

Ето защо, за да бъде демонстрирано това, ще бъде създаден един основен Core Spring Bundle, които ще е базов, като основа за цялостното приложение. Ще бъде добавен и втори Get Data Bundle, чиято цел ще е да чете данните сензорите, посредством MQTT Message Broker, а след това да ги слага в Web Socket, откъдето Angular компонента, ще може да ги взима и да ги визуализира. Така разработена системата предоставя възможност, ако например бъде добавен нов Data Analysis Bundle, които има за цел да прави анализ върху данните от сензорите, или какъвто и да е друг бъндал, той просто да бъде инсталиран в runtime-а на OSGi, без да растартираме цялата платформа.



Проекта би има следната структура с описание за функцията на важните му компоненти, класове и конфигурации:

- **core-plugin [org.iassist.core.plugin]** - основният Core Spring Bundle
  - **Activator.java** - класът, който стартира bundle-а в OSGi runtime.
  - **pom.xml** - Maven файлът отговорен за създаването на JAR файл, които изпълнява роклята и на bundle (за Core Spring Bundle-а).

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <artifactId>org.iassist.core.plugin</artifactId>
   <parent>
      <groupId>org.iassist</groupId>
      <artifactId>org.iassist</artifactId>
      <version>1.0.0</version>
   </parent>
   <packaging>bundle</packaging>
   <dependencies>
      <dependency>
         <groupId>org.osgi</groupId>
         <artifactId>org.osgi.core</artifactId>
         <version>${osgi.version}</version>
         <scope>provided</scope>
      </dependency>
      <dependency>
         <groupId>org.iassist</groupId>
         <artifactId>spring.boot</artifactId>
         <version>${project.version}</version>
```

```
        <scope>provided</scope>
      </dependency>
      <dependency>
        <groupId>org.glassfish</groupId>
        <artifactId>jsonp-jaxrs-1x</artifactId>
        <version>2.0.0-RC3</version>
      </dependency>
   </dependencies>
   <build>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-compiler-plugin</artifactId>
          <configuration>
            <source>1.8</source>
            <target>1.8</target>
          </configuration>
        </plugin>
        <plugin>
          <groupId>org.apache.felix</groupId>
          <artifactId>maven-bundle-plugin</artifactId>
          <version>${bundle.plugin.version}</version>
          <extensions>true</extensions>
          <configuration>
            <instructions>
              <Bundle-SymbolicName>
                ${project.artifactId}
              </Bundle-SymbolicName>
              <Bundle-Name>${project.name}</Bundle-Name>
              <Bundle-Version>${project.version}</Bundle-Version>
              <Bundle-Activator>${project.artifactId}.Activator</Bundle-Activator>
              <Private-Package>${project.artifactId}.*</Private-Package>
              <Import-Package>*;resolution:=optional</Import-Package>
              <Embed-Dependency>*;scope=compile</Embed-Dependency>
              <Embed-Transitive>true</Embed-Transitive>
            </instructions>
            <buildDirectory>target/deploy</buildDirectory>
          </configuration>
        </plugin>
      </plugins>
   </build>
</project>
```

- **spring-boot [org.iassist.spring.boot]** - това е Get Data Bundle, които отговаря за събирането на данните от сензорите
  - **controller** - съдържа REST заявките, които Angular front-end-а би използвал.
  - **mqqt** - съдържа логиката и конфигурацията, която позволява връзката с MQTT Message Broker-а.
    - **MqttConfiguration.java** - този клас съдържа методи и параметри за конфигурирането на MQTT Message Broker-а (клиента).

```
package org.iassist.spring.boot.mqtt;

import …;

@Configuration
public class MqttConfiguration {
```

```
    @Value("${mqtt.brokerUrl}")
    private String brokerUrl;

    @Value("${mqtt.clientId}")
    private String clientId;

    @Value("${mqtt.username}")
    private String username;

    @Value("${mqtt.password}")
    private String password;


    // Getters and setters for this class


    @PostConstruct
    public void init() {
        System.out.println("Initializing MQTT Configuration...");
        System.out.println("Broker URL: " + brokerUrl);
        System.out.println("Client ID: " + clientId);
        System.out.println("Username: " + username);
        System.out.println("Password: " + password);
    }

    @Bean
    public MqttConnectOptions configureMqttClient() {
        MqttConnectOptions options = new MqttConnectOptions();
        options.setServerURIs(new String[] { brokerUrl });
        options.setUserName(username);
        options.setPassword(password.toCharArray());
        options.setCleanSession(true);
        options.setAutomaticReconnect(true);
        return options;
    }
}
```

- **MqttSubscriber.java** - този клас съдържа методи, които позволяват връзването на back-end-а към MQTT Message Broker-а.

```
package org.iassist.spring.boot.mqtt;

import …;

@Component
public class MqttSubscriber {

    private static final Logger LOG = LoggerFactory.getLogger(MqttSubscriber.class);

    private MqttConnectOptions connOpts;
    private MqttAsyncClient mqttClient;
    private String receivedPayload;

    @Autowired
    private final SimpMessagingTemplate messagingTemplate;

    @Autowired
    private final MqttConfiguration mqttConf;
```

```
@Autowired
public MqttSubscriber(MqttConfiguration mqttConf, SimpMessagingTemplate messagingTemplate) {
    this.mqttConf = mqttConf;
    this.messagingTemplate = messagingTemplate;
}

// Getters and setters for this class

public void configureMqttClient() {
    try {
        LOG.info("Starting the MQTT Configuration...");
        connOpts = mqttConf.configureMqttClient();
        mqttClient = new MqttAsyncClient(
                    mqttConf.getBrokerUrl(), mqttConf.getClientId(), new MemoryPersistence());

        connOpts = new MqttConnectOptions();
        connOpts.setCleanSession(true);
        connOpts.setUserName(mqttConf.getUsername());
        connOpts.setPassword(mqttConf.getPassword().toCharArray());
        connOpts.setAutomaticReconnect(true);

        LOG.info("The Mqtt protocol has been configured successfully!!!");
    }
    catch (Exception ex) {
        LOG.error("An error has happened during Mqtt configuration: " + ex);
    }
}

public void connectMqttClient() {
    try {
        // Connect to MQTT client
         LOG.info("Connecting to '{}' MQTT broker", mqttConf.getBrokerUrl());
        IMqttToken token = mqttClient.connect(connOpts);
        token.waitForCompletion();
        LOG.info("Connected to '{}' MQTT broker", mqttConf.getBrokerUrl());

    } catch(MqttException ex) {
        LOG.error("An error has happened: " + ex);
        LOG.error("\nMessage: " + ex.getMessage()
                + "\nLocalized Message: " + ex.getLocalizedMessage()
                + "\nCause: " + ex.getCause());
    }
}

public void subscribeToTopics(List<String> topics) {
    try {
        mqttClient.setCallback(new MqttCallback() {
            @Override
            public void connectionLost(Throwable throwable) {
                System.out.println("Connection lost to MQTT broker: " + throwable);
            }

            @Override
            public void messageArrived(String topic, MqttMessage mqttMessage) {
                String payload = new String(mqttMessage.getPayload());
                System.out.println("Received MQTT message from '" + topic + "': " + payload);

                // Parse the topic to determine the sensor type
                 String sensorType = topic.split("/")[1].toLowerCase();

                // Broadcast the payload to WebSocket clients based on the sensor type
                 messagingTemplate.convertAndSend("/topic/" + sensorType, payload);
                receivedPayload = payload;
            }
```

```
                @Override
                public void deliveryComplete(IMqttDeliveryToken iMqttDeliveryToken) {
                    System.out.println("Message delivered successfully");
                }
            });

            for (String topic : topics) {
                LOG.info("Subscribing to the topic '{}'", topic);
                mqttClient.subscribe(topic,1);  // Quality of Service level (0, 1, or 2)
                 LOG.info("Successful subscribed to topic '{}'", topic);
            }

        } catch(MqttException ex) {
            LOG.error("An error has happened: " + ex);
            LOG.error("\nMessage: " + ex.getMessage()
                    + "\nLocalized Message: " + ex.getLocalizedMessage()
                    + "\nCause: " + ex.getCause());
        }
    }
}
```

➢ **web.socket**

- **CustomWebSocketHandler.java** - този клас отговаря за логиката, която ще свързва web socket-а, откъдето front-end-а ще чете данните от сензорите.

```
package org.iassist.spring.boot.web.socket;

import …;

@Component
public class CustomWebSocketHandler implements WebSocketHandler {

   private final List<WebSocketSession> sessions = new ArrayList<>();

   @Override
   public void afterConnectionEstablished(WebSocketSession session) throws Exception {
       sessions.add(session);
       System.out.println("WebSocket connection established: " + session.getId());
   }

   @Override
   public void handleMessage(WebSocketSession session, WebSocketMessage<?> message) throws Exception {
       String payload = message.getPayload().toString();
       System.out.println("Received message from " + session.getId() + ": " + payload);

       // Broadcast the message to all connected clients
       for (WebSocketSession s : sessions) {
           try {
               s.sendMessage(new TextMessage(payload));
           } catch (IOException e) {
               e.printStackTrace();
           }
       }
   }

   @Override
   public void afterConnectionClosed(WebSocketSession session, CloseStatus closeStatus) throws Exception {
       sessions.remove(session);
       System.out.println("WebSocket connection closed: " + session.getId());
   }
```

```
    @Override
    public void handleTransportError(WebSocketSession session, Throwable exception) throws Exception {
        System.err.println("WebSocket transport error: " + exception.getMessage());
    }

    @Override
    public boolean supportsPartialMessages() {
        return false;
    }
}
```

- **WebSocketConfig.java** - този клас отговаря за задаването на необходимата конфигурация web socket-а.

```
package org.iassist.spring.boot.web.socket;

import …;

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.enableSimpleBroker("/topic");
        registry.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/ws")
                .setAllowedOriginPatterns("*")
                .withSockJS();
    }
}
```

➢ **Activator.java** - този клас отговаря за стартирането на софтуерния проект, както локално така и в OSGi runtime-а като bundle.

```
package org.iassist.spring.boot;

import …;

@SpringBootApplication(exclude = SecurityAutoConfiguration.class)
@EnableWebSocket
public class Activator implements BundleActivator {

    private static BundleContext bundleContext;
    private ConfigurableApplicationContext appContext;

    private List<String> allTopics = Arrays.asList(
            "zigbee2mqtt/Tuya Temperature Sensor",
            "zigbee2mqtt/Sonoff Contact Sensor",
            "zigbee2mqtt/Sonoff Motion Sensor");

    @Autowired
    private MqttSubscriber mqttSubscriber;

    @Override
```

```
    public void start(BundleContext context) {
        Thread.currentThread().setContextClassLoader(this.getClass().getClassLoader());

        // Start the Spring Boot application context
        appContext = SpringApplication.run(Activator.class);
        bundleContext = context;

        // Register OSGi services
        registerServices();

        mqttSubscriber.configureMqttClient();
        mqttSubscriber.connectMqttClient();
        mqttSubscriber.subscribeToTopics(allTopics);
    }

    @Override
    public void stop(BundleContext context) {
        bundleContext = null;
        // Close the Spring Boot application context
        SpringApplication.exit(appContext, () -> 0);
    }

    private void registerServices() {
        AppService service = new Service();
        bundleContext.registerService(AppService.class.getName(), service, new Hashtable<>());
        System.out.println("Service registered: " + service.name());
    }

    public static void main(String[] args) {
        // Only use this main method if running outside OSGi
        ConfigurableApplicationContext appContext = SpringApplication.run(Activator.class, args);
        MqttSubscriber mqttSubscriber = appContext.getBean(MqttSubscriber.class);

        mqttSubscriber.configureMqttClient();
        mqttSubscriber.connectMqttClient();

        List<String> allTopics = Arrays.asList(
                "zigbee2mqtt/Tuya Temperature Sensor",
                "zigbee2mqtt/Sonoff Contact Sensor",
                "zigbee2mqtt/Sonoff Motion Sensor");
        mqttSubscriber.subscribeToTopics(allTopics);
    }
}
```

➢ **resources/application.properties** - съдържа конфигурации за проекта, заедно с конфигурация за MQTT от '**/opt/zigbee2mqtt/configuration.yaml**':

```
# Spring Boot app configurations
server.port=8081
spring.main.allow-bean-definition-overriding=true


#MQTT broker configurations
mqtt.brokerUrl=tcp://<IP-address-of-the-computer>:1883
mqtt.clientId=mqtt-subscribe
mqtt.username=user
mqtt.password=pass
```

➢ **pom.xml** - Maven файлът отговорен за създаването на JAR файл, които изпълнява ролята и на bundle (за Get Data Bundle-a).

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <artifactId>org.iassist.spring.boot</artifactId>
   <parent>
      <groupId>org.iassist</groupId>
      <artifactId>org.iassist</artifactId>
      <version>1.0.0</version>
   </parent>
   <packaging>bundle</packaging>
   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-actuator</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-jetty</artifactId>
      </dependency>
      <dependency>
         <groupId>org.osgi</groupId>
         <artifactId>osgi.core</artifactId>
         <version>${osgi.version}</version>
         <scope>provided</scope>
      </dependency>
      <dependency>
         <groupId>org.osgi</groupId>
         <artifactId>org.osgi.service.component.annotations</artifactId>
         <version>1.5.1</version>
         <scope>provided</scope>
      </dependency>
      <dependency>
         <groupId>org.osgi</groupId>
         <artifactId>org.osgi.service.event</artifactId>
         <version>1.4.1</version>
         <scope>provided</scope>
      </dependency>
      <dependency>
         <groupId>org.eclipse.paho</groupId>
         <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
         <version>1.2.5</version>
      </dependency>
      <dependency>
         <groupId>org.springframework.data</groupId>
         <artifactId>spring-data-jpa</artifactId>
         <version>2.5.6</version>
      </dependency>
      <dependency>
         <groupId>org.hibernate</groupId>
         <artifactId>hibernate-core</artifactId>
         <version>5.4.32.Final</version>
      </dependency>
       <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-websocket</artifactId>
       </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-websocket</artifactId>
```

```
      </dependency>
   </dependencies>
   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.felix</groupId>
            <artifactId>maven-bundle-plugin</artifactId>
            <version>${bundle.plugin.version}</version>
            <extensions>true</extensions>
            <configuration>
               <instructions>
                  <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
                  <Bundle-Name>${project.artifactId}</Bundle-Name>
                  <Bundle-Version>${project.version}</Bundle-Version>
                  <Bundle-Activator>${project.artifactId}.Activator</Bundle-Activator>
                  <Export-Package>${project.artifactId}.api</Export-Package>
                  <Private-Package>${project.artifactId}.*</Private-Package>
                  <Import-Package>
                     !org.springframework.*,
                     *;resolution:=optional
                  </Import-Package>
                  <Embed-Dependency>*;scope=compile</Embed-Dependency>
                  <Embed-Transitive>true</Embed-Transitive>
                  <_fixupmessages>^Classes found in the wrong directory:
\{META-INF/versions/9/[^=]+=.+}</_fixupmessages>
               </instructions>
               <buildDirectory>target/deploy</buildDirectory>
            </configuration>
         </plugin>
      </plugins>
   </build>
</project>
```

- **target/deploy** - папката в, която след билдване и компилиране на проекта, ще бъдат създадени JAR файловет (bundle-и), които след това трябва да бъдат инсталирани и активирани в OSGi платформата.
- **pom.xml** - Maven файлът отговорен за локалното билдване на целия проект.

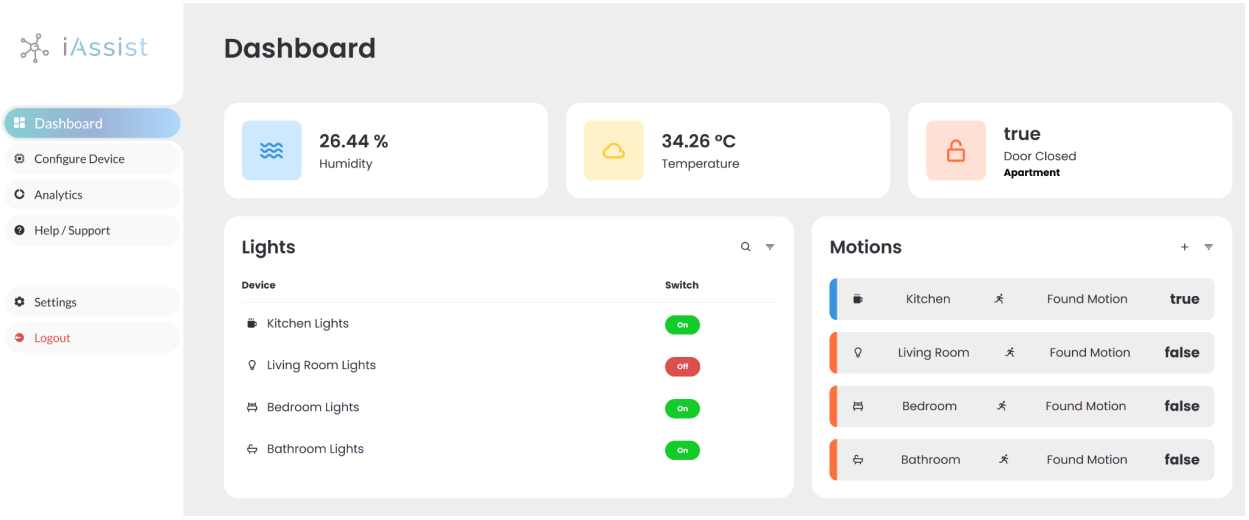
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>org.iassist</groupId>
   <artifactId>org.iassist</artifactId>
   <version>1.0.0</version>
   <packaging>pom</packaging>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.7.1</version>
      <relativePath />
   </parent>
   <properties>
      <java.version>17</java.version>
      <osgi.version>6.0.0</osgi.version>
      <bundle.plugin.version>5.1.2</bundle.plugin.version>
      <docker.plugin.version>0.28.0</docker.plugin.version>
   </properties>
   <modules>
      <module>spring-boot</module>
```

```
        <module>core-plugin</module>
    </modules>
</project>
```

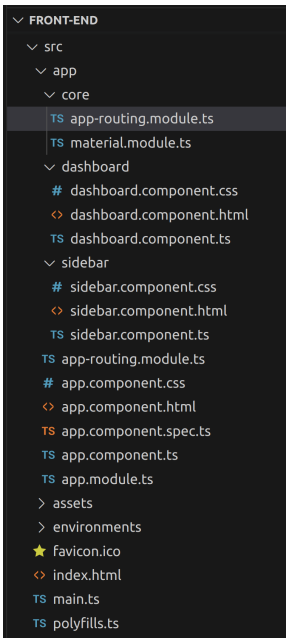
След създаването на основната системна логика на проекта, е ред на изграждането на потребителския интерфейс с Angular. Изборът на всички тези тегнологии позволява лесното и бързо създаване на всякакви модели и дизайни на front-end частта, ето защо по-долу разработения интерфейс е само пример, целящ да покаже един краен резултат на системата. В него има секция “Humidity”, която показва влажността на въздуха, както и секция “Temperature”, която показва температурата в стаята, като тази информация идва директно от сензор за температура и влажност на въздуха TuYa WSD500A. След това се визуализира секция “Door Closed - Apartament”, която индикира, когато вратата на апартамента се отваря или затваря, и в какво състояние е в момента. За тази функционалност данните се взимат от сензор за контакт SONOFF SNZB-04. Отдолу в потребителския интерфейс има секция “Motion”, която показва засечено движение в стаята, като данните се четат от сензора за движение SONOFF SNZB-03.

Разработения софтуер позволява лесното интегриране на всякакви други бъдещи функционалности, като например добавяне на секция “Lights”, която да показава, включените и изключени лампи във всяка стая, като за целта може да се използва сензорният ключ TuYa TS0041 или всеки друг ключ, които поддържа Zigbee комуникация и протокол.

Изборът на OSGi от своя страна позволява разработването например на отделен bundle, които да отговартя за секцията “Analytics” в потребителския интерфейс, която да дава резултат от анализа на данните получени от сензорите. Най-големият плюс би бил, че няма да е нужно да бъде рестартирано цялото приложение, а просто ще бъде стартиран новият Data Analysis Bundle в runtime-а на OSGi.



Потребителсцкия интерфейс на проекта би има следната структура:



- **core** - основните дефолтни файлове за проекта.
- **dashboard** - в тази папка са всички файловре свързани с “Dashboard” секцията - цветовете, шрифтовете, отстъпите и оформление, логика, която чете данните за сензорите от web socket-а на back-end-а.

➢ dashboard.component.html

```
<main>
        <div class="head-title">
                <div class="left">
                        <h1>Dashboard</h1>
                </div>
        </div>
        <ul class="box-info">
                <li>
                        <i class='bx bx-water'></i>
                        <span class="text">
                                <h3>{{ humidityData }} %</h3>
                                <p>Humidity</p>
                        </span>
                </li>
                <li>
                        <i class='bx bx-cloud'></i>
                        <span class="text">
                                <h3>{{ temperatureData }} °C</h3>
                                <p>Temperature</p>
                        </span>
                </li>
                <li>
                        <i class='bx bx-lock-open-alt'></i>
                        <span class="text">
                                <h3>{{ contactData }}</h3>
                                <p>Door Closed</p>
                                <h5>Apartment</h5>
                        </span>
                </li>
        </ul>
...
</main>
```

➢ dashboard.component.ts

```
import { OnInit, Component } from '@angular/core';
import { Stomp } from '@stomp/stompjs';
import SockJS from 'sockjs-client';

@Component({
  selector: 'app-dashboard',
  templateUrl: './dashboard.component.html',
  styleUrls: ['./dashboard.component.css'],
})
export class DashboardComponent implements OnInit {

  payloadData: string = "";
  temperatureData: string = "";
  humidityData: string = "";
  motionData: string = "";
  contactData: string = "";

  constructor() { }

  ngOnInit(): void {
    const socket = new SockJS('http://localhost:8081/ws');
    const stompClient = Stomp.over(socket);

    stompClient.connect({}, () => {
      stompClient.subscribe('/topic/contact', (message) => {
        this.contactData = JSON.parse(message.body).contact
```

```
      });

      stompClient.subscribe('/topic/temperature', (message) => {
        this.temperatureData = JSON.parse(message.body).temperature;
        this.humidityData = JSON.parse(message.body).humidity;
      });

      stompClient.subscribe('/topic/motion', (message) => {
        this.motionData = JSON.parse(message.body).occupancy
      });
    });

    stompClient.activate();
  }

  refresh(): void {
    window.location.reload();
  }
}
```

3.3. Стартиране на OSGi и Angular

Angular платформата може да бъде стартирана на `**localhost:42000**` със следните команди:

```
~$ cd /<main-project-path>/front-end
~$ ng serve
```

OSGi платформата може да бъде свалена от https://download.eclipse.org/equinox/drops/R-4.28-202306050440/index.php, като за ще бъде използвана версия 4.28 на Equinox OSGi SDK. Следва свалянето му и разархивирането на ZIP файла в папка с име `**../EclipseRT-OSGi**`.

За да бъдат използвани всички ползи на OSGi системата, трябва ново създадените JAR (Java Archive) файлове, да бъдат конфигурирани в OSGi конфигурационния файл, за да се пускат от **target/deploy** папката на проекта автоматично в OSGi Runtime-а. За целта трябва да се отвори конфигурацията на OSGi:

```
~$ cd /../EclipseRT-OSGi/rt/configuration/org.eclipse.equinox.simpleconfigurator
~$ vi bundles.info
```

И в нея да се добавят следните редове:

```
#encoding=UTF-8
#version=1
org.iassist.core.plugin,1.0.0,/<main-project-path>/back-end/target/deploy/org.iassist.core.pl
ugin-1.0.0.jar,4,true
```

```
org.iassist.spring.boot,1.0.0,/<main-project-path>/back-end/target/deploy/org.iassist.spring.boot-1.0.0.jar,4,true
```

След това трябва да се стартира самият OSGi Runtime:

```
~$ cd /../EclipseRT-OSGi/rt/
~$ ./rt
```

3.4. Автоматизиране на процеса по стартиране

Стъпките по стартирането на цялата система от docker контейнери, може лесно да бъде автоматизиране след първоначалното конфигуриране. Това е възможно с използването на софтуерният компонент Docker Compose, чрез който с помощта на YAML файл, може лесно да се стартират, менажират и управляват множество docker контейнери. С по-долу посочения файл и командата за неговото стартиране, целия процес се автоматизира в рамките на една команда.

- Съдържание на файла '**`docker-compose.yaml`**' за автоматизиран процес по стартирането на docker контейнерите:

```
version: '3'
services:
  # Start the Mosquitto container
  mosquitto:
    image: eclipse-mosquitto
    container_name: mosquitto
    volumes:
      - /opt/mosquitto:/mosquitto
      - /opt/mosquitto/data:/mosquitto/data
      - /opt/mosquitto/log:/mosquitto/log
    ports:
      - 1883:1883
      - 9001:9001  # This port is for Webhooks
    restart: unless-stopped
  # Start the Zigbee2MQTT container
  zigbee2mqtt:
    image: koenkk/zigbee2mqtt
    container_name: zigbee2mqtt
    volumes:
      - /opt/zigbee2mqtt:/app/data
      - /run/udev:/run/udev:ro
    depends_on:
      - mosquitto
    devices:
      - /dev/ttyACM0:/dev/ttyACM0
    ports:
      - 8082:8080  # This port is for Frontend
    environment:
      - TZ=Europe/Sofia
```

```
    restart: unless-stopped
```

- Проверка на средата за предварително вървящи контейнери, и тяхносот спиране при наличие на такива:

```
~$ docker rm -f zigbee2mqtt
~$ docker rm -f mosquitto
```

- Команда за стартиране на '**docker-compose.yaml**':

```
~$ docker compose up
```

- Команда за спиране на '**docker-compose.yaml**':

```
~$ docker compose down
```

Глава 4

Изследване на характеристиките на реализираната система

Изследването на характеристиките на реализираната система предоставя дълбок поглед върху нейната работа и функционалности, като осигурява изчерпателна информация за това как тя се представя в реални сценарии. От изключително значение е да се осигури, че системата може да се адаптира към разнообразни условия и да предоставя съгласувана и надеждна работа.

Разглеждайки внимателно резултатите от изследването, можем да направим няколко ключови извода за системата. Първо, системата демонстрира висока степен на надеждност и устойчивост при събирането, обработката и предаването на данни. Различните компоненти на системата успешно се съгласуват и работят в хармония, което подчертава добрата архитектурна основа, върху която е изградена.

Следващо, изследването показва, че системата е способна да се справя със значителни натоварвания и обеми на данни. Това я прави подходяща за разширени IoT приложения, където е необходимо да се обработват големи потоци от информация от различни сензори. Системата поддържа ефективност и отзивчивост, като предоставя важни данни за вземане на решения в реално време.

Освен това, изследването разкрива възможностите за разширяемост на системата. Тя е готова да приеме нови сензори и устройства без значителни промени в архитектурата, което предоставя възможности за бъдещо развитие и подобрения. Този аспект е от съществено значение, тъй като IoT технологиите продължават да се развиват и разширяват.

Изводите от изследването подчертават, че реализираната система отговаря на високите изисквания за ефективност, надеждност и разширяемост. Тя предоставя стабилно и качествено събиране, обработка и анализ на данни от различни сензори, като може да се интегрира успешно в разнообразни IoT приложения. Откритостта й за бъдещи разширения и подобрения я прави подходяща за дългосрочни иновации в областта на Internet of Things.

Заключение

В настоящата дипломна работа "Софтуер за 'Сензора като услуга' в екосистемата на IoT базирана на технологията OSGi" разгледахме и реализирахме едно иновативно и ефективно решение за събиране, управление и анализ на данни от различни сензори и устройства в интернет на нещата.

Проектът имаше за цел да отговори на нарастващата нужда от съвременни технологии, които могат да улеснят и оптимизират взаимодействието и управлението на множество IoT устройства в единна система. В процеса на разработка и реализация, активно използвахме технологията OSGi, която ни даде възможност да създадем модулна и мащабируема архитектура, като по този начин подобрихме производителността и гъвкавостта на системата.

В рамките на обзора на проблемите и съществуващите решения, изследвахме различни технологии за събиране и управление на данни от сензори в IoT системите. Проведохме подробен анализ на типовете сензори и устройства, методите за комуникация, архитектурните подходи и методите за анализ на данните. Това ни предостави ценни насоки при избора на подходящите технологии и методи за нашия проект.

Проектирането на софтуерната архитектура беше от съществено значение за успешното изпълнение на проекта. Предвидихме и разпланирахме всички основни модули и компоненти, които трябваше да бъдат включени в системата, и ги интегрирахме чрез OSGi стандартите. Това осигури ефективно управление и разширяемост на системата, като лесно можем да добавяме нови функционалности и да променяме съществуващите без да нарушаваме функционалността на цялата система.

Реализацията на системата включваше използването на различни технологии и инструменти. Програмирането с Java и използването на Spring и Angular фреймуърките ни дадоха мощни инструменти за създаване на бизнес логика и потребителски интерфейс. Използвахме Docker за контейнеризация, която допринесе за лесното деплойване и мащабиране на системата. Също така, използвахме различни протоколи за комуникация, като MQTT и Zigbee2MQTT, за успешното интегриране на разнообразни сензори и устройства.

Системата ни предостави възможност да събираме и управляваме данни от различни сензори в реално време. Анализът на данните ни даде ценна информация за тенденциите и събитията, които могат да бъдат използвани за оптимизация и подобрение на различни процеси и системи. Това съчетание от функционалности предостави полезни и релевантни данни за крайните потребители и бизнеса, които допълнително подобриха ефективността и стойността на системата.

В заключение, дипломната работа по проекта "Софтуер за 'Сензора като услуга' в екосистемата на IoT базирана на технологията OSGi" успешно постигна своите цели и задачи, представяйки иновативно и мащабируемо решение за събиране, управление и анализ на данни от сензори и устройства в IoT среда. Разработеният софтуерен модел демонстрира значителен потенциал за бъдещи приложения в различни области и открива нови възможности за развитие и напредък в областта на интернет на нещата.

Декларация за авторство

Използвани източници

1. OSGi Alliance. (2023). OSGi Core Specification. Извлечено от https://osgi.org/
2. Eclipse OSGi Project. (2023). Извлечено от https://www.eclipse.org/equinox/
3. Zigbee Alliance. (2023). Zigbee Technology Overview. Извлечено от https://www.zigbeealliance.org/
4. MQTT (2023). Извлечено от MQTT https://mqtt.org/
5. Zigbee2MQTT. (2023). Zigbee2MQTT Documentation. Извлечено от https://www.zigbee2mqtt.io/
6. Mosquitto. (2023). Eclipse Mosquitto Documentation. Извлечено от https://mosquitto.org/documentation/
7. Spring Framework Reference Documentation. (2023). Извлечено от https://docs.spring.io/spring-framework/docs/current/reference/html/index.html
8. Spring Framework Reference Documentation. (2023). Извлечено от https://docs.spring.io/spring-framework/docs/current/reference/html/index.html
9. Angular Documentation. (2023). Извлечено от https://angular.io/docs
10. Docker Documentation. (2023). Извлечено от https://docs.docker.com/
11. Ubuntu Documentation. (2023). Извлечено от Ubuntu https://docs.ubuntu.com/
    Phillips, C. (2020). Practical MQTT with Paho. Packt Publishing.
12. Banks, E., & Scullion, G. (2017). MQTT Essentials - A Lightweight IoT Protocol.
13. Horton, A. (2019). Mosquitto MQTT Broker. Packt Publishing. [Книга]
14. Walla, G., & Mayer, J. (2016). Spring Boot in Action. Manning Publications.
15. Wilson, J., & Kovalenko, O. (2018). Angular in Action. Manning Publications.
16. Kumar, R., & Jain, S. (2021). IoT Projects with Raspberry Pi Zero. Apress.
17. Dargan, A., & Singh, M. (2019). Internet of Things: Novel Advances and Envisioned Applications. Springer. [Книга]
18. Sheng, M., Yu, J., & Dey, A. K. (2017). The Internet of Things. Springer.
19. Ray, S., & Ray, A. (2020). Internet of Things. CRC Press.
20. Schmidt, D. C., & Cunha, D. A. (2016). Engineering Internet of Things (IoT) Systems: Architectures, Algorithms, Methodologies.