ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - ВАРНА

ФАКУЛТЕТ ПО ИЗЧИСЛИТЕЛНА ТЕХНИКА

И АВТОМАТИЗАЦИЯ

****

Катедра „Компютърни науки и технологии“

**ДИПЛОМНА РАБОТА**

**Тема:**

Проектиране и разработка на мобилно приложение за поръчка на храна

**Изготвил:**

**Специалност:**

**Факултетен номер:**

Стефан Генчев Генев

Компютърни системи и технологии

19623318

ТУ Варна, 2024 г. Ръководител:

23.06.2024г. /гл. ас. д-р Д. Николаева/

**Съдържание**

Contents

[1. Въведение 4](#_Toc170381831)

[2. Анализ на проблема и избор на технологии 5](#_Toc170381832)

[2.1 Анализ на проблема 5](#_Toc170381833)

[2.2 Избор на технологии 10](#_Toc170381834)

[2.2.1 Microsoft SQL Server 11](#_Toc170381835)

[2.2.2 Kotlin 12](#_Toc170381836)

[2.2.3 Spring Framework 14](#_Toc170381837)

[2.2.4 Retrofit 16](#_Toc170381838)

[2.2.5 Cloudinary 17](#_Toc170381839)

[2.2.6 Material design 19](#_Toc170381840)

[3. Проектиране на системата 21](#_Toc170381841)

[3.1 Спецификация на приложението 21](#_Toc170381842)

[3.2 Задачи за изпълнение 24](#_Toc170381843)

[3.3 Архитектура 26](#_Toc170381844)

[3.3.1 MVC 26](#_Toc170381845)

[3.3.2 MVVM 26](#_Toc170381846)

[3.4 Функционален дизайн 26](#_Toc170381847)

[3.4.1 Use Case диаграма 28](#_Toc170381848)

[4. Реализиране на системата 33](#_Toc170381849)

[4.1 Система за контрол на версиите 33](#_Toc170381850)

[4.2 База данни 33](#_Toc170381851)

[4.2.1 Code first: 33](#_Toc170381852)

[4.2.2 Структура: 35](#_Toc170381853)

[4.3 Сървър 40](#_Toc170381854)

[4.3.1 Структура 40](#_Toc170381855)

# 1. Въведение

В забързаното ни ежедневие времето е един от най-ценните ресурси. Хората са заети с работа, учене, семейни задължения и социални ангажименти, което създава необходимост от бързи и ефективни решения за ежедневни задачи, включително храненето.

Мобилните приложения за поръчка на храна предлагат удобен и лесен начин за избор и поръчка на ястия от различни ресторанти и заведения директно от мобилното устройство, без да се налага физическо присъствие. В този контекст, тези приложения се утвърждават като незаменим инструмент за много хора.

Тези приложения решават редица проблеми, свързани с модерния начин на живот. Първо, те предоставят лесен достъп до разнообразие от храни и кухни, което е особено полезно за хора, живеещи в райони с ограничен избор на ресторанти. Второ, процесът на поръчка е бърз и интуитивен, като позволява на потребителите да избират, поръчват и плащат за храната си без усилие. Това спестява време и усилия, които могат да бъдат насочени към други важни дейности.

Една от ключовите характеристики на успешните мобилни приложения за поръчка на храна е лесният за използване интерфейс. Интуитивният дизайн и опростената навигация правят приложението достъпно за широк кръг от потребители, независимо от тяхната технологична грамотност. Чистият и подреден интерфейс позволява бързо и лесно разглеждане на менютата, избор на ястия и завършване на поръчката с минимални усилия.

Тези платформи също така използват данни от предишни поръчки и потребителски предпочитания, за да предоставят персонализирани препоръки, което подобрява общото потребителско изживяване.

С този проект дипломантът цели да предложи решение, което да отговаря на сегашните стандарти свързани с мобилните приложения, както и да бъде подготвено за за бъдещите иновации в дигиталните технологии и променящите се изисквания на потребителите.

# 2. Анализ на проблема и избор на технологии

## 2.1 Анализ на проблема

За да разберем по - детайлно проблема и ролята на приложението, трябва да обясним как протича целия процес по поръчка на храна.

A diagram of a delivery service

Description automatically generated

*Фиг. 1 – процес за поръчка на храна чрез платформа/приложение*

Онлайн платформите за поръчка на храна се базират основно на модела на "агрегаторския бизнес модел", при който собственикът на бизнеса генерира приходи чрез комисионна такса за услугата. С други думи, те предоставят централизиран хъб, където всички ресторанти и други места за хранене на целевия пазар са изброени, за да могат клиентите да поръчват.

Отделните стъпки на това как работят онлайн платформите за поръчка на храна биват:

1. **Разглеждане на опциите:** Клиентите използват платформата, за да разгледат разнообразието от кухни и изброени ресторанти, което им позволява да сравняват различни възможности и менюта, предлагани от различни доставчици на храна.
2. **Поставяне на поръчката:** След като клиентите изберат желаните ястия, те посочват детайлите за доставката, включително адрес и време, и осъществяват плащането за поръчката си чрез онлайн платформата.
3. **Обработка на поръчката:** Собственикът на платформата получава част от плащането като комисионна такса за своите услуги, като след това препраща поръчката към избрания ресторант или доставчик за изпълнение.
4. **Доставка:** Поръчката може да бъде доставена от самия ресторант чрез технически ресурси, или чрез трета страна, специализирана в доставки на храна, което осигурява удобство и избор за клиентите.

Някои собственици на ресторанти се колебаят да започнат с онлайн поръчки на храна. Това може да изглежда като сложен и времеизтощителен процес за внедряване или труден за поддръжка.

Затова те или отказват или напълно делегират тази част. Но тогава цялостният процес става неизменяем или твърде скъп. Други решения са по-малко надеждни или пропускат твърде много функции.

От гледната точка на ресторанта този тип платформа трябва да е система, която не само улеснява поръчките, но и не ограничава техните приходи или сложността на бизнеса.

За да работи онлайн системата за поръчки на храна в тяхна полза трябва да се изпълнят няколко основни принципа:

1. **Финансов аспект:** Тя не трябва да изтощава печалбата на ресторанта, а по-скоро да допринася за увеличаването й чрез увеличаване на обема на поръчките и ефективността на обслужване.
2. **Лесна настройка:** Така те могат да се фокусират върху управлението на кухнята и обслужването на клиентите, вместо да отделят време за технологични аспекти.
3. **Минимална поддръжка:** Трябва да изисква минимална поддръжка, осигурявайки автоматизирани и надеждни процеси за приемане на поръчки и управление на доставки, без необходимост от често техническо вмешателство.
4. **Подобрение на клиентския опит:** Трябва да работи за собствениците на бизнеса, като предоставя лесен достъп до анализи и данни за поръчките, за да могат да оптимизират операциите си. Повече от това, трябва да предоставя удобство и персонализирани преживявания на клиентите, което да ги мотивира да използват приложението редовно.

Интересно е да се отбележи, че статистиките свидетелстват за следното:

95% от собствениците на ресторанти заявяват, че управлението на бизнеса им с помощта на съвременни технологии като онлайн система за поръчки на храна е по-ефективно. Тази статистика ясно показва, че ресторантьорският бранш прегръща дигитализацията, осъзнавайки нейните предимства както за бизнеса, така и за клиентите.

В подкрепа на това са и данните за предпочитанията на клиентите:

* 70% от клиентите предпочитат онлайн поръчки и доставки. Удобството и гъвкавостта, които онлайн платформите предлагат, ги правят предпочитан вариант за много хора.
* 63% от клиентите предпочитат да поръчат храна за вкъщи, вместо да посещават ресторанти със семействата си. Това подчертава тенденцията към по-домашен начин на живот и търсенето на удобни решения за хранене.

Има два основни начина за интегриране на приложения за доставка на храна по заявка на даден бизнес:

1. **Разработка по поръчка:** При този подход се разработва системата за поръчки на храна изцяло според нуждите на клиента. Това дава пълен контрол върху дизайна и функционалността, но изисква значителни финансови инвестиции и време за разработка.
2. **SaaS система за поръчки на храна:** Този подход позволява ползване на софтуер като услуга (SaaS) за система за поръчки на храна. С този вариант няма първоначални разходи и се получава достъп до пълната гама от функции и услуги, включително редовни актуализации и 24/7 поддръжка, срещу месечна абонаментна такса.

Някои от известните и популярни приложения за поръчки на храна в индустрията за доставка на храна са:

1. **UberEats:**

Платформа за онлайн доставка на храна, основана през 2014 г. в Сан Франциско, САЩ, от Uber Technologies Inc., позволяваща поръчки от местни ресторанти с доставка до дома.

1. **Swiggy:**

Индийска платформа за онлайн доставка на храна, основана през 2013 г. в Бангалор, предлага широка гама от ресторанти, хранителни стоки и други продукти.

1. **Zomato:**

Платформа за търсене и резервации на ресторанти, основана през 2008 г. в Ню Делхи, Индия, предлагаща и онлайн доставка на храна.

1. **FoodPanda:**

Международна платформа за онлайн доставка на храна, основана през 2012 г. в Берлин, Германия, оперираща в над 120 държави.

1. **Postmates:**

Платформа за доставка на храна и стоки, основана през 2011 г. в Сан Франциско, САЩ, предлагаща доставка от ресторанти, магазини и други търговци. Postmates е **практична алтернатива** за поръчка на храна или други стоки, освен от ресторанти.

1. **GrubHub:**

Платформа за онлайн доставка на храна, основана през 2004 г. в Чикаго, САЩ, предлагаща широка гама от ресторанти в над 2500 американски града.

1. **Seamless:**

Платформа за онлайн доставка на храна, основана през 2002 г. в Ню Йорк, САЩ, предлагаща доставка от ресторанти в над 100 американски града. Seamless е добър вариант за поръчка на храна в Ню Йорк и други големи американски градове.

Някои от платформите предлагат и:

1. **Начини за доставка от отделни ресторанти:** Някои ресторанти предлагат собствена доставка, освен доставката чрез платформата, често на по-ниски цени от платформите за доставка. **Може да имат по-бързи срокове за доставка** или по-голям избор от ястия, които не се предлагат чрез платформи.
2. **Възможност за поръчка на храна за взимане от място:** Можете да поръчате храна онлайн и да я вземете от ресторанта сами. Някои ресторанти предлагат отстъпки за поръчки за взимане.
3. **Програми за лоялност:** Някои платформи предлагат програми за лоялност, които ви дават точки или отстъпки за бъдещи поръчки. Някои програми за лоялност предлагат ексклузивни предимства като раннен достъп до нови менюта или покани за специални събития.

Няколко фактора, които са важни да се вземат предвид са:

1. **Таксите за доставка:** Повечето платформи начисляват такси за доставка, които могат да варират в зависимост от платформата, ресторанта и разстоянието до вас.
2. **Наличност на ресторанти:** Не всички ресторанти са партньори с всички платформи.
3. **Ограничения в местоположението:** Някои платформи са налични само в определени страни или градове.

## 2.2 Избор на технологии

Успешното завършване на дипломна работа, особено в сферата на софтуерната разработка, е силно зависимо от правилния избор на технологии. Те играят ключова роля в множество аспекти на проекта, определяйки неговата функционалност, ефективност, актуалност и устойчивост.

**Съответствие с изискванията на проекта:** Технологиите трябва да бъдат съобразени с конкретните нужди и цели на дипломната работа. Те трябва да са в състояние да поддържат желаната функционалност и да позволят да се постигнат желаните резултати.

**Ефективност и производителност:** Изборът на правилните програмни езици, библиотеки и инструменти е ключов фактор за съкращаване на времето за разработка, оптимизация на ресурсите и подобряване на производителността. Това от друга страна помага да се приложи фокус върху други съществените аспекти на проекта.

**Актуалност:** Използването на съвременни технологии е от съществено значение за осигуряване на висока конкурентоспособност на проекта.

**Поддръжка и общност:** Популярните технологии с налична документация и активна общност могат да окажат ценна помощ при решаване на проблеми, внедряване на подобрения и бъдещи разширения на даден проект. Поддръжката осигурява увереност и улеснява разработчика в процеса на разработка.

**Индивидуален подход:** Няма универсална формула за успеха. Най-подходящите технологии за даден проект зависят от неговата специфика, уменията на разработчика и наличните ресурси.

### 2.2.1 Microsoft SQL Server

A logo for a microsoft server

Description automatically generated

*Фиг. 3 – Microsoft SQL Server лого*

**Microsoft SQL Server** е релационна система за управление на бази данни (RDBMS), разработена от Microsoft. Първата й версия е пусната през 1993 г., а оттогава платформата претърпява непрекъснато развитие и усъвършенстване.

Днес SQL Server се отличава с множество предимства пред други системи за управление на бази данни:

1. **Безпроблемна интеграция с Microsoft технологии:** SQL Server се интегрира безпроблемно с други продукти от екосистемата на Microsoft, като Windows Server, Azure, Visual Studio и др. Тази интеграция улеснява работата и сътрудничеството с други софтуерни решения.
2. **Широк спектър от функционалности:** SQL Server предлага богато портфолио от функции, включително:
   1. Обработка на геопространствени данни
   2. Анализ на Big Data
   3. Инструменти за интеграция на данни
   4. Напреднали възможности за бизнес интелигентност (BI)
   5. Много други
3. **Сигурност и защита на данните:** Предоставени са следните мерки**:**
   1. Вградено криптиране
   2. Автентикация и авторизация
4. **Ефективно управление на данни:** SQL Server предоставя набор от инструменти за управление на бази данни, които обхващат:
   1. Резервно копиране и възстановяване на данни
   2. Мониторинг на производителността
   3. Оптимизация на производителността
5. **Възможности за облачно развитие:** Благодарение на тясната си интеграция с **Azure**, облачната платформа на Microsoft, **SQL Server** предоставя несравними възможности за управление на данни в облачна среда.
6. **Лесен за работа и програмиране** Приложението разполага с лесен за използване интерфейс, който прави работата с базата данни достъпна дори за неопитни потребители. Visual Studio и други инструменти предоставят средства за лесно програмиране и управление на бази данни.
7. **Добра поддръжка и обучение:** Налични са богати ресурси за обучение и документация, както и обширна общност от разработчици. Това улеснява процеса на учене и решаване на проблеми.

В заключение, Microsoft SQL Server се явява мощен, надежден и гъвкав инструмент за управление на данни, подходящ за широк спектър от приложения.

### 2.2.2 Kotlin

A logo of a company

Description automatically generated

Фигура 1 - Kotlin лого

**Kotlin** се появява на сцената на програмирането през 2011 г., разработен от JetBrains, компания, известна с мощните си инструменти за разработка, включително IntelliJ IDEA. Оттогава езикът изминава бърз път, превръщайки се в любим на мнозина, печелейки признание и широка употреба.

Главните предимства и характеристики на Kotlin биват:

**1. Съвместимост с Java:**

Едно от най-големите предимства на Kotlin е безпроблемната му интеграция с Java. Той е проектиран да работи с Java код и платформи, позволявайки на разработчиците на Java да се възползват от неговите предимства, без да се налага да пренаписват съществуващия си код.

**2. Красив и ясен синтаксис:**

Kotlin е известен със своя елегантен и изчистен синтаксис, наподобяващ на естествен език. В сравнение с Java, кодът на Kotlin е по-кратък, по-изразителен и по-лесен за четене. Тази простота води до по-малко грешки, по-добра поддръжка и по-приятно програмиране.

**3. Сигурност на нулевите пойнтери:**

Kotlin е език със статична типологизация, което елиминира риска от грешки, свързани с нулеви пойнтери – често срещан проблем в Java. Тази характеристика прави Kotlin по-сигурен и надежден език за програмиране.

**4. Функционални възможности:**

Kotlin е вдъхновен от функционалното програмиране, парадигма, фокусирана върху функциите и обработката на данни. Това му позволява да предложи елегантни решения за задачи, свързани с работа с данни и паралелно програмиране. Функционалните му възможности го правят предпочитан инструмент за разработчици, търсещи по-кратък, изразителен и тестваем код.

**5. Поддръжка от Google:**

Google официално приема Kotlin като език за програмиране за Android, платформата му за мобилни устройства. Това гарантира активна общност, богата екосистема от инструменти и библиотеки, както и постоянна поддръжка и развитие.

**6. По-добра производителност:**

В някои случаи Kotlin кодът може да се изпълнява по-бързо от Java код. Причината се крие в по-ефективната компилация и оптимизация на Kotlin.

**7. Многостранно приложение:**

Освен за Android разработка, Kotlin намира приложение и в други сфери:

* **Уеб разработка:** С помощта на фреймуърк-и като Ktor и Spring Boot, Kotlin може да се използва за изграждане на мащабируеми и сигурни back-end приложения.
* **Разработка на настолни приложения:** Kotlin е съвместим с мултиплатформени фреймуърк-и като NativeScript, позволявайки създаването на приложения за различни операционни системи.
* **Разработка на игри:** Kotlin е подходящ и за разработка на игри, благодарение на своята производителност и възможности за работа с 3D графика.

**8. Бъдеще на езика:**

Kotlin е сравнително млад език, но еволюира с бързи темпове, печелейки все повече популярност. С подкрепата на Google, активната си общност и непрекъснатото си развитие, Kotlin има всички предпоставки да се превърне в доминиращ език за програмиране в близко бъдеще.

Съвместимостта му с Java, елегантният синтаксис, функционалните възможности, поддръжката от Google и широкото му приложение го правят предпочитан избор за разработчици, търсещи гъвкав и ефикасен инструмент за своите проекти.

### 2.2.3 Spring Framework

A green text on a black background

Description automatically generated

Фигура 2 - Spring Лого

Spring е мощен и гъвкав фреймуърк за разработка на Java приложения, създаден от Rod Johnson и първоначално пуснат през 2003 година. Той предлага цялостна инфраструктура за разработка на Java приложения и е известен със своята модулност и лесна интеграция с различни технологии.

**Основни предимства на Spring:**

1. **Инверсия на контрола (IoC)**: IoC контейнерът на Spring позволява управление на зависимости чрез инжектиране на зависимости (Dependency Injection). Това улеснява тестването и подобрява модулността на приложението, като прави компонентите независими и лесни за подмяна.
2. **Аспектно-ориентирано програмиране (AOP)**: Spring поддържа AOP, което позволява разделяне на бизнес логиката от технически аспекти като логиране, управление на транзакции и сигурност. Това води до по-чист и по-лесен за поддръжка код.
3. **Модулност и гъвкавост**: Spring е проектиран като модулен фреймуърк с множество компоненти (модули) като Spring Core, Spring AOP, Spring ORM, Spring Security и други. Това позволява използването само на необходимите модули, което прави приложението по-ефективно и лесно за управление.
4. **Интеграция с различни технологии**: Spring се интегрира лесно с разнообразни технологии и фреймуъркове, включително JPA/Hibernate за персистенция, JMS за съобщения, JMX за управление и много други. Това улеснява разработката на сложни и разпределени приложения.
5. **Управление на транзакции**: Spring предлага мощна поддръжка за управление на транзакции, която позволява декларативно управление на транзакции чрез анотации или XML конфигурации. Това улеснява работата с бази данни и осигурява надеждност на операциите.
6. **Сигурност**: Spring Security е мощен модул за осигуряване на защита на приложенията, като предлага автентикация, авторизация и други функции за сигурност. Той е лесен за конфигуриране и поддръжка, като осигурява високо ниво на защита.
7. **Удобство за разработчиците**: Spring предлага множество удобства за разработчиците като опростена конфигурация, поддръжка на аннотации, Spring Expression Language (SpEL) и мощна интеграция с IDE-та. Това ускорява разработката и прави процеса по-интуитивен.
8. **Голяма общност и богата документация**: Spring разполага с голяма и активна общност, както и обширна документация. Това прави лесно намирането на решения на проблеми, споделянето на опит и научаването на добри практики.

Благодарение на лесната си интеграция с разнообразни технологии и мощната поддръжка на различни аспекти на разработката, Spring е предпочитан избор за разработка на съвременни Java приложения.

### 2.2.4 Retrofit

A green rectangle with white text

Description automatically generated

Фигура 3 - Retrofit лого

**Retrofit** е мощен и популярен фреймуърк за Java и Android, създаден от Square, който улеснява взаимодействието с RESTful уеб услуги. Той предлага лесен и удобен начин за извършване на HTTP заявки и обработка на отговорите им, като използва аннотации за дефиниране на API методите. Първоначално пуснат през 2013 година, Retrofit бързо се утвърждава като предпочитан избор за разработчиците, работещи с мрежови операции.

Основни предимства**:**

1. **Лесна употреба**: Retrofit значително опростява извършването на HTTP заявки чрез използването на аннотации за дефиниране на API методите. Разработчиците могат бързо и лесно да създадат интерфейси, които описват уеб услугите, с които ще работят. Това прави кода по-четим и поддържаем.
2. **Конвертиране на данни**: Retrofit автоматично конвертира JSON или XML отговорите в Java/Kotlin обекти, като използва конвертори като Gson, Moshi или Jackson. Това премахва необходимостта от ръчно парсване на отговорите и улеснява работата с данните, получени от уеб услугите.
3. **Поддръжка на различни HTTP методи**: Фреймуъркът поддържа всички основни HTTP методи като GET, POST, PUT, DELETE и PATCH. Това позволява гъвкавост при взаимодействие с различни типове уеб услуги и извършване на различни операции върху ресурсите.
4. **Управление на заявки и отговори**: Retrofit предлага вградена поддръжка за управление на HTTP заявки и отговори чрез интерфейса Call. Разработчиците могат лесно да изпращат синхронни или асинхронни заявки и да обработват отговорите или грешките, получени от сървъра.
5. **Интеграция с OkHttp**: Retrofit използва OkHttp като HTTP клиент по подразбиране, което осигурява висока производителност и надеждност. OkHttp предоставя множество функции като кеширане на отговори, поддръжка на HTTP/2 и уеб сокети, което прави Retrofit още по-мощен.
6. **Гъвкавост и разширяемост**: Фреймуъркът позволява лесна интеграция с други библиотеки и инструменти, като RxJava и Kotlin Coroutines, за асинхронно програмиране и по-удобна обработка на мрежовите заявки. Това улеснява разработката на сложни и реактивни приложения.
7. **Поддръжка на URL параметри и заглавки**: Retrofit предлага удобен начин за добавяне на URL параметри, заявки и заглавки към HTTP заявките чрез аннотации като @Query, @Path и @Header. Това улеснява конфигурирането на заявките и прави кода по-чист и четим.

Тези предимстваго правят изключително полезен за създаване на съвременни, ефективни и лесни за поддръжка мрежови приложения.

### 2.2.5 Cloudinary

A blue and white logo

Description automatically generated

Фигура 4 - Cloudinary лого

Cloudinary е водеща платформа за управление на медийни активи, която предоставя мощни инструменти за съхранение, трансформация, оптимизация и доставка на изображения и видеа.

**Основни характеристики и предимства на Cloudinary:**

1. **Съхранение в облака**: Cloudinary предоставя сигурно и мащабируемо съхранение за изображения и видеа. Това елиминира нуждата от локално съхранение и управление на файлове, като позволява лесен достъп и споделяне на медийни ресурси от всяка точка на света.
2. **Автоматична трансформация на медии**: Една от най-мощните функции на Cloudinary е възможността за автоматична трансформация на изображения и видеа. Платформата поддържа различни операции като преоразмеряване, изрязване, въртене, промяна на формат и качество, добавяне на водни знаци и филтри. Това се извършва в реално време чрез лесни за използване URL параметри.
3. **Оптимизация на изображения и видеа**: Cloudinary автоматично оптимизира медийните файлове, като намалява техния размер без компромис с качеството. Това подобрява скоростта на зареждане на уеб страниците и мобилните приложения, като същевременно намалява използването на честотната лента и съхранението.
4. **Доставка на съдържание чрез CDN**: Платформата използва глобална мрежа за доставка на съдържание (CDN), което гарантира бързо и надеждно доставяне на медийни ресурси до крайните потребители, независимо от тяхното географско местоположение. Това осигурява висока производителност и минимално време за зареждане.
5. **Интеграция с други платформи и инструменти**: Cloudinary предлага широка гама от SDK (Software Development Kits) и плъгини за лесна интеграция с популярни платформи и инструменти като WordPress, Shopify, Magento, както и различни програмни езици като JavaScript, Ruby, PHP, Python и други. Това улеснява внедряването и използването на услугата в различни проекти.
6. **Анализи и отчетност**: Платформата предоставя подробни анализи и отчети за използването на медийните ресурси. Потребителите могат да проследяват трафика, да анализират производителността и да получават информация за различни метрики като брой изгледи, честотна лента, грешки и други.
7. **Поддръжка на различни формати**: Cloudinary поддържа широк спектър от файлови формати за изображения (JPEG, PNG, GIF, SVG и други) и видеа (MP4, WebM, Ogg и други). Това осигурява гъвкавост и съвместимост с различни устройства и платформи.
8. **Сигурност и контрол на достъпа**: Платформата предлага различни механизми за сигурност и контрол на достъпа, като защита с парола, подписани URL адреси, SSL криптиране и други. Това гарантира, че медийните ресурси са защитени и достъпни само за оторизирани потребители.
9. **Поддръжка на AI и машинно обучение**: Cloudinary включва функции за автоматично разпознаване на обекти, лица и сцени в изображения и видеа, като използва алгоритми за машинно обучение и изкуствен интелект. Това улеснява търсенето и категоризирането на медийни ресурси.

### 2.2.6 Material design

A white triangle in a circle

Description automatically generated

Фигура 5 - Material design лого

Material Design е дизайнерска система, разработена от Google и представена през 2014 година. Целта ѝ е да създаде единен и последователен визуален и потребителски опит през различни платформи и устройства. Material Design комбинира класически принципи на добрия дизайн с иновации и технологии, за да предостави съвременен и интуитивен интерфейс.

**Основни характеристики и предимства на Material Design:**

1. **Принципи и философия**: Material Design е вдъхновен от физическия свят и неговите текстури, включително хартия и мастило. Основните принципи включват използването на реалистични светлинни ефекти, сенки и движения, които създават усещане за дълбочина и пространство. Целта е потребителите да се чувстват комфортно и естествено при взаимодействие с интерфейса.
2. **Йерархия и смисъл**: Материалните елементи в дизайна са организирани по начин, който създава визуална йерархия и яснота. Важните елементи са подчертани чрез размер, контраст и разположение, което помага на потребителите бързо да се ориентират и да намерят необходимата информация.
3. **Движение и анимация**: Анимациите играят важна роля в Material Design, като помагат за създаването на плавни и интуитивни преходи между различните състояния на интерфейса. Анимациите трябва да бъдат естествени и реалистични, като същевременно подпомагат потребителите да разберат причинно-следствените връзки между действията и резултатите.
4. **Отзивчив и адаптивен дизайн**: Material Design е проектиран да бъде отзивчив и адаптивен, което означава, че интерфейсите автоматично се настройват за различни устройства и екранни размери. Това включва използването на гъвкави решетки, отзивчиви изображения и адаптивни компоненти.
5. **Достъпност**: Материалният дизайн поставя силен акцент върху достъпността, като осигурява инструменти и насоки за създаване на интерфейси, които са лесни за използване от всички потребители. Това включва използването на контрастни цветове, четливи шрифтове и достъпни елементи за навигация.

**Готови ресурси и класове в Material Design:**

1. **Material Components for Android**: За разработчиците на Android приложения, Material Components for Android предлага широк набор от готови UI компоненти, като RecyclerView, CardView, BottomNavigationView и много други. Тези компоненти улесняват изграждането на модерни и привлекателни интерфейси, които следват насоките на Material Design.
2. **Material Theming**: Material Theming позволява на разработчиците да персонализират визуалния стил на своите приложения, като променят цветове, шрифтове и форми. Инструментите за темизация предоставят гъвкавост и възможност за адаптиране на интерфейса към специфичния бранд на приложението.
3. **Material Icons**: Google предоставя голяма колекция от икони, които са част от Material Design. Тези икони са лесни за интегриране и използване, като могат да бъдат персонализирани по размер и цвят, за да се впишат хармонично в дизайна на приложението или уебсайта.
4. **Android Studio и Material Design**: Android Studio, официалната среда за разработка на Android приложения, предоставя инструменти и шаблони, които улесняват прилагането на Material Design. С вградената поддръжка за Material Components и предварително подготвени шаблони, разработчиците могат бързо да започнат работа и да създадат приложения, съответстващи на най-добрите практики в дизайна.
5. **Документация и ресурси**: Google предлага обширна документация и ръководства за Material Design, които включват примери, кодови откъси и най-добри практики. Това помага на разработчиците да разберат и приложат принципите на Material Design в своите проекти.

# 3. Проектиране на системата

Проектирането е съществена фаза в разработката на всеки продукт. През този период трябва да бъдат ясно дефинирани целите на продукта. Също така, етапът на проектиране трябва да установи изискванията към приложението, крайните срокове за различните задачи и времевата рамка за изпълнение на всяка задача поотделно.

## 3.1 Спецификация на приложението

Мобилното приложение за поръчка на храна е дигитална платформа, която позволява на потребителите да поръчват храна от ресторанти. Тя се състои от два основни компонента: **frontend** и **backend**.

**Frontend** е частта от софтуерна система, с която взаимодействат потребителите. Тя е отговорна за визуализацията на информацията, събирането на данни от потребителя и изпращането им до **backend** (сървърната част).

Общи изисквания към Frontend:

1. **Удобен за потребителя интерфейс:** Приложението трябва да е лесно за навигация и използване, с интуитивен дизайн и ясна структура.
2. **Функционалности за търсене и филтриране:** Потребителите трябва да могат лесно да търсят ресторанти, ястия и категории, използвайки различни критерии.
3. **Детайлни описания на ястията:** Описанията на ястията трябва да бъдат информативни и атрактивни, включително снимки, цени и съставки.
4. **Кошница за пазаруване:** Потребителите трябва да могат да добавят ястия към кошницата си, да преглеждат съдържанието ѝ и да финализират поръчката си.
5. **Сигурни платежни методи:** Приложението трябва да предлага различни сигурни платежни методи, като например кредитни карти, дебитни карти, мобилни плащания и др.
6. **Проследяване на поръчката:** Потребителите трябва да могат да проследяват статуса на поръчката си в реално време.
7. **Система за оценка:** Потребителите трябва да могат да оценяват ресторанти и ястия, за да споделят своя опит.

**Backend** е сървърната част на софтуерна система, която е отговорна за съхраняване на данни, обработка на заявки от **frontend** (частта, с която взаимодействат потребителите) и изпълнение на бизнес логиката на приложението.

**Някои от основните изисквания към backend частта са:**

1. **Сигурност:**
   1. **Шифриране на данните:** Всички данни, предавани между backend и frontend, трябва да бъдат шифрирани, за да се предпазят от неоторизиран достъп.
   2. **Сигурно съхранение на данните:** Данните на потребителите и други чувствителни данни трябва да се съхраняват сигурно на сървъри, защитени от неоторизиран достъп.
   3. **Проверка на автентичност и авторизация:** Backend системата трябва да може да проверява автентичността на потребителите и да им предоставя разрешения за достъп до данни и функции.
   4. **Защита от атаки:** Backend системата трябва да бъде защитена от различни видове атаки, като например XSS, SQL injection и DDoS.
2. **Производителност:**
   1. **Мащабируемост:** Backend системата трябва да може да се мащабира хоризонтално, за да се справи с нарастващия брой потребители и поръчки.
   2. **Висока наличност:** Backend системата трябва да е с висока наличност, за да може да работи непрекъснато без прекъсвания.
   3. **Бързо време за реакция:** Backend системата трябва да може да обработва заявките на потребителите бързо и ефективно.
3. **Удобство за разработчици:**
   1. **Добре документиран код:** Backend кодът трябва да бъде добре документиран, за да може да се разбира и поддържа лесно.
   2. **Използване на утвърдени практики:** Backend разработката трябва да се основава на утвърдени практики и стандарти.
   3. **Лесно тестване:** Backend кодът трябва да бъде лесен за тестване, за да се гарантира неговата функционалност и надеждност.
4. **Интеграция:**
   1. **Интеграция с други системи:** Backend системата трябва да може да се интегрира с други системи, като например системи за доставка на храна, платежни системи и системи за управление на взаимоотношенията с клиенти (CRM).
   2. **API:** Backend системата трябва да предостави API, което да позволи на други разработчици да се интегрират с нея.
5. **Гъвкавост:**
   1. **Модулна архитектура:** Backend системата трябва да има модулна архитектура, за да може да се адаптира лесно към нови изисквания.
   2. **Лесно промяна:** Backend кодът трябва да бъде лесен за промяна, за да може да се добавят нови функции и да се коригират грешки.
6. **Наблюдение:**
   1. **Система за мониторинг:** Backend системата трябва да има система за мониторинг, която да следи нейната производителност, сигурност и други важни показатели.
   2. **Система за записване на лог файлове:** Backend системата трябва да записва лог файлове, които да могат да се използват за отстраняване на проблеми и анализ.
7. **Поддръжка:**
   1. **План за поддръжка:** Backend системата трябва да има план за поддръжка, който да гарантира, че ще бъде поддържана и актуализирана.
   2. **Документация за поддръжка:** Backend системата трябва да има документация за поддръжка, която да помогне на разработчиците да я отстраняват и да я поддържат.

Тази спецификация е само обща рамка и конкретните изисквания могат да варират в зависимост от нуждите на бизнеса.

## 3.2 Задачи за изпълнение

1. **Проектиране на Архитектурата:**

Проектиране на ясна модулна архитектура, която да улесни разработката, тестването и поддръжката на приложението, както и да отговаря на изискванията за бързина и сигурност.

1. **Разработване на Софтуер:**

Изграждане на Backend:

* 1. **База данни:** Разработване на надеждна и мащабируема база данни за съхранение на данните на приложението, съобразена с нуждите на проекта.
  2. **Контролери:** Създаване на API контролери, които да обработват заявки от клиента, да валидират данни, да изпълняват бизнес логиката и да връщат appropriate responses.
  3. **Автентикация:** Имплементиране на JSON Web Token (JWT) автентикация за сигурно удостоверяване на потребители, валидиране на сесии и контрол на достъпа до ресурси, съобразно ролите на потребителите.

Изграждане на Frontend:

* 1. **Удобен за потребителя дизайн:** Разработване на атрактивен и интуитивен интерфейс, съобразен с най-добрите практики за UX/UI дизайн, за да се осигури приятно и безпроблемно потребителско изживяване.
  2. **ViewModel архитектура:** Прилагане на ViewModel архитектура за управление на потока от данни и поддържане на "life-cycle" на данните при навигация през различните екрани на приложението.
  3. **Локализация:** Имплементиране на локализация на приложението, за да може то да се използва от потребители на различни езици, разширявайки неговата аудитория.

Дефиниране на подходящи DTO структури за трансфериране на данни между frontend и backend, оптимизирайки комуникацията и намалявайки ненужния трафик.

Интеграция с Google Maps API – за извличане и показване на данни за местоположения

Създаване на лесен за ползване интерфейс за изпращане на заявки от клиента до backend, оптимизирайки процеса на комуникация и взаимодействие

1. **Сигурност:**

Осигуряване на криптиране на данните, предавани между клиента и сървъра, за да се защити поверителността и целостта на информацията.

1. **Тестване и Оценка:**

Създаване на автоматични тестове за тестване на разписаните заявки за комуникация с Backend

## 3.3 Архитектура

### 3.3.1 MVC

A diagram of a diagram

Description automatically generated

Фигура 6 - MVC диаграма

MVC (Model-View-Controller) е архитектурен шаблон за проектиране, широко използван в разработката на софтуер, особено за уеб приложения. Той разделя приложението на три взаимосвързани части:

**1. Модел (Model):**

* Съхранява данните на приложението и бизнес логиката.
* Не взаимодейства директно с потребителя.
* Достъпът до данните и бизнес логиката от него се осъществява чрез контролера.
* Примерни компоненти на модела: База данни, бизнес правила, валидатори.

**2. Изглед (View):**

* Отговорен за визуализацията на данните пред потребителя.
* Получава данни от модела чрез контролера.
* Не модифицира директно данните в модела.
* Взаимодейства с потребителя, за да получи обратна връзка.

**3. Контролер (Controller):**

* Посредник между модела и изгледа.
* Получава заявки от потребителя чрез изгледа.
* Взаимодейства с модела, за да извърши необходимите действия с данните.
* Актуализира изгледа с данните от модела.
* Примерни компоненти на контролера: API контролери, бизнес правила, валидатори.

MVC в действие:

**1. Потребителят взаимодейства с изгледа:**

* Потребителят кликва върху бутон, въвежда информация във формуляр или извършва друго действие в изгледа.

**2. Изгледът изпраща заявка до контролера:**

* Изгледът генерира HTTP заявка, съдържаща информацията от взаимодействието на потребителя.
* Заявката се изпраща до контролера, който е конфигуриран да обработва този тип заявки.

**3. Контролерът взаимодейства с модела:**

* Контролерът декодира информацията от заявката и я използва, за да взаимодейства с модела.
* В зависимост от типа на заявката, контролерът може:
  + Да извлече данни от модела.
  + Да актуализира данни в модела.
  + Да изпълни бизнес правила.

**4. Контролерът актуализира изгледа:**

* След взаимодействието с модела, контролерът формира отговор, съдържащ данните, които трябва да бъдат визуализирани.
* Отговорът се изпраща обратно на изгледа.

**5. Изгледът визуализира данните:**

* Изгледът използва получените от контролера данни, за да обнови визуалното представяне за потребителя.
* Могат да се генерират нови HTML страници, да се актуализира съдържанието на съществуващи страници или да се изпълни JavaScript код.

Предимства на MVC архитектурата:

* **Разделяне на отговорностите:** MVC разделя ясно отговорностите на приложението, което го прави по-лесно за разбиране, разработка и поддръжка.
* **Повторно използване на код:** Моделите, изгледите и контролерите могат да се използват повторно в различни части на приложението, което намалява дублирането на код.
* **Тестване:** MVC архитектурата улеснява тестването на отделните компоненти на приложението.
* **Мащабируемост:** MVC приложенията са по-мащабируеми, тъй като отделните компоненти могат да се разширяват и модифицират независимо.

### 3.3.2 MVVM

A diagram of a model

Description automatically generated

Фигура 7- MVVM диаграма

Model-View-ViewModel архитектурата е шаблон за проектиране, използван в разработването на мобилни приложения, за да се раздели потребителският интерфейс (UI) от бизнес логиката. Тя е подобна на MVC (Model-View-Controller) архитектурата, но с някои ключови разлики.

**Основни компоненти:**

* **Модел (Model):** Съхранява и управлява данните на приложението.
* **Изглед (View):** Определя визуалното представяне на данните и взаимодейства с потребителя.
* **ViewModel (Модел-изглед):** Действа като посредник между модела и изгледа. Подготвя данните за изгледа и обработва действията на потребителя.

**Взаимодействие между компонентите:**

1. **Изгледът наблюдава ViewModel:** Изгледът се абонира за промени в свойствата на ViewModel.
2. **ViewModel наблюдава Model:** ViewModel наблюдава промени в Model и актуализира своите свойства съответно.
3. **Потребителят взаимодейства с изгледа:** Потребителят докосва бутони, въвежда текст или извършва други действия в изгледа.
4. **Изгледът уведомява ViewModel:** Изгледът изпраща събития до ViewModel, за да сигнализира за действията на потребителя.
5. **ViewModel обработва действията:** ViewModel обработва получените събития, актуализира Model (ако е необходимо) и уведомява изгледа за промените.
6. **Изгледът се актуализира:** Изгледът се актуализира, за да отрази промените в свойствата на ViewModel.

Предимства на ViewModel архитектурата:

* **Разделение на отговорностите:** Ясно разделя UI от бизнес логиката, което улеснява тестването, поддръжката и повторното използване на код.
* **Тестване:** ViewModel може да се тества самостоятелно, без да е необходимо да се взаимодейства с UI.
* **Гъвкавост:** Подходяща е за различни платформи и UI библиотеки.
* **Повторно използване на код:** ViewModel може да се използва повторно в различни части на приложението.

## 3.4 Функционален дизайн

В приложението има три вида потребители, като всеки от тях има достъп до различни функционалности:

**1. Клиент:**

**Функции:**

* + Разглеждане на ресторанти и техните ястия.
  + Запазване на ресторант в “Любими”.
  + Поръчка на ястия от ресторант.
  + Оставяне на отзиви за ресторанти.
  + Разглеждане на направени поръчки
  + Разглеждане на направени отзиви
  + Смяна на език на приложението
  + Вход в приложението
  + Изход от приложението

**2. Ресторант:**

При регистрация ресторантът, трябва да бъде одобрен от администратор за да може да бъде виждан от клиенти. Останалите му функционалности са достъпни веднага след регистрация.

**Функции:**

* + Създаване на ястия
  + Изтриване на ястия
  + Скриване на ястия
  + Добавяне/изтриване на промоция
  + Проследяване на статуса на поръчките.
  + Разглеждане на отзиви от клиенти.
  + Разглеждане на история на продажбите.
  + Редакця на профил
  + Смяна на език на приложението
  + Вход в приложението
  + Изход от приложението

**3. Администратор:**

**Функции:**

* + Управление на потребители (клиенти, ресторанти).
  + Смяна на статус на ресторант
  + Управление на типове храни
  + Смяна на език на приложението
  + Вход в приложението
  + Изход от приложението

**4. Неоторизиран потребител:**

**Функции:**

* + Регистрация
  + Смяна на език на приложението
  + Вписване в приложението

Тип Клиент и Ресторант могат да бъдат избрани при регистрация, като преди нея потребителят има достъп само за смяна на езика на приложението и до опит за вход.

### 3.4.1 Use Case диаграма

**Диаграмите на случаите на употреба** са вид UML диаграма, използвана за визуализация на взаимодействията между **външни потребители** (актьори) и **системата**, която се моделира.

**Цел:**

* Да се **опишат** функционалностите на системата от гледна точка на потребителя.
* Да се **идентифицират** различните типове потребители и техните взаимодействия със системата.
* Да се **дефинират** бизнес изискванията на системата.

**Основни елементи:**

* **Актьори:** Представляват външни потребители или системи, които взаимодействат със системата.
* **Системи:** Представляват самата система, която се моделира.
* **Случаи на употреба:** Представляват набор от стъпки, които актьорът предприема, за да постигне определена цел.
* **Връзки:** Показват как актьорите взаимодействат със системата и с отделните случаи на употреба.

A diagram of a diagram

Description automatically generated

Фигура 8 - Use case диаграма на Клиент

A diagram of a diagram

Description automatically generated with medium confidence

Фигура 9 - Use case диаграма на Ресторант

A diagram of a diagram

Description automatically generated

Фигура 10 - Use case диаграма на Администратор

A diagram of a diagram

Description automatically generated

Фигура 11 - Use case диаграма на Неоторизиран потребител

# 4. Реализиране на системата

Реализирането на приложението може да се раздели на две основни части: разработка на **backend** и разработка на **frontend**. Разработването на backend-а е отговорно за бизнес логиката на приложението, обработката на данни, сигурността и взаимодействието с базата данни. Разработването на frontend-а е отговорно за потребителския интерфейс, UX/UI дизайна и взаимодействието с потребителя. Използван е Kotlin и за двете части за да се спестят време и усилия и да се осигури по – бързо интерпретиране при едновременната им разработка.

## 4.1 Система за контрол на версиите

За двете части на приложението използвах GitHub като платформа за контрол на версиите. **GitHub** е популярна платформа за хостинг на код, която предлага множество предимства за управление на версиите на софтуерни проекти.

Някои от **предимствата** на GitHub са:

* **Съхранение на код в облака:** Кодът на проекта се съхранява в онлайн хранилище, което премахва риска от загуба на проекта при проблем с хардуера на локалния компютър.
* **Интегриран е с Git**: Позволява да се използват всички функции на Git чрез GitHub.
* **Проследяване на промените:** GitHub записва история на всички промени, направени в кода, което позволява да се проследява кой е направил коя промяна и кога, което улеснява идентифицирането и отстраняването на грешки.

## 4.2 База данни

### 4.2.1 Code first:

Използван е code first подход за изграждане на базата данни. Първо са дефинирани така наречените Entity класове, като всеки от тях отговаря за структурата на дадена таблица, а след това се генерира съответстващата база данни. След това фреймуърк-а автоматично създава таблиците спрямо описаните Entity модели.

A computer screen shot of a program

Description automatically generated

Фигура 12 - Пример за Entity

### 4.2.2 Структура:

A screenshot of a computer

Description automatically generated

Фигура 13 - Диаграма на базата данни

Таблица **“users” –** в тази таблица се съдържат всички потребители на системата

|  |  |
| --- | --- |
| id | bigint |
| email | nvarchar(32) |
| name | nvarchar(32) |
| pass\_word | nvarchar(32) |
| role | smallint |

Таблица **“refresh\_tokens” –** в тази таблица се съдържат токени за опресняване на достъпа до системата на всеки потребител

|  |  |
| --- | --- |
| id | bigint |
| token | nvarchar(255) |
| expiry\_date | datetimeoffset(6) |
| user\_id | bigint |

Таблица **“restaurants” –** в тази таблица се съдържат ресторантите

|  |  |
| --- | --- |
| id | bigint |
| name | nvarchar(64) |
| address | nvarchar(255) |
| image\_url | nvarchar(255) |
| price\_range | smallint |
| owner\_id | bigint |
| food\_type\_id | bigint |
| status | smallint |
| status\_note | nvarchar(255) |
| phone\_number | nvarchar(16) |

Таблица **“food\_types” –** в тази таблица се съдържат типовете храни

|  |  |
| --- | --- |
| id | bigint |
| name | nvarchar(64) |
| name\_english | nvarchar(64) |

Таблица **“meals” –** в тази таблица се съдържат ястията предлагани от ресторантите

|  |  |
| --- | --- |
| id | bigint |
| name | nvarchar(64) |
| description | nvarchar(255) |
| has\_promotion | bit |
| image\_url | nvarchar(255) |
| is\_hidden | bit |
| price | float |
| promotion\_percent | int |
| promotion\_type | smallint |
| restaurant\_id | bigint |

Таблица **“orders” –** в тази таблица се съдържат поръчките

|  |  |
| --- | --- |
| id | bigint |
| address | nvarchar(255) |
| delivered\_on | datetime2(6) |
| delivery\_price | float |
| ordered\_on | datetime2(6) |
| payment\_method | smallint |
| status | smallint |
| restaurant\_id | bigint |
| user\_id | bigint |

Таблица **“order\_items” –** в тази таблица се съдържат поръчаните ястия на поръчките

|  |  |
| --- | --- |
| id | bigint |
| count | int |
| meal\_id | bigint |
| order\_id | bigint |

Таблица **“reviews” –** в тази таблица се съдържат ревютата на клиентите

|  |  |
| --- | --- |
| id | bigint |
| feedback | nvarchar(255) |
| rating | int |
| restaurant\_id | bigint |
| user\_id | bigint |

Таблица **“favorite\_restaurants” –** в тази таблица се съдържат ресторантите, които всеки клиент е запазил като любими

|  |  |
| --- | --- |
| user\_id | bigint |
| favorite\_restaurant\_id | bigint |

## 4.3 Сървър

Използвайки Spring MVC, сървърната част е реализирана като REST API, осигуряващо лесно интегриране с мобилното приложение. Spring Security е интегриран за осигуряване на защита на приложението и удостоверяване на потребителите. Spring Boot е използван за опростяване на конфигурацията и стартирането на приложението, улеснявайки разработката и тестването.

### 4.3.1 Структура

A screenshot of a computer

Description automatically generated

Фигура 14 - Структура на backend проект

Логиката на **backend** проекта е разделена на няколко части:

В **Config** се съдържат класове за:

* Управление на аутентификацията на потребителите: Определят се роли, създават се beans(обекти, които се съхраняват и управляват от контейнер за зависимости) за потребителски детайли и кодиране на пароли
* Конфигуриране на JWT аутентификация за валидиране на токени
* Дефиниране на права за авторизация: Разрешаване на достъп до определени маршрути(регистрация, вход) без аутентификация и изискване на така за останалите API маршрути
* Настройка на CORS(механизъм за сигурност, който ограничава достъпа на от един домейн до ресурси от друг домейн) и други защитни функциии
* Филтър клас за проверка на наличност на JWT токен във всяка заявка към сървъра. Ако има валиден token, извлича потребителската информация и я предоставя на Spring Security за удостоверяване.

A screen shot of a computer

Description automatically generated

Фигура 15 - Кконфигурация на достъп до маршрути

В **Controller** се съдържат класове за обработка на заявки към различни API endpoint – и. Тe извикват съответните **Service** класове, в които се съдържа бизнес логиката на приложението. В тях може да се опише и какъв достъп имат отделните роли към заявките.

A screenshot of a computer program

Description automatically generated

Фигура 16 - Контролер клас за ресторанти

В **Dto** се съдържат **Data Transfer Object** класове, които са прости обекти, които съдържат само данните, необходими за конкретната задача, за която са предназначени, в този случай данните нужни само за работа на определената заявка. Като заявката работи с два такива обекта, един за заявяване и един за отговор.

A computer screen shot of a program

Description automatically generated

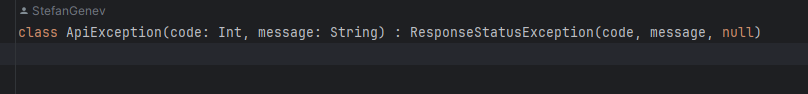
Фигура 17 - Dto за промяна по тип на храна

A screenshot of a computer

Description automatically generated

Фигура 18 - Dto за отговор от промяна по тип на храна

В **Exception** се съдържаткласове за дефиниране налогика която се извиква при прекъсване на нормалния поток на изпълнение на програмата.



Фигура 19 - Клас за сигнализиране на HTTP грешки с персонализиран код и съобщение

В **Model** се съдържат Entity класовете отговарящи за отделните таблици в базата данни. Там се съдържа и класът, който взема информация от даден потребител, като роля, парола и потребителско име и я подава на Spring Security, за да се провери неговият достъп до заявките.

A screen shot of a computer program

Description automatically generated

Фигура 20 - Клас извличаш данни нужни за аутентификация на потребител

В **Repo** се съдържат Repository класове работещи с базата данни. Тези класове позволяват избягването на писане на SQL, като вместо това се използва името на дадената функция.

A computer screen shot of a program code

Description automatically generated

Фигура 21 - Repository клас за отзиви

В **Service** се съдържат класовете, в които се описва бизнес логиката използвана при извикването на заявка към сървъра. Това включва и логиката за генериране на JWT токен при вход или регистрация на даден потребител.

A screenshot of a computer program

Description automatically generated

Фигура 22 - Service клас за генериране на JWT токени

В **Utils** се съдържат полезни класове свързани с различни функционалности.

A screenshot of a computer program

Description automatically generated

Фигура 23 - Клас за генериране на стандартизирани HTTP отговори

В **resources** са дефинирани два файла от по - голяма значимост:

* **application.properties :** Конфигурира връзка с базата данни, задава тип на лог на съобщения и дефинира секретния ключ за JWT токените, както и тяхното време на валидност

**A screenshot of a computer

Description automatically generated**

Фигура 24 - application.properties файл

* **data.sql :** Файл със скриптове, който се изпълнява автоматично при инициализация на базата данни

## 4.4 Frontend

Frontend проекта е реализиран чрез Android Studio, като в него са описани UI елементите и взаймодествието им и данните който получават с backend, чрез изпращането и обработването на асинхронни заявки.

### 4.4.1 Структура

A screenshot of a computer program

Description automatically generated

Фигура 25 - Структура на Frontend проекта