**ДИПЛОМНА РАБОТА**

на тема

„Платформа за събиране на данни в реално време”

Дипломант: **Иван Цветомиров Иванов**

Специалност: **Разпределени системи и мобилни технологии**

Факултетен номер: **5MI3400182**

Научен ръководител:

**Проф. д-р Милен Петров**

**Съдържание**

**Глава 1. Увод**

1.1. Актуалност на проблема и мотивация

1.2. Цел и задачи на дипломната работа

1.3. Структура на дипломната работа

**Глава 2. Технически обзор**

2.1. Съвременни софтуерни технологии за уеб разработка

2.2. Съвременни мрежови и интернет протоколи за комуникация

2.3. Популярни IoT платформи с общо предзначение и сензори за измервания на околната среда

2.4. Пример на проекта „Sensor Community“

**Глава 3. Използвани технологии**

3.1. Изисквания към средствата (технологии, платформи и методологии)

3.2. Видове средства (технологии, платформи и методологии) и начин и място за използването им – сравненителен анализ

3.3. Избор на средствата (технологии, платформи и методологии)

3.4. Изводи

**Глава 4. Анализ**

4.1. Концептуален модел

4.2. Потребителски (функционални) изисквания (права, роли, статуси, диаграми, ...)

4.3. Качествени (нефункционални) изисквания (като напр. преносимост, използваемост, скалируемост, поддръжка, ...)

4.4. Работни (бизнес) процеси

4.5. Изводи

**Глава 5. Проектиране**

5.1. Обща архитектура – напр. слоеве, модули, блокове, компоненти...

5.2. Модел на данните (напр. база данни, файлова структура, ...)

5.3. Диаграми (на структура и поведение - по слоеве и модули, с извадки от кода)

5.4. Потребителски интерфейс (опционално)

5.5. Ресурсни и спомагателни модули (опционално)

**Глава 6. Софтуерна реализация**

6.1. Реализация на модулите

6.2. Системна интеграция (опционално)

6.3. Планиране на тестването - тестови сценарии, процедури, ...

6.4. Модулно и системно тестване

6.5. Анализ на резултатите от тестването и начин на отразяването им

6.6. Експериментално внедряване (технологични изисквания, инсталиране, условия, използване, ...)

**Глава 7. Заключение**

7.1. Обобщение на изпълнението на началните цели

7.2. Насоки за бъдещо развитие и усъвършенстване

# Глава 1. Увод

# Актуалност на проблема и мотивация

Последните няколко години доста силно навлизат интелигентни сензори за измерване на различни параметри на околната среда, съответно и доста нашумяват различни уеб платформи за събиране, обработка и съхранение на различни измервания в реално време.

Подобни проекти са доста предпочитани от любители на метеорологията защото чрез публикуваните данни, може да бъде реализирана мрежа за „отворени данни“, които да се използват за различни цели.

# Цел и задачи на дипломната работа

Основна цел на текущата дипломна работа е проектирането и разработването на платформа за събиране, обработване и съхранение на данни от различни измервателни IoT устройства в реално време.

При проектирането и разработването на проекта, трябва да бъде предвидено крайното предназначение – работа с любители на метрологията и да бъде предоставен удобен и лесен потребителски интерфейс, но също така и лесен, но сигурен начин за публикуване на данни в платформата.

Освен определяне на основни изисквания към платформата, трябва да бъдат проучени и подходящи бази от данни за съхранение на обработените данни.

При разработването на проекта ще бъде основен фокус върху използването на езици, работни рамки и проекти с отворен код.

Също така самият проект ще се разработва като проект с „отворен код“.

# Структура на дипломната работа

Дипломанта работа е структурирана от следните глави. Всяка главa описва конкретна стъпка от анализиране на основните изисквания към проекта, проектирането на основните модули и тяхната разработка с конкретно използвани технологии.

**Глава 1. Увод** – въведение в тематиката и целите на дипломната работа.

**Глава 2. Технически обзор –** разглеждане на съществуващи софтуерни решения и основни дефиниции на разработваната тема.

**Глава 3. Използвани технологии –** кратък анализ на Java и Spring и възможностите за софтуерна разработка на проекта.

**Глава 4. Анализ –** описание на основните функционални и нефункционални изисквания при проектирането на проекта.

**Глава 5. Проектиране –** описание на архитектурата и изисвканията към нейните модули.

**Глава 6. Софтуерна реализация** –описание на софтуената реалиазация посредворм избраните софтуерни тенологии и архитектура.

**Глава 7. Заключение** – обобщение на крайния резултат и бъдещо развитие на проекта.

# Глава 2. Технически обзор

Съвременните софтуерни технологии за уеб разработка на приложения и услуги основно е свързана с различни софтуерни работни рамки който предлагат конкурентен модел/многонишнков модел на работа за разработка на сървърната част на едно приложение, но също така и конкретни софтуерни архитектури за реализация, например microservices и методи за комуникация между отделните услуги.

# Съвременни софтуерни технологии за уеб разработка

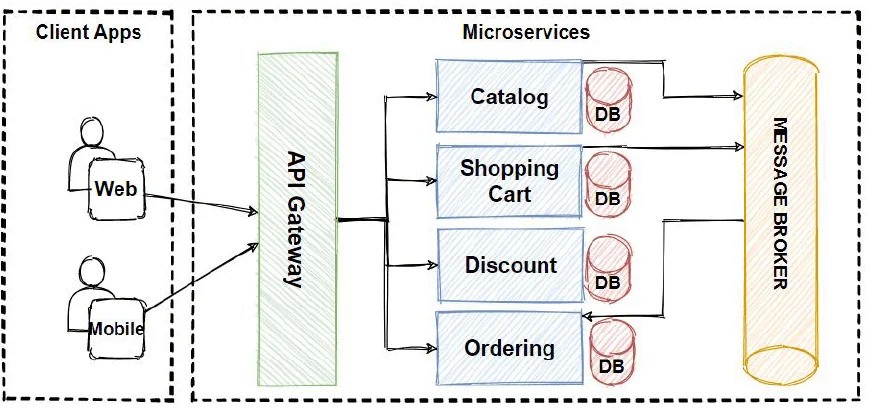
Тази тематика основно е свързана е свързана с различни софтуерни рамки който предлагат различни предимства и техники за разработване на сървърната и клиентската част на дадено софтуерно приложение.

Но важен фактор при софтуерната реализация е избора на архитектурен модел при сървърната част на приложението. Последните няколко години, много силно набира популярност употребата на microservices като архитектура на приложението, а не използването на познатата монолитна архитектура.

Употребата на microservices има основна цел за разделянето на бизнес логиката на приложението на отделни „services” за който се стреми независимост и взаимосвързаност между тях.

Употребата на microservices архитектурен модел е вече широко използван от различни компании например Google, Pivotal, Twitter, Meta, Netlfix и доста други.

Нагледен пример за microservices софтуерна архитектура е показана на долната фигура.



*Фигура 1.*

Основните предимства пред монолитната архитектура са основно свързани със гъвкавост, мащабируемост, бързо внедряване в облачна среда.

Но основно предимство на microservices, че разделянето на приложението отделни services предлага лесно декомпозиране, реализиране и тестване на проекта, за разлика от монолитната архитектура при която имаме комплексно реализиране на бизнес логиката.

Както беше споменато за отделните услуги са независими и където е необходимо се реализира комуникация между тях. Тук са налични няколко начина за комуникация:

**HTTP базирана** – класически и прост метод на комуникация посредством уеб HTTP клиент.

**AMQP базирана** – тук е широко разпространена употребата на RabbitMQ, ActiveMQ и други базирани проекти.

**Аpache Kafka** – това е софтуерна платформа за data streaming и обмен на съобщения.

Освен работната рамка и софтуерната архитектура за реализация на сървърната част е важна и реализацията на клиентската част посредсвом софтуерна рамка за клиентката част и тук широко се употребяват следните проекти: Angular, React, Vue, Svelte.

# Съвременни мрежови и интернет протоколи за комуникация

Освен комуникацията между отделните услуги в разработеното приложение, също така е важен и интересен подхода за реализация на комуникация между IoT устройството и интернет платформата.

Общо взето комуникацията между IoT и платформата може да бъде осъществена по следните начини:

**HTTP:** Най-разпространения протокол за интернет комуникация използвайки REST API заявки. Това е най-лесния и прост в същото време и предлага добра сигурност посредством различни методи за авторизация. Но е важно да се подчертае, че този начин на комуникация не е много оптимизиран за IoT устройства с ограничени ресурси.

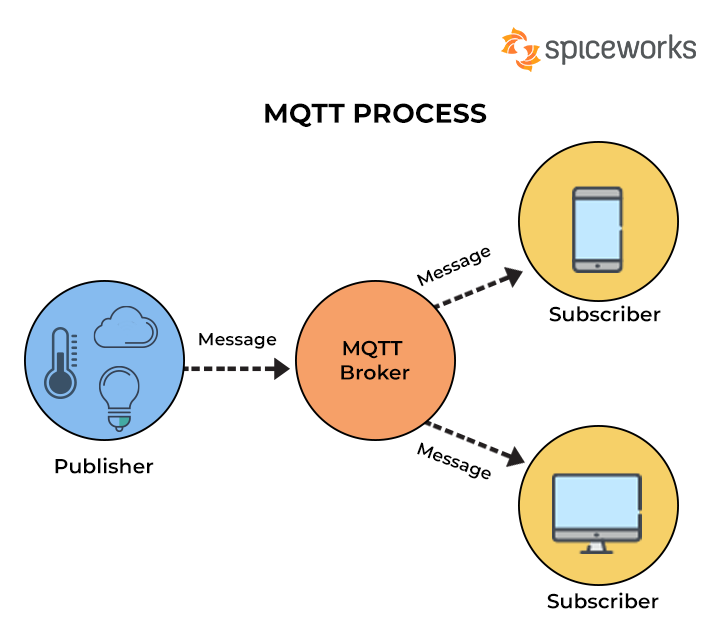
**MQTT:** Популярен протокол за IoT устройства с ограничени ресурси, като комуникацията между IoT и платформата е доста лека защото е базирана на модела „publish–subscribe” и също така широко разпространен и при реализация на комуникация между отделни IoT устройства.

**CoAP:** Основно този протокол е алтернатива на MQTT, но е UDP базиран и тук не се гарантира успешното доставяне на данните, но това го прави подходящ за приложения който имат ниска латентност и IoT устройствата имат силни ограничения от страна на захранване например.

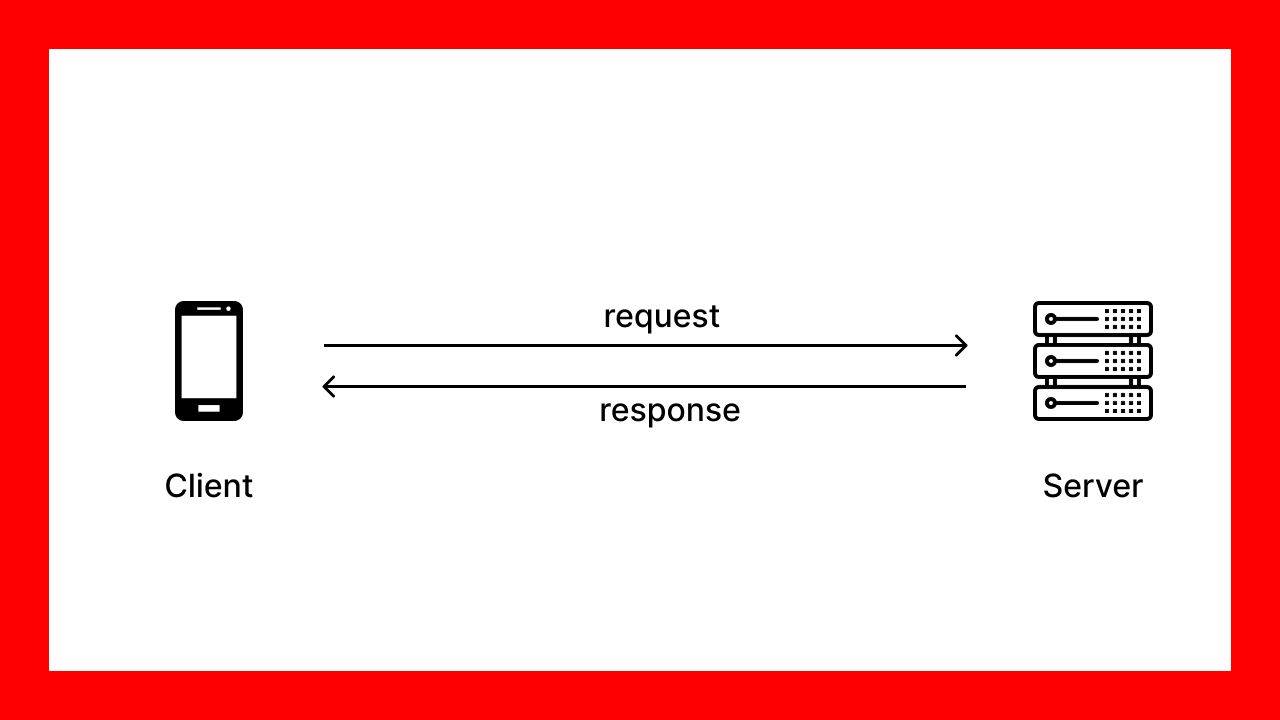
**WebSockets:** Използвайки този начин за комуникация, има възможност за двупосочна комуникация в реално време и този протокол е подходящ когато се изисква постоянна връзка между IoT устройството и уеб платформата.

Разбира се има и други протоколи за комуникация например AMQP и XMPP, които се базират на разгледаните начини за комуникация, но биват използвани за различни цели. Например XMPP e подходящ когато имаме комуникация не само между IoT и уеб среда, но и комуникация между отделни устройства и обмен на мета данни.

Друго важно пояснение за протоколи като MQTT, CoAP, AMQP е необходим „брокер“ за осъществяване на комуникация, за разлика от HTTP(S) и (Secure) WebSocket където имаме познатия модел „клиент-сървър“. Нагледен пример е показан на долните две фигури:



Фигура 2.



Фигура 3.

# Популярни IoT платформи с общо предзначение и сензори за измервания на околната среда

С навлизането на различни потребителски IoT хардуерни и софтуерни платформи предлагащи голяма гама от сензори за различни измервания на широк спектър от параметри на околната среда например:

С навлизането на различни потребителски IoT хардуерни и софтуерни платформи предлагащи голяма гама от сензори за различни измервания на широк спектър от параметри на околната среда например:

* Температура
* Влажност на въздуха
* Качество на въздуха спрямо фини прахови частици
* Качество на въздуха спрямо различни газове
* UV индекс

Освен изборните параметри съществуват и различни сензори който предлагат комплексни измервания на базата на няколко параметъра например:

* Коефициент за качество на въздуха реализиран чрез количество фини прахови частици и наличие на определени газове.
* Сила на слънцето спрямо UV индекс и сила на греене.
* Надморска височина на база атмосферно налягане

**## Controllers**

\* Arduino

\* Raspberry Pi

\* Raspberry Pi Micro

\* NodeMCU

\* STMicro

\* PIC32

\* MSP430

\* ESP32

\* ESP8266

**## Sensors**

\* BME280

\* BME680

\* SDS011

# Пример на проекта „Sensor Community“

Проекта Sensor Community е реализиран по следния начин:

…..

# Глава 3. Използвани технологии

Основен фокус при избора на езици за разработка, софтуерни рамки, платформи и технологии е да бъдат с „отворен код“.

**Въведение и основни изисквания към използваните езици, работни рамки, бд, SQL срещу NoSQL**

# Java

Обектно ориентиран език за софтуерна разработка разработен и подържан от Oracle и има слените предимства:

**Платформено независим** – веднъж компилирано приложението, то може да бъде изпълнявано от всички операционни системи без да е необходима компилация за всяка от тях.

**Сигурност** – базирайки се на платформената независимост, то реализираните приложения не се изпълняват директно от операционната система, а от виртуална машина наречена “Java Virtual Machine” или JVM.

Тази виртуална машина се грижи за заделяне на ресурси като памет, оптимизиране на изчислителното време на процесора, освобождаване на паметта.

**Голям набор от софтуерни пакети** – платформата предоставя широк набор от класове за работа с различни ресурси от операционната система, файлова система и други.

Както всички обектно ориентирани езици софтуерната реализация се организира в класове, обекти и пакети.

**Java се организира в класове и пакети.**

**Структура на Java приложения.**

Процесът на компилация и изпълнение на Java приложение е показан на долната фигура. [link]

# 

*Фигура 1.*

Както беше споменато приложенията не се изпълняват директно от операционната система, а от виртуална машина, като за тази цел при компилация не се компилира до машинен език, а до “java byte code”, за което изпълнение е необходима инсталирана Java RE.

Java е широко използван за реализация на различен тип приложения като:

1. Уеб приложения и уеб услуги
2. Мобилни и десктоп приложения
3. Разпределени и облачни приложения

Java основно се разработва и подържа от Oracle и софтуерната общност на Java екосистемата (SAP, VMware, Eclipse, Apache, …).

Като платформа за софтуерна разработка има следните версии/editions:

**Java Standard Edition (SE**) – съдържа основните пакети и основните API на езика за разработка на приложения.

**Java Enterprise Edition (EE)** – пакети и спецификация за разработка на бизнес приложения. В последствие преименуван на Jakarta EE и се разработва и подържа от Eclipse Software Foundation.



*Фигура 2*

Съществуват и други допълнения или видове на Java като JavaFX, JavaME но те не са от особен интерес при разработката на проекта.

<https://www.scaler.com/topics/types-of-java/>

# Основни Java пакети

**Пакет java.lang** или това е основния Java пакет с фундаментални класове на езика.

**Пакет java.util** съдържа широк набор от класове за работа с колекции и различни структури от данни, интернационализация и други.

**Пакет java.math** предоставя основни математически функции като sin, cos и други класове за работа с числа и математически операции. В този пакет са въдени и типовете BigInteger, BigFloat за точни изчисления на големи числа.

**Пакет java.io** е основния пакет на Java платформата за работа с файлова система, работа с входни/изходни данни, serialization/ deserialization на данни.

**Пакет java.net** е набор от класове за работа с мрежови сокети и мрежови протоколи като UDP и TCP.

**Пакет java.text** предлага основни класове за обработка и форматиране на числови типове, символни низове, часове и дати.

След Java 8 е въведен и **пакета java.time** който специализира работата с часови зони, дата и час, календар. Както и класове за форматиране.

Пакет java.sql е основни пакет в платформата за достъпване и обработване на данни от SQL бази данни.

# Jakarta ЕЕ

Java EE или Jakarta ЕЕ е набор от спецификации и стандарти за разработката на бизнес приложения и е широко използван за реализация на различни облачни приложение, уеб услуги и приложения.

При Jakarta EE основен фокус е сигурността и различни оптимизации при внедряването в облак.

В момента Jakarta се разработва и подържа от Eclipse Software Foundation и софтуерната общност на IBM, Apache Software Foundation, VMware, SAP и други.

На долната фигура са показани основните Jakarta EE профили.[2]



*Фигура 6. Jakarta EE профили*

# Основни спецификации за уеб приложения

**Jakarta Servlet –** основата на другите спецификации от екоситемата на Jakarta за обработка на HTTP заявки. Тук буквално се дефинира „Как се управляват HTTP заявките?“ чрез синхронна или асинхронна обработка.

**Jakarta WebSocket –** спецификация за разработването на WebSocket приложения изискващи комуникация в „реално време“.

**Jakarta Faces –** спецификация позната още като: Jakarta Server Faces или Java Server Faces и един от първите стандарти за създаването на уеб приложения използвайки MVC шаблон.

**Jakarta MVC –** това е стандарт базиран на идеологията на Jakarta Faces но реализиран посредством …. [todo]

# Основни спецификации за уеб услуги

**Jakarta RESTful Web Service** – основната спецификация за разработка на уеб услуги използвайки REST шаблон за комуникация… [todo] – also Microprofile.

**Jakarta XML Web Service** – основната спецификация за разработка на уеб услуги използвайки SOAP протокол.

# Enterprise спецификации

# Spring

**Въведение в Spring, история на Spring и Как Spring надгражда Jakarta EE.**

Spring e широко използвана софтуерна рамка с отворен код за разработка на уеб приложения, уеб услуги, автоматизации на бизнес процеси, работа с бази от данни базирайки се на Java EE стандарта.

В голяма степен надгражда Java EE/Jakarta и предлага широк спектър от оптимизации и автоматизации при реализацията на основните бизнес процеси, работа с бази от данни, допълнителни библиотеки и услуги от трети страни.

# Spring Framework

Реализация на високо ниво за разработка на уеб приложения чрез Java EE/Jakarta и в голяма степен е алтернатива на Enterprise Java Beans.

На фигура 2 е показана основната структура на приложната рамка за разработка.



Фигура 2. Архитектура на Spring Framework

Spring Framework предлага следните неща:

* Уеб приложения използвайки MVC шаблона
* Поддръжка на JSP/JSTL
* REST уеб услуги
* Работа с релационни БД
* Работа с нерелационни БД
* Средства за автирузация и аут…
* Предаване на съобщения с различни протоколи

Spring Framework подържа слените Servlet приложни сървъри:

* Apache Tomcat
* Eclipse Jetty

# Spring Boot

Spring Boot предлага готова конфигурация на Spring Framework базирана на шаблона „Convention over configuration“, включително предлага и вграден приложен уеб сървър – Apache Tomcat, Eclipse Jetty и Netty.

Архитектура на Spring Boot приложение e показана на фигура 3.



Фигура 3.

Тук се подържат два стандарта за разработка на уеб приложения и услуги:

**Spring web/Servlet**

**Spring Webflux/Reactive**

# Spring Cloud

…..

**Spring Security**

Тук се предлагат голям набор от класове и конфигурации за authentication и authorization на разработеното приложение. Принципът на работа на Spring Security e показан на фигура 4.



Фигура 4.

**Spring Data**

…

# Mongo Db

Тук се предлагат голям набор от класове и конфигурации за authentication и authorization на разработеното приложение. Принципът на работа на Spring Security e показан на фигура 4.

# Docker

Тук се предлагат голям набор от класове и конфигурации за authentication и authorization на разработеното приложение. Принципът на работа на Spring Security e показан на фигура 4.

# React.js и Next.js

Тук се предлагат голям набор от класове и конфигурации за authentication и authorization на разработеното приложение. Принципът на работа на Spring Security e показан на фигура 4.

# Други

Тук ще се изброят всички помощни библиотеки и средства.

Може да се включат IDE и описание на някой процеси.

* TypeScript
* Material UI
* Formik
* Yup

Тук се предлагат голям набор от класове и конфигурации за authentication и authorization на разработеното приложение. Принципът на работа на Spring Security e показан на фигура 4.

# Глава 4. Анализ

4.2. Потребителски (функционални) изисквания (права, роли, статуси, диаграми, ...)

4.3. Качествени (нефункционални) изисквания (като напр. преносимост, използваемост, скалируемост, поддръжка, ...)

4.4. Работни (бизнес) процеси

4.5. Изводи

**Концептуален модел**

# Функционални изисквания

**Изисквания на потребителите към платформата.**

**Use-case диаграма и анализ**

**5 страници, детайлно разглеждане на всеки случай на употреба**

# Нефункционални изисквания

1. **Сигурност**
2. **Производителност**
3. **Комуникация между IoT и платформата**
4. **Управление и анализ на устройства и данни**

**3 страници**

# Основни бизнес процеси

Use case диаграми.

# Изводи

# Глава 5. Проектиране

Тук се разглежда и описва процеса на проектиране на отделните модули от платформата основавайки се на основните изисквания и ограничения към тях.

# Обща архитектура

Общата софтуерна архитектура на приложение е показана на долната фигура.



*Фигура 4: Обща архитектура на приложението*

Избира се подхода за софтуерна реализация посредством уеб услуги защото **….ТОДО… от класическата MVC архитектура** и има следното разделение на слоевете:

**Клиентската логика (front-end application) –** това е клиентката част на приложението, тук е включен и потребителския интерфейс UI, също така той изцяло двупосочно комуникира с сървърната част.

**Сървърната логика (back-end application) –** сървърната част на приложението и тук е реализация на бизнес логиката, в конкретния случай е важен и подбора на софтуерна архитектура, модел за управление на данните в БД и т.н.

**Слой за съхранение –** отговаря за съхранение на обработените данните от сървърната част.

# Реализация на клиентката част

Тук ще бъде описана реализацията на сървърната част на приложението.

# Реализация на сървърната част

Тук ще бъде описана реализацията на сървърната част на приложението.

**# N-Tier architecture**

1. Presentation layer - UI/front-end

2. Controller layer - Orchestrates actions between UI and bussines logic

3. Service layer - implements core app logic, provides services

4. Data Access layer DAO - manages databases and operatiosn

**## Advantages**

1. Modularity

2. Scalanility

3. Security

4. Mainrainability

**## Layers**

---------------

|  API Layer  |

---------------

    I   I

---------------

|   Business  |

|   Layer     |

---------------

   I   I

**---------------**

|   DAO       |

|   Layer     |

---------------

      I

---------------

|   Database  |

---------------

# Модел на данните

Основен модел на данните.

# Глава 6. Софтуерна реализация

Реализация на модулите

Системна интеграция

Модулно и системно тестване

Анализ на резултатите от тестването

Внедряване на проекта (технологични изисквания, инсталиране, условия, използване, ...)

Spring Resource Server

Spring Authorization Server

Микросървисна архитектура и концепция за реализация

Zookeeper, Spring Cloud предимства, Spring Eureka

# Глава 7. Заключение

7.1. Обобщение на изпълнението на началните цели

7.2. Насоки за бъдещо развитие и усъвършенстване

# Използвана литература

1. Лекции от дисциплината „Обектно-ориентиран анализ и проектиране на софтуерни системи“ – проф. Боян Бончев.
2. Java в облака – Джон Лонг и Кени Бастани.
3. <https://www.javaguides.net/2020/01/solid-principles-in-java-with-examples.html>
4. <https://waytoeasylearn.com/learn/microservices-architecture/>
5. <https://www.mongodb.com/resources/products/fundamentals/why-use-mongodb>
6. <https://www.freecodecamp.org/news/what-is-http/>
7. <https://www.spiceworks.com/tech/iot/articles/what-is-mqtt/>

# Списък с фигури

* Фигура 1 –
* Фигура 2 –
* Фигура 3 –
* Фигура 4 –

# Приложение № 1: Терминологичен речник

# Приложение № 2: Потребителски наръчник