

ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ

ФАКУЛТЕТ ПО КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ

КОМПЮТЪРНО И СОФТУЕРНО ИНЖЕНЕРСТВО

**Програмиране за мобилни
устройства:**

Курсов проект:

, „Cafe-Dash“

Имена:	Факултетен номер:	Група:
Ангел Любомиров Стойнов	121222150	406
Запрян Запрянов	121222157	406
Васил Стойчев Стойков	121222094	406

Съдържание:

Увод:	4
Анализ на съществуващи разработки:	5
Проектиране:	7
Кой ще използва продукта?	7
Какви данни ще се използват?	7
Как ще бъдат достъпни функционалностите от потребителя?	11
1. Клиент	11
2. Собственик на кафене.....	12
3. Служител на кафене.....	13
4. Администратор.....	13
Концептуален модел на Cafe-Dash	14
Софтуерна архитектура на система:	15
Mobile App (React Native).....	16
Backend (Spring Boot):	16
Database (PostgreSQL).....	16
Azure Blob Storage.....	16
3rd Party Integration: EmailJS	16
Декомпозиция на системата на модули:	17
UML class диаграми за определени заявки	17
UML sequence диаграми за определени заявки	22
Спецификация на изискванията:	26
Функционални изисквания:	26
Нефункционални изисквания:	27
Потребителски истории:	28
Backend реализация:	33
Технически спецификации:	33
Файлова структура на backend:	34
Controller:	35
Service:	36
Repository:	38
Mapper:	39
DTO/Entity:.....	40
DTO:	40
Entity:.....	42
Exceptions:	43
Frontend реализация:	52
Файлова структура:	52
Assets:	53
Components:.....	53
Screens:	57
Navigation:	63

Services:	63
Hooks:	65
Context:	66
Theme:	67
Types:	67
Utils:	67
Допълнителни функционалности:	68
Deployment реализация:.....	70
Архитектура:	70
Backend:	70
База данни:.....	70
Съхранение на изображения:.....	71
Конфигурация:	71
Docker конфигурация:	71
Makefile:	72
Потребителско ръководство:.....	74
Общ преглед на някои екрани:.....	74
Начин на употреба:.....	83
Логин:	83
Правене на поръчка:.....	86
Обработка на поръчка:.....	89
Оставяне на ревю:.....	91
Свържи се с кафетериията:	96
Промяна на паролата:.....	103
Добавяне на продукт в кафетерия:.....	108
Принос и отговорност:	115
В процеса на разработка:	115
Бъдещи промени:	115
Необходими промени:.....	115
Заключение:	116
Github:	116
Използвана литература и материали:	11116

УВОД:

Cafe-Dash е мобилно приложение, създадено е с една цел - да улесни намирането на качествени кафенета. Приложението разполага с интуитивен интерфейс за разглеждане на всички възможни кафетерии и техните продукти. Системата разполага с възможността за направа на поръчка, за оставяне на ревю, както и контактна форма за свързване със съответното кафене.

Държим нашето приложение да бъде базирано на съвременни технологии като: Spring Boot, React Native, Azure и други. Проектът цели да дигитализира и оптимизира процесите в заведенията за обществено хранене и отдих, с ограничен асортимент от предлагани продукти, сред които доминира кафето. Предлага функционалности както за крайните потребители, така и за администраторите, отговарящи за менютата и обслужването.

Анализ на съществуващи разработки:

1. Glovo

Glovo е популярна платформа за доставки, предлагаща широка гама от продукти (храна, лекарства, стоки от магазини и др.). Платформата разчита на опростен и интуитивен дизайн, който позволява лесна ориентация и бързо създаване на поръчки.

Недостатъци на Glovo:

- Ограничена избор на заведения в определени региони.
- В някои случаи доставката е по-бавна поради голямото натоварване.
- Потребителите често трябва да преминават през допълнителни стъпки при избор на услуги.
- Липса на ясна комуникация между клиент и заведение, освен чрез стандартна поддръжка.

Предимства на Cafe-Dash спрямо Glovo:

- Cafe-Dash предлага тясно специализиран интерфейс само за заведения, което ускорява процеса на избор и поръчка.
- Налични са директни ревюта и оценки от потребители, което повишава доверието и прозрачността.
- Предлага възможност за лесна комуникация между клиент и заведение.
- Позволява на самите заведения да управляват промоции и купони за отстъпки, което увеличава гъвкавостта и стимулира лоялността на клиентите.

2. Takeaway (Just Eat Takeaway)

Takeaway е добре установена платформа за онлайн поръчки на храна, ориентирана към голямо разнообразие от ресторани и кухни. Платформата предлага удобни функции като бързо повторение на поръчки и филтриране на ресторани по различни критерии.

Недостатъци на Takeaway:

- Бавна или недостатъчна реакция при проблеми с поръчките (ограничена клиентска поддръжка).
- Липса на детайлно GPS проследяване в реално време.
- Наличие на минимални стойности на поръчките и понякога завишени цени за доставка.

Предимства на Cafe-Dash спрямо Takeaway:

- Cafe-Dash осигурява директна комуникация и по-добро обслужване на клиентите благодарение на локалния подход.

- По-добра интеграция на обратната връзка чрез ревюта и оценки, директно видими за клиентите.
- Администраторите на заведенията имат по-голяма автономност в управлението на съдържанието и промоциите.

3. Uber Eats

Uber Eats е глобално известна платформа за доставка на храна, разполагаща с широка мрежа от ресторани и силна интеграция с услугите на Uber. Платформата е известна със своята бърза доставка и GPS проследяване на поръчките.

Недостатъци на Uber Eats:

- Високи допълнителни такси (доставка, обслужване, малки поръчки).
- Интерфейсът е визуално натоварен с множество промоции, което може да затрудни ориентацията на новите потребители.
- Липса на плащане в брой на много пазари, което ограничава достъпността за част от клиентите.

Предимства на Cafe-Dash спрямо Uber Eats:

- Cafe-Dash предоставя по-изчистен и лесен за навигация интерфейс, фокусиран само върху менюта и заведения.
- Поддържа гъвкави методи на плащане, включително плащане в брой.
- По-добра персонализация и възможност за адаптиране към специфични нужди на общности или локални пазари.

Изводи и конкурентно предимство на Cafe-Dash:

Cafe-Dash успешно идентифицира и преодолява много от слабите места на популярните услуги за поръчка на храна, като предлага:

- Локализирано и персонализирано обслужване.
- По-добро управление на обратната връзка и прозрачност в оценките.
- По-висока автономност и гъвкавост за заведенията, позволявайки им самостоятелно да управляват маркетингови и промоционални активности.

Cafe-Dash е оптимален избор за общности и малки заведения, търсещи лесна за използване, прозрачна и гъвкава система за поръчки, без ограничения, характерни за големите комерсиални платформи.

Проектиране:

Кой ще използва продукта?

Приложението е предназначено за **широк кръг потребители**, които имат различни роли и нужди в рамките на цялата платформа. То е създадено така, че да обхване **всички участници в процеса на предлагане и поръчване на кафе**, както от страна на клиента, така и от страна на бизнеса, включително и техническата поддръжка и управление на системата. По-долу са описани основните групи потребители и как точно те ще взаимодействват с приложението.

Приложението е насочено основно към **възрастова група 18–45 години**, но може да бъде използвано и от по-широк кръг потребители, в зависимост от нуждите и дигиталната им грамотност.

Видове потребители/роли:

- **Клиенти (крайни потребители)** - Клиентите са най-многобройната група потребители на приложението. Те са хора, които ежедневно консумират кафе и желаят да спестят време, като направят своята поръчка предварително чрез мобилното приложение. Това са работещи хора, студенти, туристи или просто любители на кафето, които не искат да чакат на опашка.
- **Собственици на бизнеси(кафенета)** - Собствениците са юридически или физически лица, които притежават или управляват кафене и искат да го включат в системата. Те използват приложението, за да **разширят клиентската си база** и да улеснят обработката на поръчки чрез централизирана система.
- **Служители на кафене** - Служителите са хора, които работят в конкретно кафене и имат роля, свързана с **обработка на поръчки**. Те взаимодействват с приложението ежедневно, за да виждат постъпващи заявки, да ги изпълняват и да комуникират с клиенти при нужда
- **Системен администратор** - Това е човек или екип, отговорен за следене на коректното функциониране на приложението, за модериране на съдържание и за контрол на потребителски действия. Задача на системния администратор също е да добавя нови кафенета/собственици при подписване на договор с нов бизнес така, че бизнесът да може да ползва функционалностите на приложението. Администраторът има право да изтрива/променя кафенета и потребителски профили

Какви данни ще се използват?

Приложението използва **различни типове данни**, необходими за неговото функциониране, комуникация между потребителите и осигуряване на основните бизнес процеси. Всички данни се съхраняват в базата дори след тяхното изтриване. При “изтриване” на данни от базата се обновява полето `is_deleted`, което не позволява на данните да бъдат извлечени от базата

Данните могат да бъдат разделени в няколко основни категории:

- **Лични данни на потребителя** - Това са данни, чрез които дадено физическо лице може да бъде директно или индиректно идентифицирано. Те се съхраняват сигурно и са обект на защита
 - **Основни данни:**
 - **username:** потребителско име (видимо в системата)
 - **email:** имейл адрес (използван за регистрация, логин, ресет на парола)
 - **password:** парола (съхранява се хеширана, нечетима за администратори)
 - **roles:** роли на потребителя (client, owner, employee, admin)
 - **is_deleted:** индикатор дали профилът е логически изтрит
- **Данни за възстановяване на достъп (токен за промяна на парола)** - такъв запис се създава когато потребител иска да смени своята парола. Този токен се ползва от системата, за да даде лимитирано време на един потребител за смени паролата си и същевременно показва на кой потребител принадлежи
 - **Основни данни**
 - **token:** уникален низ за възстановяване на парола
 - **expiry_date:** срок на валидност на токена
 - **user_id:** връзка между токена и потребителя
- **Данни за кафенетата (физическите обекти)** - тези данни се използват за цялостно описание на търговски обект (кафене). Тези данни са видими за всички потребители. Търговските имат уникално име, но могат да принадлежат на един бизнес (бранд)
 - **Основни данни**
 - **name:** име на кафенето
 - **brand:** марка или франчайз
 - **location:** адрес или описание на местоположението
 - **phone_number:** служебен телефон за контакт
 - **image_url:** снимка на заведението
 - **opening_hour и closing_hour:** работно време
 - **rating:** текуща оценка (изчислена от ревюта)
 - **count_reviews:** брой ревюта
- **Данни за менюто (продуктите) на кафенето** - описва всеки артикул предлаган от кафенето. Тези данни са видими за всички потребители. Клиентите използват тези данни, за да направят своята поръчка.
 - **Основни данни**
 - **name:** име на продукта (напр. Лате, Капучино)
 - **price:** единична цена
 - **product_type:** категория на продукта (напитка, храна)
 - **image_url:** изображение на продукта
 - **cafeteria_id:** връзка към кафенето, което предлага продукта
- **Данни за поръчки** - приложението съхранява данни за всички поръчки направени от потребителите. Тези данни се използват от служителите, за да могат да обработват/завършват поръчката, ползват се от клиентите за да следят за статуса

на своята поръчка, ползват се от администраторите и собствениците, за преглед и модификация.

- **Основни данни**

- **id:** уникален идентификатор на поръчката
- **user_id:** кой е направил поръчката
- **cafeteria_id:** от кое кафене е поръчката
- **status:** статус (напр. "в процес", "готова", "взета")
- **discount:** приложена отстъпка
- **tip:** бакшиш
- **ready_pickup_time:** очакван час за вземане

- **Продукти в поръчката (order_product)-** представлява междинна връзка между поръчките и продуктите и е създадена с цел да осигури гъвкаво и мащабирамо управление на съдържанието на всяка поръчка. Основната ѝ функция е да поддържа възможността в една поръчка да има множество артикули от един и същ тип продукт, като например 3 броя еспресо и 2 броя капучино. Без тази таблица не би било възможно коректно да се моделират такива сценарии, тъй като релацията между orders и product би била ограничена до един продукт на поръчка. За всеки различен продукт в една поръчка се създава един запис в тази таблица.

- **product_id:** кой продукт е поръчен
- **order_id:** връзка към поръчката
- **product_price:** цена в момента на поръчката
- **product_quantity:** брой от избрания продукт

- **Данни за ревюта/отзиви** - представляват мнения, оставени от клиенти за определени кафенета, които са използвали услугите им. Всеки отзив съдържа текстово съдържание и оценка, и е свързан с конкретен потребител и конкретно заведение. **Информират клиентите** като показват реални преживявания на други потребители, отзивите подпомагат процеса на вземане на решение при избор на кафене. Всички отзиви са публични и могат да бъдат преглеждани от останалите потребители в приложението.

- Основни съхранявани данни в ревютата:

- **title:** заглавие на отзива
- **body:** пълно текстово съдържание на мнението
- **rating:** чисрова оценка, например от 1 до 5
- **created_at:** дата и час на публикуване
- **user_id:** потребителят, който е оставил мнението
- **cafeteria_id:** кафенето, за което се отнася отзивът

- **Данни за роли на потребителите** - Приложението използва ролево-базиран контрол на достъпа, като всеки потребител може да има една или повече роли. Това са референтни данни, до които има достъп само системния администратор. Промяната на тези данни трябва да се правят изключително рядко и това се извършва само от администратора. Тази таблица съдържа само name

Връзки между данните

1. users ↔ orders

- Вид на връзката: Един потребител може да направи много поръчки (1:M)
- Описание: Таблицата orders съдържа външен ключ user_id, който показва кой потребител е направил конкретната поръчка.

2. users ↔ review

- Вид на връзката: Един потребител може да остави много отзиви (1:M)
- Описание: Полето user_id в review показва кой потребител е автор на съответния отзив.

3. users ↔ user_roles ↔ role

- Вид на връзката: Много към много (M:N)
- Описание: Един потребител може да има множество роли, и една роля може да принадлежи на много потребители. Таблицата user_roles реализира тази връзка като асоциативна таблица със следните полета:
 - user_id: връзка към users
 - role_id: връзка към role

4. users ↔ password_reset_token

- Вид на връзката: Един потребител може да има много токени за възстановяване (1:M)
- Описание: Всеки токен за ресет на парола е обвързан с конкретен потребител чрез user_id.

5. users ↔ review ↔ cafeteria

- Вид на връзката: Индиректна, но логически важна. Един потребител може да остави отзиви за много кафенета; едно кафе може да има отзиви от много потребители (M:N, реализирана чрез review)
- Описание: Таблицата review свързва users и cafeteria, като всяко ревю има user_id и cafeteria_id.

6. orders ↔ cafeteria

- Вид на връзката: Едно кафе може да получи много поръчки (1:M)
- Описание: В orders се намира външен ключ cafeteria_id, който посочва към кое заведение е направена поръчката.

7. orders ↔ order_product ↔ product

- Вид на връзката: Много към много (M:N)

- Описание: Една поръчка може да съдържа множество продукти, а един продукт може да бъде част от много поръчки. Това се реализира чрез асоциативната таблица `order_product`, която съдържа:

8. `product ↔ cafeteria`

- Вид на връзката: Едно кафене може да предлага много продукти (1:M)
- Описание: Таблицата `product` съдържа `cafeteria_id`, който указва заведението, в което се предлага съответният продукт.

9. `cafeteria ↔ review`

- Вид на връзката: Едно кафене може да има много отзиви (1:M)
- Описание: `cafeteria_id` в таблицата `review` указва за кое заведение е даден съответният отзив.

Как ще бъдат достъпни функционалностите от потребителя?

В приложението са дефинирани четири основни типа потребители: клиент, собственик на кафене, служител в кафене и администратор. Всяка роля има строго определени права и функционалности, които ѝ се предоставят чрез специално проектиран потребителски интерфейс. По този начин се гарантира сигурност, яснота и ефективност в управлението на задачите, специфични за всеки тип потребител.

1. Клиент

Клиентът е крайният потребител на приложението, който използва платформата с цел намиране, поръчване и получаване на кафе от близко кафене. След регистрация и вход в системата, клиентът има достъп до следните функционалности:

- Откриване на кафенета:
Чрез вграден филтър и геолокация клиентът може да открие най-близките кафенета, достъпни в системата, като търси по име, местоположение или тип продукти.
- Преглед на менюта и продукти:
Всяко кафене има собствено меню, което клиентът може да разгледа, като получава подробна информация за:
- Създаване на поръчка:
Чрез добавяне на продукти в количката клиентът може да оформи поръчка, да избере начин на плащане и да изпрати заявката към избраното кафене.
- Следене на поръчка в реално време:
Системата позволява на клиента да наблюдава състоянието на своята поръчка – от приемането, през приготвянето, до нейното приключване. При всяка промяна клиентът получава известие.

- Оставяне на отзиви и оценки:

След получаване на поръчката клиентът може да остави коментар и да оцени качеството на обслужване или продуктите. Тези оценки са публично видими и влияят на репутацията на кафенето.

- Профил и настройки:

Клиентът има достъп до:

- История на поръчките
- Смяна на парола
- Актуализиране на имейл и име
- Изход от системата

- Изпращане на сигнали (bug reports):

При откриване на грешка, потребителят може да изпрати директно съобщение до екипа чрез формуляр за обратна връзка.

2. Собственик на кафене

Собствениците представляват юридическите лица или физически представители на дадено заведение. Те получават достъп до административен изглед, който позволява цялостно управление на кафенето, неговото меню и персонал. Предоставените функционалности включват:

- Регистрация и настройка на кафене:

- Име, локация, изображение, телефон за връзка
- Работно време (начален и краен час)
- Преглед на статистики – общ рейтинг, брой ревюта

- Управление на менюто:

- Добавяне на нови продукти (име, цена, изображение, тип)
- Редакция на съществуващи продукти
- Изтриване на артикули, които вече не се предлагат

- Наблюдение и управление на поръчки:

- Преглед на текущи и приключени поръчки
- Анализ на често поръчвани артикули
- Обработка на заявки в реално време (в сътрудничество със служителите)

- Управление на служители:

- Добавяне на нови служители към кафенето
- Назначаване или премахване на роли
- Преглед на активности и производителност

- Достъп до анализи и статистики:

- Визуализация на брой поръчки на ден
- Оценки от потребители
- Средно време за изпълнение на поръчка

- Подаване на сигнали:

- Система за обратна връзка при проблем с платформата

3. Служител на кафене

Служителите са регистрирани от собственика на заведението и имат ограничен достъп, съобразен със служебните им задължения. Интерфейсът е опростен и ориентиран към бързо и ефективно обслужване на поръчки.

- Получаване на нови поръчки в реално време
- Преглед на съдържанието на поръчката и инструкции от клиента
- Обработка на поръчката: маркиране като приета, в процес или изпълнена
- Комуникация с клиента, ако се налагат уточнения
- Преглед на дневна активност и изчисление на натовареност
- Подаване на технически сигнали

4. Администратор

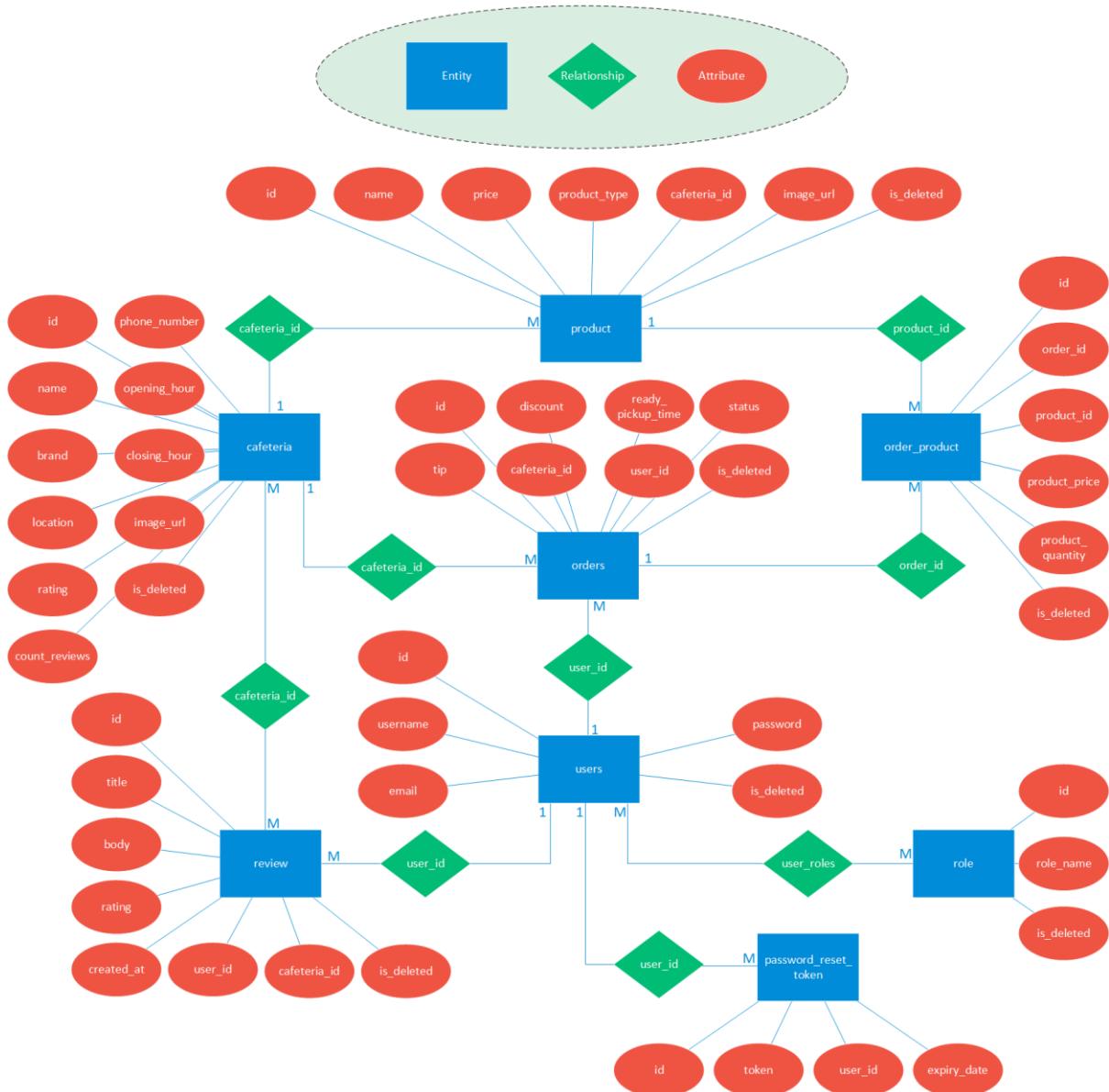
Администраторът има най-високо ниво на достъп в системата и отговаря за цялостния контрол на платформата. За разлика от собственика, администраторът не представлява конкретно кафене, а системата като цяло. Неговите възможности включват:

- Преглед и модерация на:
 - Всички потребителски акаунти
 - Всички регистрирани кафенета
 - Всички продукти в системата
 - Всички направени поръчки и оставени ревюта
- Управление на съдържание:
 - Одобряване или премахване на нови кафенета
 - Ръчно прекратяване на акаунти
 - Промяна на роли на потребители (например: клиент → служител)
- Системна сигурност и контрол:
 - Преглед на логове и опити за неоторизиран достъп
 - Модериране на активността

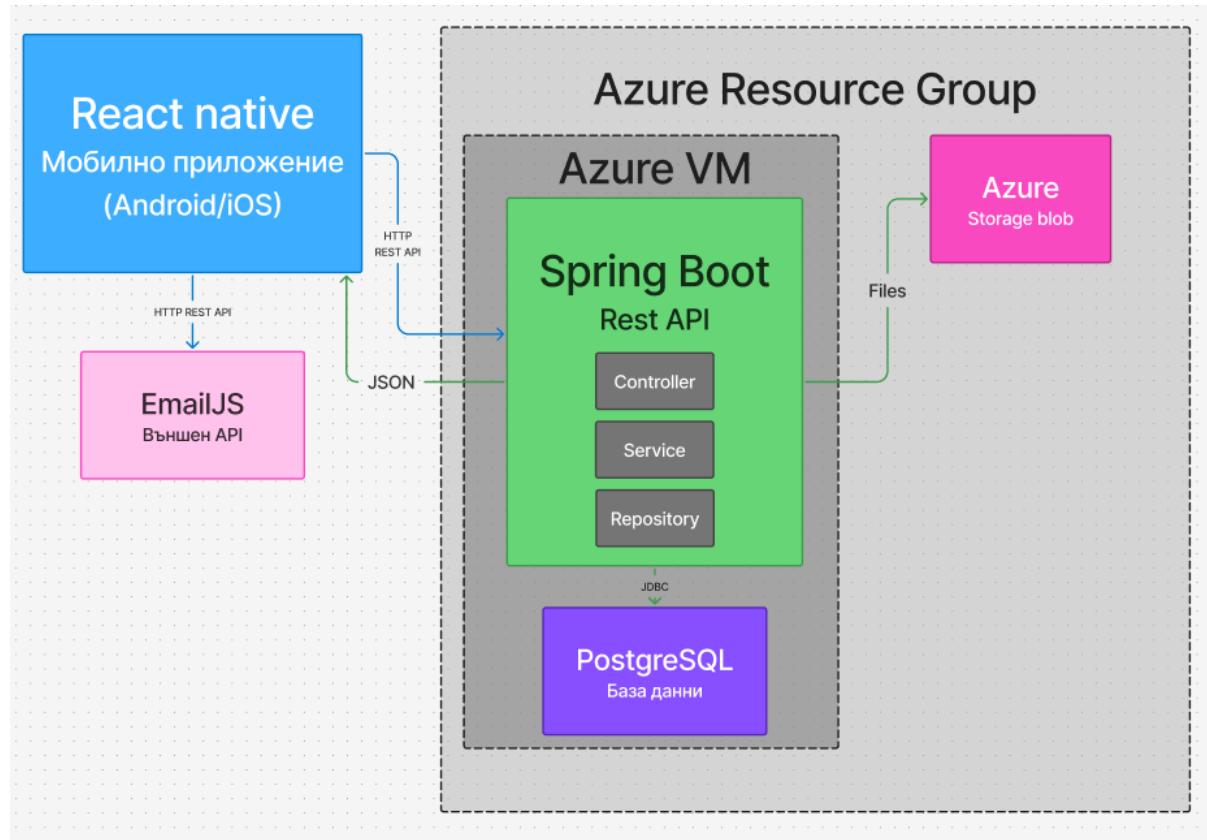
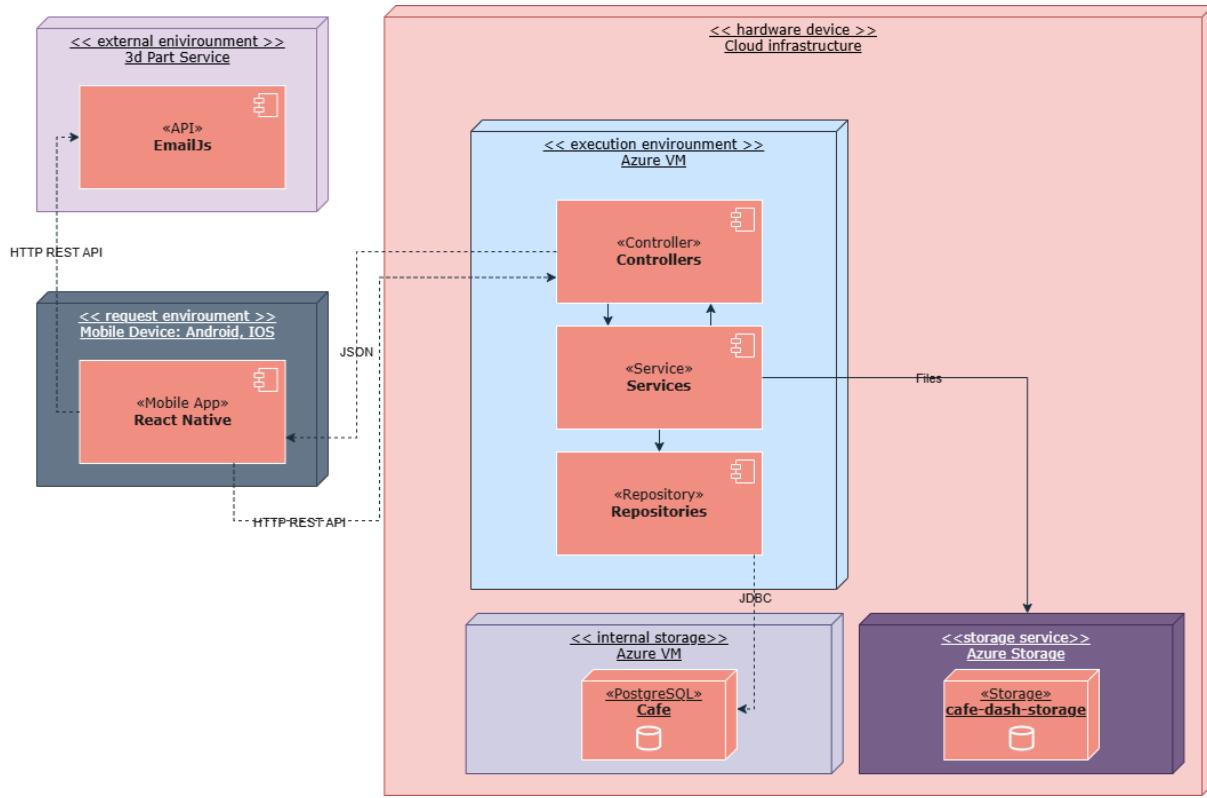
Ограничения на администратора:

- Няма достъп до пароли или възможност да ги променя от чуждо име
- Не може да редактира или изтрива отзиви, публикувани от клиенти

Концептуален модел на Cafe-Dash



Софтуерна архитектура на система:



Mobile App (React Native)

Изпълнява се на мобилни устройства с Android и iOS.

Комуниира с backend чрез HTTP REST API.

Изпраща и получава JSON данни.

Backend (Spring Boot):

Хостната в Azure Virtual Machine.

Основни слоеве:

- Controllers: Приемат HTTP заявки.
- Services: Съдържат бизнес логиката.
- Repositories: Достъп до базата данни чрез JPA/JDBC.

Database (PostgreSQL)

Хостната локално в същата Azure VM.

Достъпът до нея се осъществява чрез JDBC от backend приложението.

Денните не са достъпни отвън.

Azure Blob Storage

Използва се за съхранение на файлове - изображения.

Backend-ът комуниира директно с Blob Storage за upload и достъп на файлове.

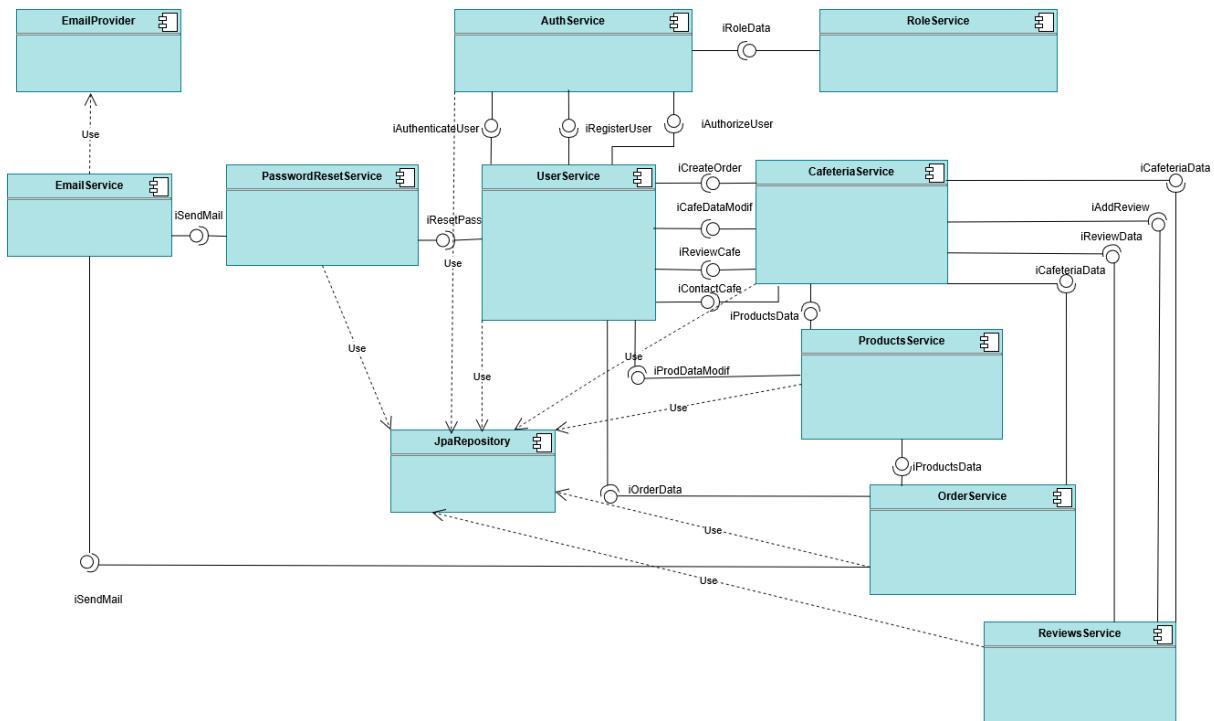
Blob Storage е конфигуриран с CORS правила, за да се осигури директен достъп от мобилното приложение.

3rd Party Integration: EmailJS

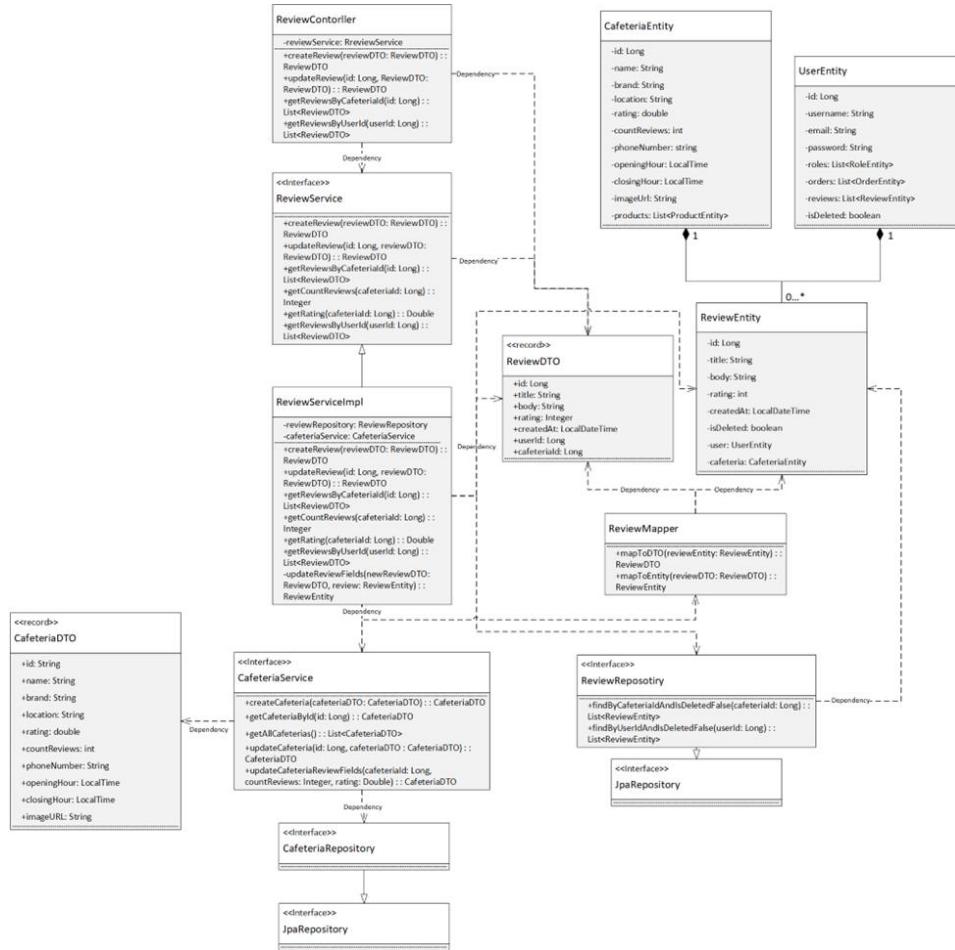
Използва се за изпращане на имейли - контактна форма.

Достъпът се осъществява чрез HTTP REST API.

Декомпозиция на системата на модули:

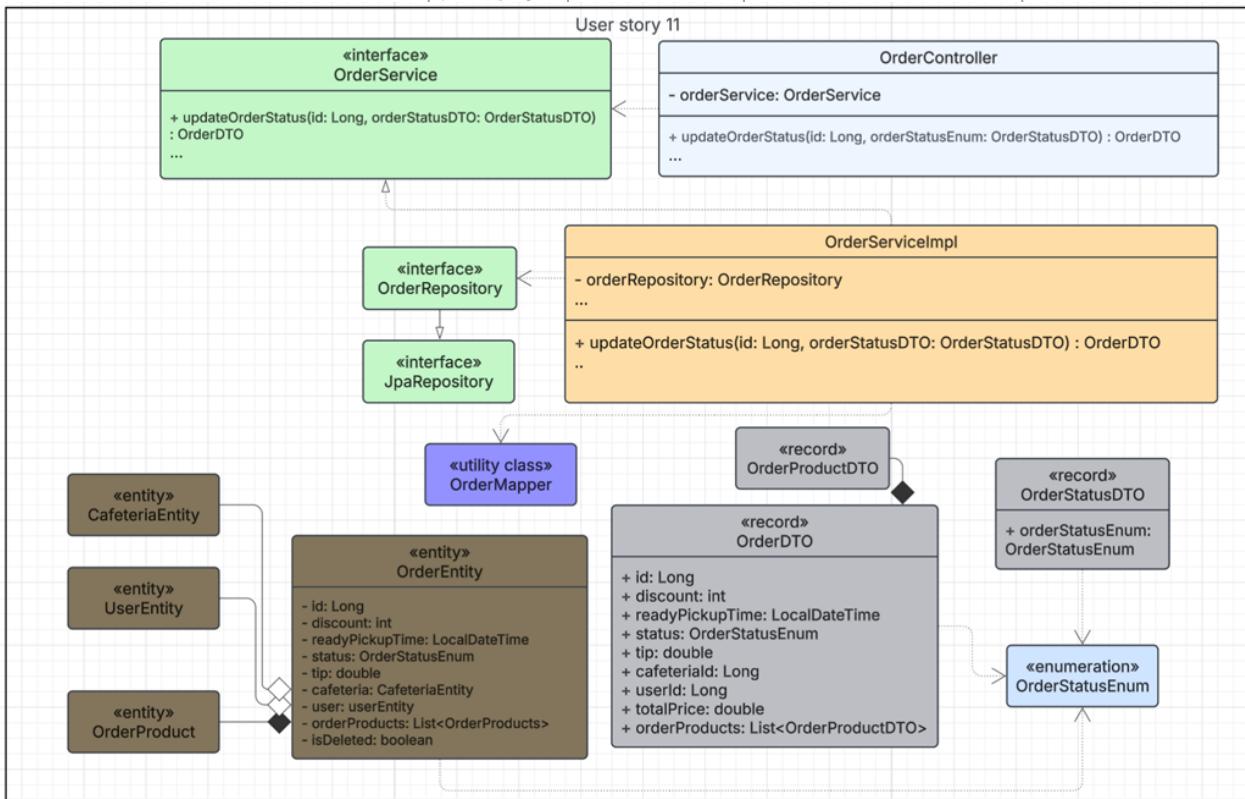
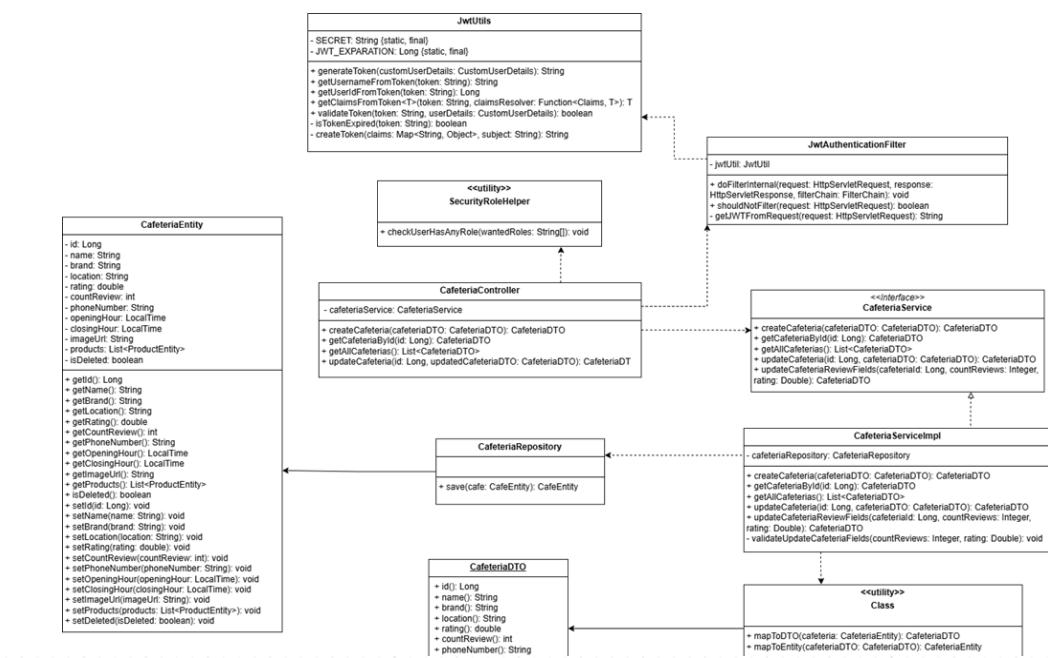


UML class диаграми за определени заявки

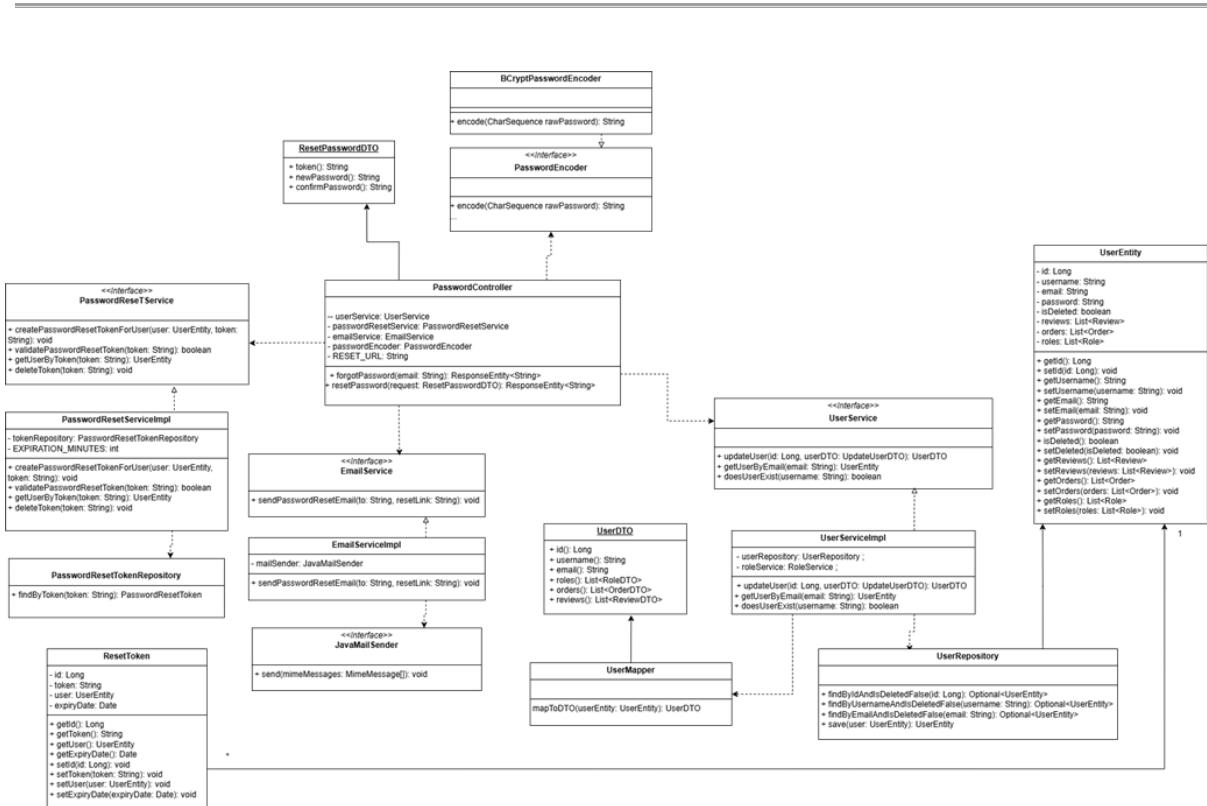


Клас диаграма на заявка за оставяне на ревю.

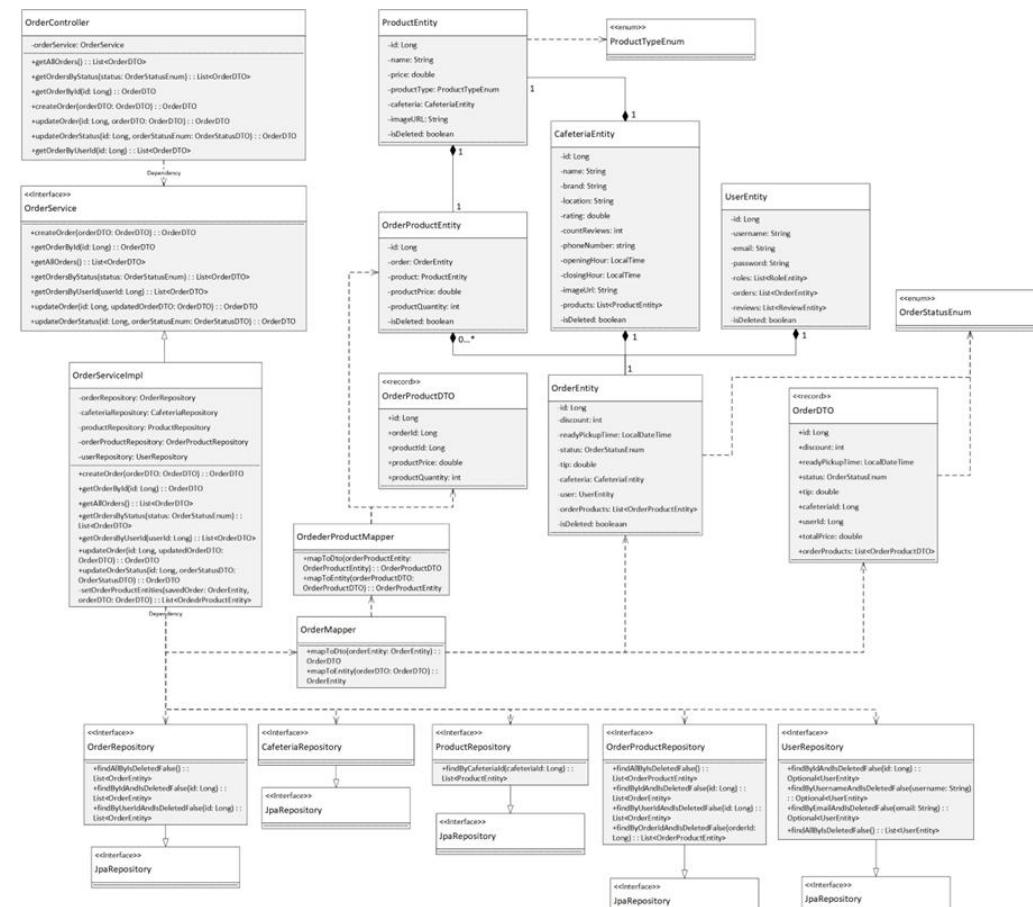
Клас
диаграма на
заявка за
добавяне на
кафене



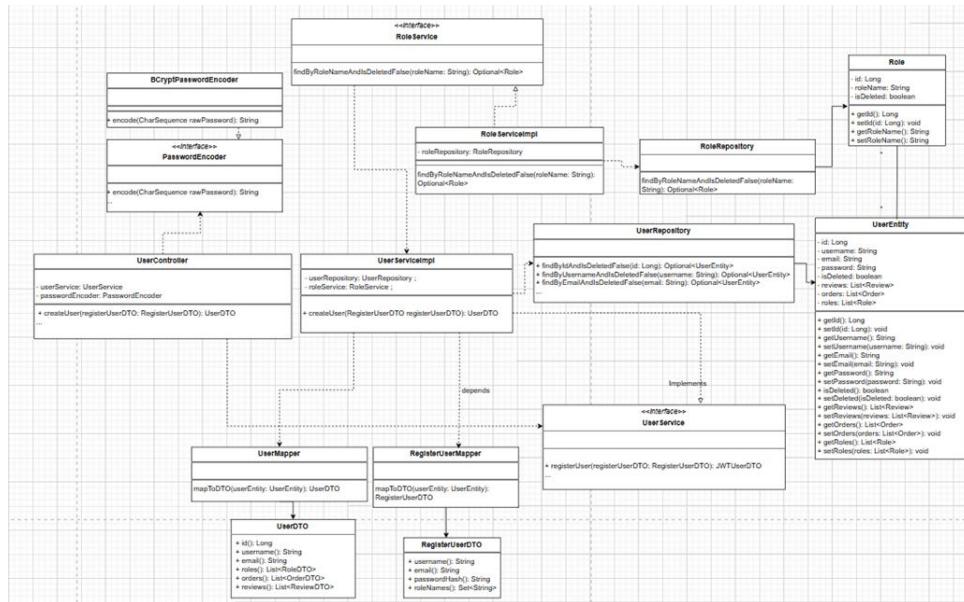
Клас диаграма на заявка за обработване на поръчка от служител



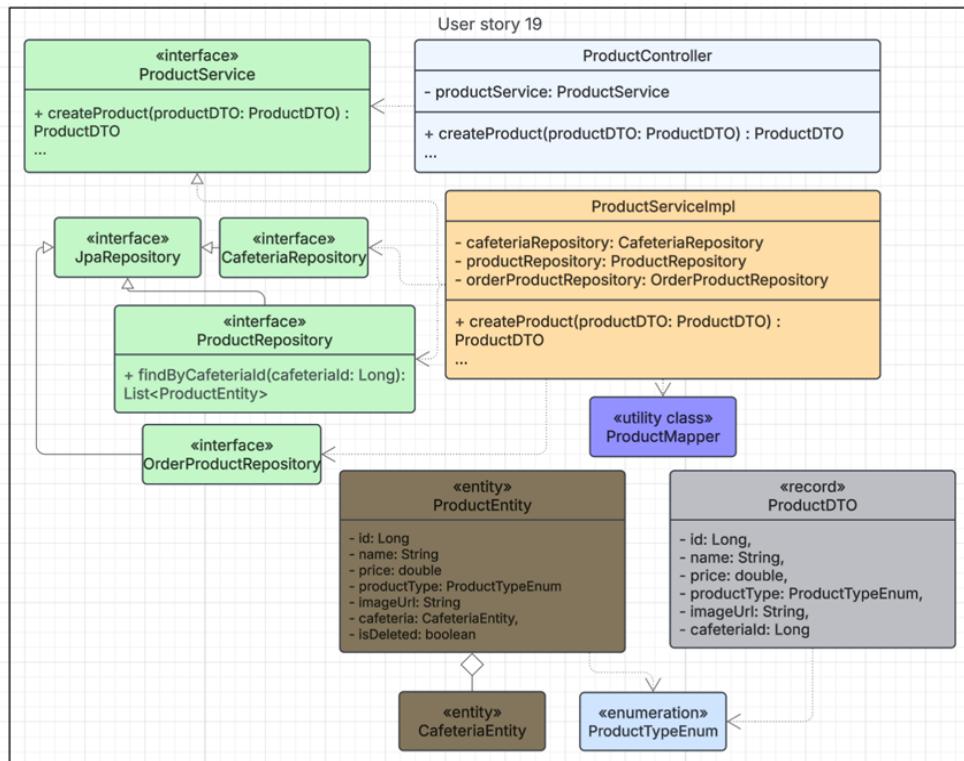
Клас диаграма на заявка за възстановяване на достъп без парола



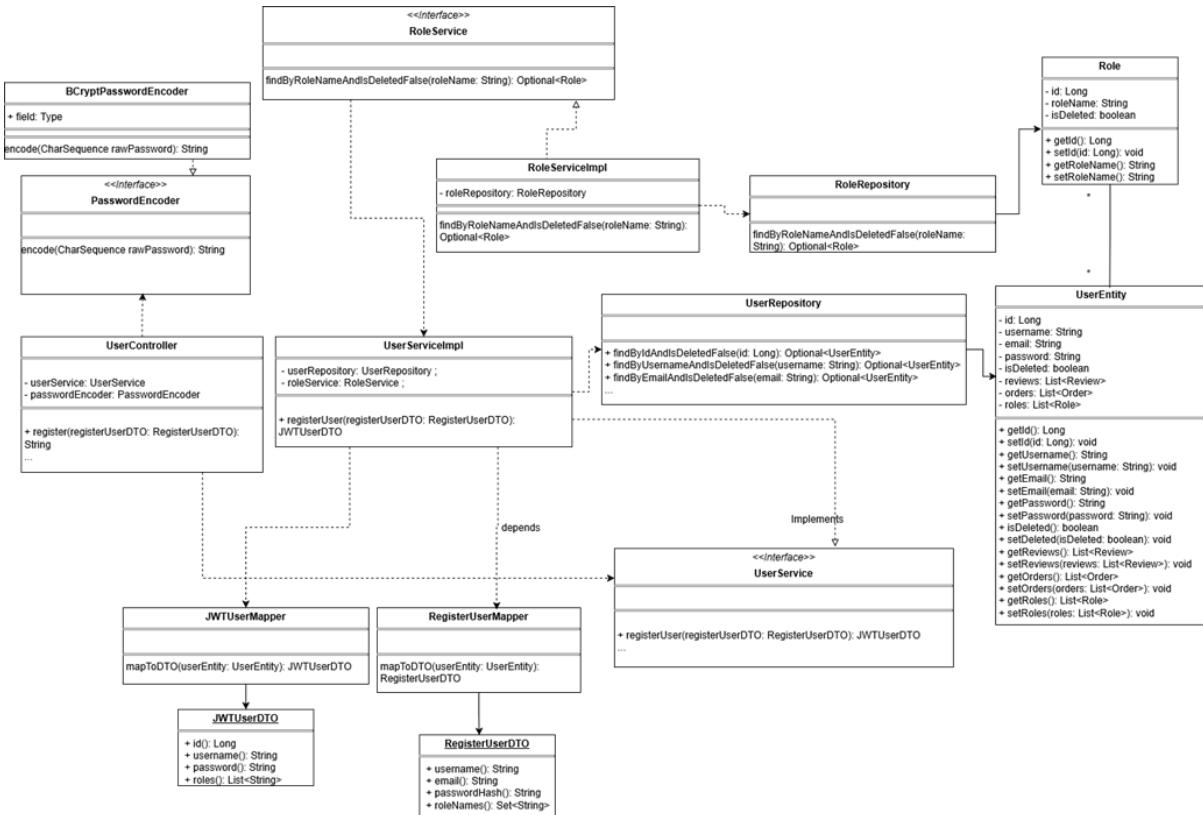
Клас
диаграма
на заявка
за преглед
на минали
поръчки



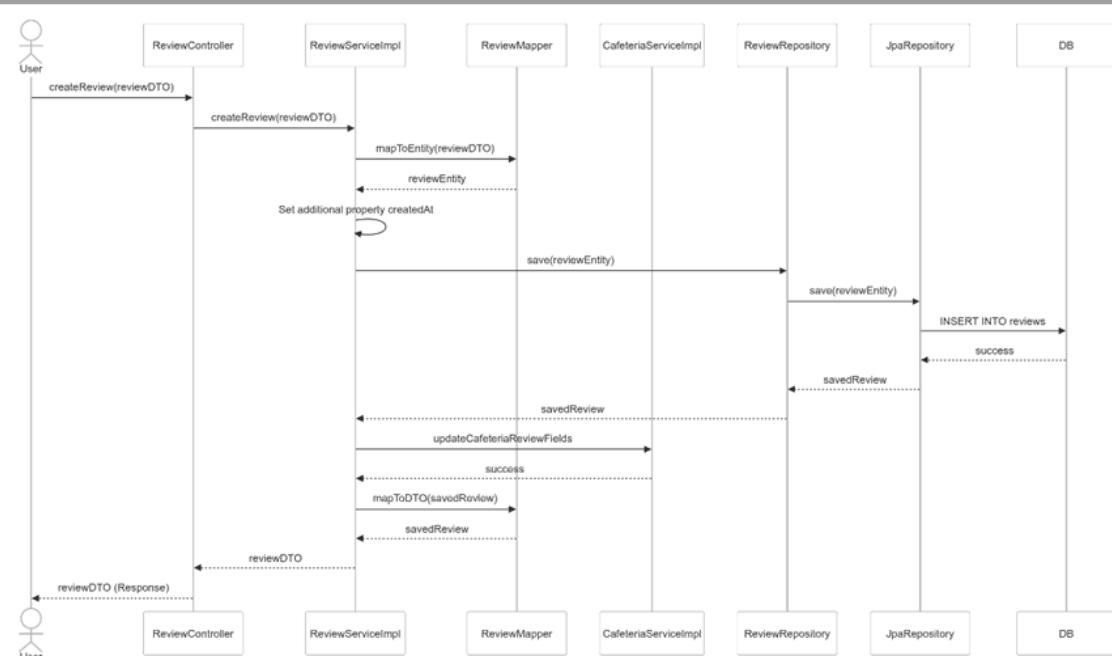
Клас диаграма на заявка за създаване на акаунт със специални роли от администратор



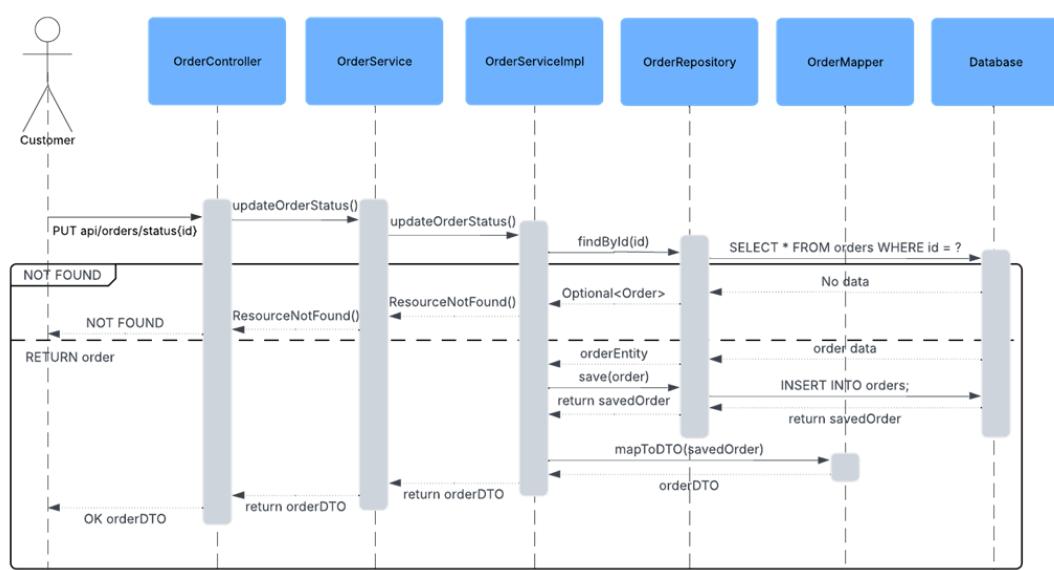
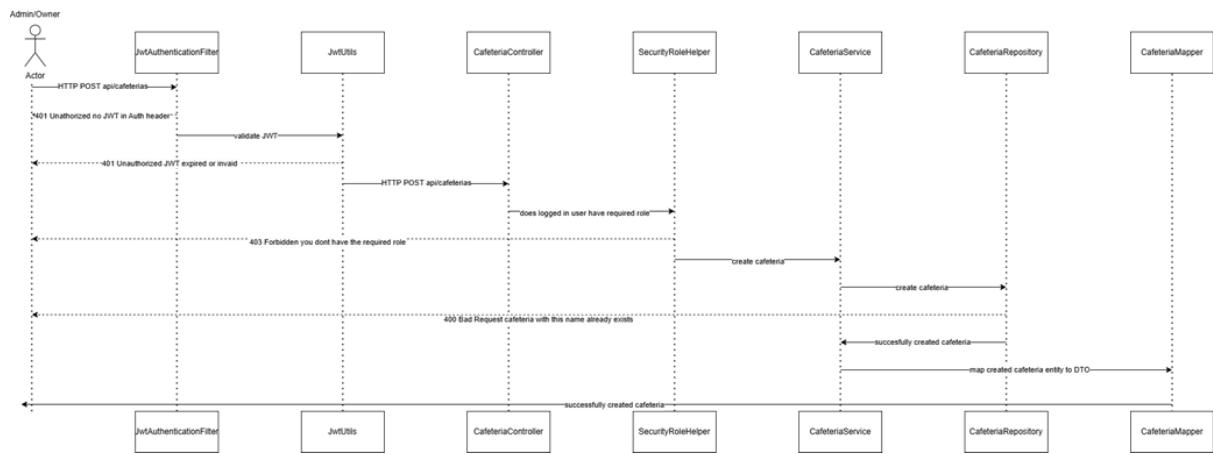
Клас диаграма за добавяне на продукт към кафене

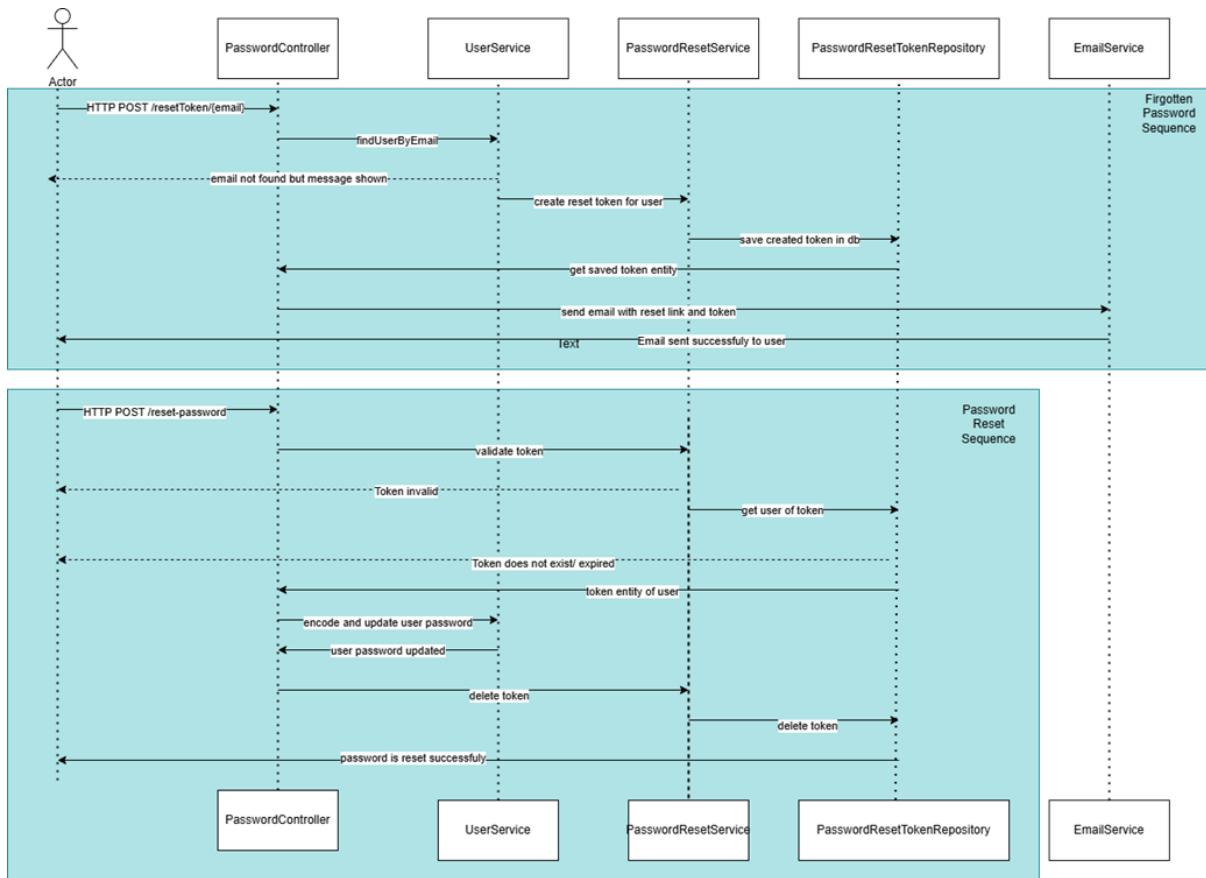


UML sequence diagrams за определени заявки

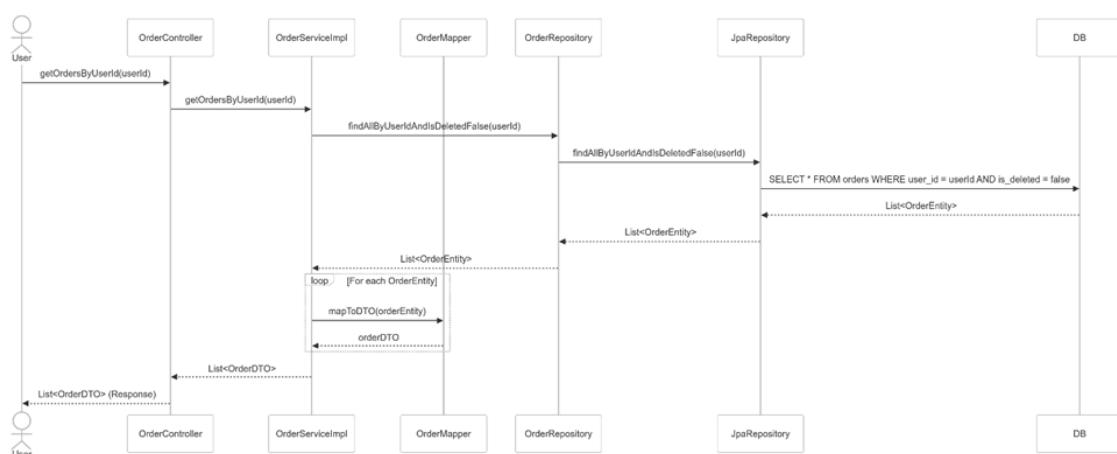


Клас диаграма на заявка за оставяне на ревю.

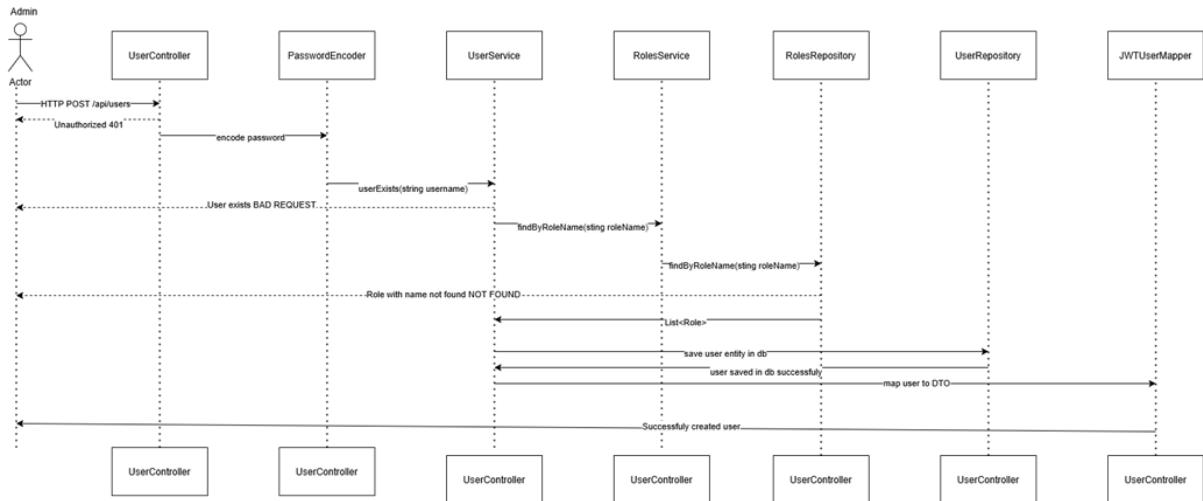




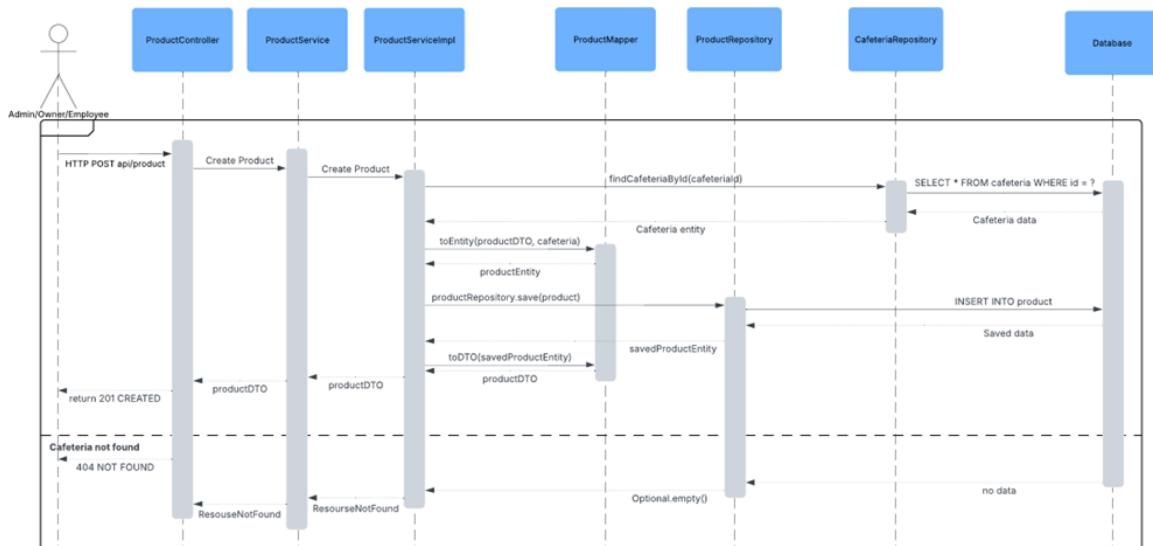
Клас диаграма на заявка за възстановяване на достъп без парола



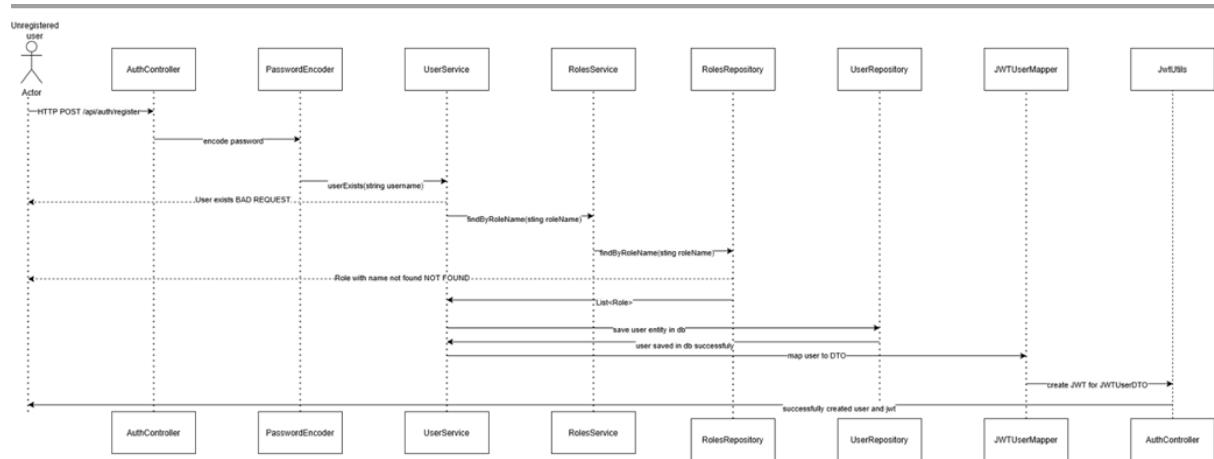
Клас диаграма на заявка за преглед на минали поръчки



Клас диаграма на заявка за създаване на акаунт със специални роли от администратор



Клас диаграма за добавяне на продукт към кафене



Клас диаграма на заявка за създаване на профил от потребител

Спецификация на изискванията:

Функционални изисквания:

№	Изисквания	Приоритет
1	Реализация на потребителски роли - различните роли трябва да имат различен достъп до приложението	1 (Висок)
2	Потребителски профил - създаване, редактиране и изтриване на потребителски профил	1 (Висок)
3	Кафенета - избор на кафене от списък с кафенета с кратка информация	2 (Среден)
4	Директна връзка - директна електронна връзка с кафенето	3 (Нисък)
5	Ревюта - преглед на всички ревюта, както и обща информация за тях (средно аритметично и общ брой)	1 (Висок)
6	Продукти - преглед на всички предлагани продукти, подредени по секции	2 (Среден)
7	Количка - добавяне и премахване на продукти от количка	1 (Висок)
8	Поръчване - реализиране на поръчка	1 (Висок)
9	Минали поръчки - преглед на вече направени поръчки с възможност за следене на статус	2 (Среден)
10	Статус на поръчка - статусът на поръчката трябва да може да се променя на ръка, както и автоматично да се актуализира	2 (Среден)
11	Добавяне на информация - добавяне на кафене, продукти и потребители със специални роли	1 (Висок)
12	Известия - изпращане на известия, съдържащи статуса на поръчката	3 (Нисък)

Нефункционални изисквания:

№	Изисквания	Приоритет
1	Потребителският токен трябва да бъде JWT (JSON Web Token) с валидност до 24 часа.	1 (Висок)
2	Приложението трябва да работи коректно на минимум 95% от устройства с Android 9.0+ и iOS 13+.	1 (Висок)
3	Системата трябва да върне HTTP 403 за всяка заявка без необходимата роля/permission.	1 (Висок)
4	Паролите трябва да бъдат невъзможни за познаване чрез метода на brute force. Всички пароли трябва да се хешират с bcrypt с минимум 10 salt rounds.	1 (Висок)
5	Интерфейсът за служителя трябва да позволява маркиране на поръчка като „готова“ за под 3 клика/действия.	1 (Висок)
6	95% от заявките към публичните API точки трябва да се изпълняват под 500 ms.	1 (Висок)
7	Системата трябва да поддържа 99.5% uptime месечно.	1 (Висок)
8	Системата трябва да поддържа до 1000 едновременни потребителя без забавяне над 20% от нормалното време за отговор.	1 (Висок)
9	Всички POST/PUT заявки трябва да имат валидиране с описание на грешките.	2 (Среден)
10	Минимум 80% покритие на backend кода с unit и integration тестове.	2 (Среден)
11	Всички критични операции (вход, поръчки, грешки) трябва да се логват с ниво на лог INFO и ERROR в централизирана лог система.	2 (Среден)
12	Всички външни API заявки трябва да имат retry механизъм (минимум 3 опита) и timeout от 5 секундия.	2 (Среден)
13	Данните трябва да се архивират автоматично поне веднъж на 24 часа.	2 (Среден)
14	При промяна на статус на поръчката, клиентът трябва да получи push известие в рамките на 5 секунди.	2 (Среден)

Потребителски истории:

№ на user story	Критерии за приемане
1	<ul style="list-style-type: none"> ● При наличие на кафенета на близо: <ul style="list-style-type: none"> ○ На клиента се предоставят всички кафенета, които се намират наблизо, спрямо неговата локация или локацията, която е изbral. ● При липса на кафенета наблизо: <ul style="list-style-type: none"> ○ Клиентът бива информиран от системата, че няма кафенета наблизо.
2	<ul style="list-style-type: none"> ● При наличие на продукти: <ul style="list-style-type: none"> ○ На клиента се предоставя списък от продукти, от който може да избере какво да закупи. ● При липса на продукти: <ul style="list-style-type: none"> ○ Клиентът бива информиран, че няма предлагани продукти от съответното кафене .
3	<ul style="list-style-type: none"> ● При завършена поръчка: <ul style="list-style-type: none"> ○ Информацията за клиента и поръчката бива изпратена към съответното кафене.
4	<p>Клиентът попълва бланката, като описва кой е, задава рейтинг и изказва мнението си.</p> <ul style="list-style-type: none"> ● Добавено ревю: <ul style="list-style-type: none"> ○ При правилна попълнена бланка се създава ново ревю. ○ Актуализира средноаритметичната стойност на рейтинга. ○ Актуализира се броят направени ревюта. ● Грешно попълнена бланка: <ul style="list-style-type: none"> ○ Клиентът е уведомен от системата, ако има налична грешка в някое от полетата. ○ Той не може да създаде ревю, докато грешката не се отстрани.
5	<ul style="list-style-type: none"> ● При избор за плащане онлайн, с разплащателна карта: <ul style="list-style-type: none"> ○ На екрана на клиента се появява форма за плащане, предоставена от външен доставчик, след което при успешно плащане поръчката се изпраща към съответното кафене. ● При избор за плащане в брой: <ul style="list-style-type: none"> ○ Поръчката на клиента е изпратена към съответното кафене.
6	<ul style="list-style-type: none"> ● За промяна на броя продукти: <ul style="list-style-type: none"> ○ Клиентът избира един от двата бутона “+” или “-”, за да редактира количеството на продуктите. ● За цялостно изчистване на количката: <ul style="list-style-type: none"> ○ Клиентът натиска бутона “Clear Cart”, който премахва всички продукти от количката му.

№ на user story	Критерии за приемане
7	<p>При създаване на профил от външен provider:</p> <ul style="list-style-type: none"> • На екрана за регистрация и вход има видими бутони като: „Регистрация с Google“, „Регистрация с Facebook“ и т.н. • При натискане на бутон, клиентът се препраща към избрания provider (OAuth 2.0 flow) и може да се впише успешно • След първоначален вход с външен provider, в системата се създава нов потребителски акаунт с данните от профила (име, имейл и др.) • Ако потребител вече съществува с този имейл, не се създава нов акаунт — връската се установява със съществуващия профил • След успешна автентификация потребителят получава токен както при стандартна регистрация и се счита за вписан. • Ако външният provider върне грешка (напр. отказан достъп), се показва разбираемо съобщение.
8	<ul style="list-style-type: none"> • При отваряне на приложението: На клиента му се предоставя опция за използване на приложението без да се налага да си прави регистрация.
9	<ul style="list-style-type: none"> • При добавяне на кафене: Администраторът отваря формата за добавяне на кафене. <ul style="list-style-type: none"> ◦ При неправилна роля на потребителя: <ul style="list-style-type: none"> ■ Връща се 401 Unauthorized грешка от сървъра ◦ При неправилно попълване на формата: <ul style="list-style-type: none"> ■ Формата не се изпраща към сървъра. ■ Системата уведомява кои полета са попълнени неправилно. ■ Формата не може да бъде изпратена, докато информацията не е в правилния формат. ◦ При правилно попълване на формата: <ul style="list-style-type: none"> ■ Формата се изпраща към сървъра за обработка. ■ Връща се грешка при опит за създаване на кафене с вече съществуващо име ■ Може всеки потребител да види новото кафене
10	<ul style="list-style-type: none"> • При добавяне на служител/собственик: <ul style="list-style-type: none"> ◦ Администраторът/собственикът отваря формата за добавяне на потребител, като той задава потребителско име, имейл и парола. <ul style="list-style-type: none"> ■ При неправилно попълване на формата: <ul style="list-style-type: none"> • Тя не се изпраща към сървъра. ■ При правилно попълване на формата: <ul style="list-style-type: none"> • Тя се изпраща към сървъра.
11	<p>Служителите разполат с отделна страница, в която могат да достъпват и обработват всички поръчки, които все още са в статус: “PROCESSING”.</p> <ul style="list-style-type: none"> • При наличие на необработени поръчки: <ul style="list-style-type: none"> ◦ С кликване на dropdown менюто, те могат да изберат един от изброените статус: <ul style="list-style-type: none"> ■ “DELIVERED”, “COMPLETED”, “POSTPONED”. ◦ При промяна на статуса, конкретната поръчка престава да бъде видима в този прозорец. • При липса на необработени поръчки: <ul style="list-style-type: none"> ◦ Статус код 404. ◦ Служителите биват информирани от системата, че няма поръчки, които да бъдат обработени. ◦ Ако след време, клиент направи поръчка, те ще бъдат уведомени. • При липса на авторизация: <ul style="list-style-type: none"> ◦ Ако потребителят е клиент, той няма права да достъпи обработката на поръчки.

№ на user story	Критерии за приемане
12	<p>Сменянето на паролата става с въвеждане на имейл във формата на екрана за забравена парола.</p> <ul style="list-style-type: none"> ● При натискане на "Изпрати", се валидира дали имейлът е в правилен формат. ● След изпращане на заявката винаги се показва съобщение: "<i>If an account with that email exists, you will receive a password reset link.</i>": ● Ако имейлът е регистриран: <ul style="list-style-type: none"> ○ Сървърът създава токен, свързан с потребителя. ○ Изпраща се имейл на съответната поща с линк съдържащ токена. ○ Токенът има време за валидност (30 минути). ○ Токенът се валидира от сървъра при всяка заявка за промяна на парола. ○ Токенът е уникален, труден за отгатване, и е достатъчно дълъг (UUID). ○ Токенът се изтрива перманентно от базата след успешно променяне на паролата. ● Ако имейлът НЕ е регистриран: Не се създава токен, съответно потребителят не може да промени собствената си парола, докато не въведе правилната поща.
13	<p>Потребителят желае да промени личната си информация в приложението:</p> <ul style="list-style-type: none"> ● При правилно попълнена форма: <ul style="list-style-type: none"> ○ Системата прави PUT Request. ○ Информацията на потребителя се обновява в базата. ● При грешни или липсващи данни: <ul style="list-style-type: none"> ○ Ако не е попълнена формата, системата запазва преходните промени. ○ Потребителят бива информиран за това.
14	<p>Собственикът отваря прозорец, съдържащ информация за поръчките, които са били извършени през приложението, където той може да ги филтрира по различни начини.</p>
15	<p>Клиентът избира кафетерия и попълва контактната форма:</p> <ul style="list-style-type: none"> ● При правилно попълнена контактна форма: <ul style="list-style-type: none"> ○ Системата прави POST Request до имейла на съответната кафетерия с потребителско съобщение. ● При грешни или липсващи данни: <ul style="list-style-type: none"> ○ Ако не е попълнено, от кого е имейла, системата не прави заявка. ○ Кафетериията не получава потребителско съобщение. ○ Потребителят бива инструктиран да попълни полето.
16	<ul style="list-style-type: none"> ● При налични поръчки: <ul style="list-style-type: none"> ○ Клиентът разполага с всички направени поръчки от него, независимо от статуса им. ● При липса на направени поръчки от потребителя: <ul style="list-style-type: none"> ○ Статус код 404 Not Found. ○ Клиентът вижда специален отговор от системата, който го известява, че още не е направил нито една поръчка.
17	<ul style="list-style-type: none"> ● При наличие на ревюта: <ul style="list-style-type: none"> ○ Клиентът разполага с всички направени ревюта за дадено кафене, средноаритметичната стойност на рейтинговата система, както и броя ревюта. ● При липса на налични ревюта: <ul style="list-style-type: none"> ○ Статус код 404 Not Found. ○ Клиентът бива информиран с известие от системата, че няма добавено ревю за конкретната кафетерия. ○ Следователно средноаритметичната стойност и броя ревюта е 0.

№ на user story	Критерии за приемане
18	<p>Администраторът трябва да може да попълни форма, чрез която да създава нови акаунти със определени роли</p> <ul style="list-style-type: none"> ● Администраторът може да създаде акаунт, като избере роля от предварително дефиниран списък. ● Формата трябва да съдържа име, имейл, парола и бутони за селектиране с възможните роли в приложението. ● След създаването на новия профил, при опит за достъп до функционалност извън разрешената роля, създаденият потребител получава съобщение за отказ на достъп (HTTP 403 Forbidden). ● При правилно попълнена форма: <ul style="list-style-type: none"> ○ Трябва да се създаде нов акаунт в базата данни със съответните данни. ● При неправилно попълнена форма: <ul style="list-style-type: none"> ○ Системата сигнализира кои полета не са попълнени правилно. ○ Формата не се изпраща за обработка.
19	<ul style="list-style-type: none"> ● При правилно попълнена форма на продукт: <ul style="list-style-type: none"> ○ Създава се нов продукт в базата. ○ Продуктът е видим в съответната кафетерия. ● При грешни или липсващи данни: <ul style="list-style-type: none"> ○ Самата форма е валидирана. ○ При натискане на “Create” без да е попълнено нито едно поле, ще изпише, че даденото поле не е попълнено.
20	<p>Потребителят иска да създаде собствен профил. Профилът се създава от форма за регистрация, в която има полета име, имейл и парола.</p> <ul style="list-style-type: none"> ● Валидация на данните: <ul style="list-style-type: none"> ○ Паролата е минимум 8 символа. ○ Името е минимум 6 символа. ○ Проверка на наличието на профил със същото име или имейл. ● При регистрация: <ul style="list-style-type: none"> ○ Потребител по подразбиране получава роля ‘клиент’. ● При успешно създаване: <ul style="list-style-type: none"> ○ Потребителят се логва в приложението (вече може да използва всичката налична функционалност в зависимост от ролята му). ● След излизане от профила: <ul style="list-style-type: none"> ○ Потребителят може отново да се логне с информацията за своята регистрация.
21	<ul style="list-style-type: none"> ● Правата на администратора съдържат: <ul style="list-style-type: none"> ○ Детайлна информация за всеки регистриран потребител, независимо от ролята: <ul style="list-style-type: none"> ■ Име, имейл, роли. ■ Бутон за преглед на поръчки/отзиви. ○ Редакция на следните данни: <ul style="list-style-type: none"> ■ имейл, роля, име на потребителя. ○ Изтриване на профили при нарушение. ○ Търсене на потребители по имейл, име или роля. ○ Филтриране по роля (напр. само клиенти). ● Достъп до панела: Само потребители с роля "администратор" имат достъп. ● Логване: Всички действия на администратора се логват за одитни цели.
22	<p>При натискане на бутона със знак “+” в горния ляв ъгъл на всеки продукт, той се добавя към количката.</p>

№ на user story	Критерии за приемане
23	<p>При натискане на бутона “Previous orders” се отваря модал, който съхранява всички поръчки на даден клиент.</p> <ul style="list-style-type: none"> ● При наличие на направени поръчки: Клиентът разполага със следната информация: <ul style="list-style-type: none"> ○ Дата на поръчката. ○ Цена на продукт. ○ Закупени артикули. ○ Броя продукти. ○ Статус на поръчката (например: доставена). ○ Общата сума на поръчката. ● При липса на направени поръчки: Клиентът е информиран, че все още не е направил нито една поръчка.
24	<p>При натискане на бутона “Details” на потребителят му се предоставя цялата информация известна за дадения обект.</p>
25	<p>При натискане на бутона “Contact us here” потребителят попълва форма, която съдържа:</p> <ul style="list-style-type: none"> ● Имейл: Където ще бъде продължена комуникацията. ● Име: С цел потребителят да даде реалното си име за контакт. ● Съобщение: Съобщението, което иска да остави за съответния обект. ● При неправилно попълване на формата: <ul style="list-style-type: none"> ○ Появява се “pop-up”, който гласи, че всички полета трябва да бъдат попълнени. ● При правилно попълване на формата: <ul style="list-style-type: none"> ○ Тя се изпраща към имейл, посочен от обекта за връзка.
26	<p>Правата на администратора съдържат:</p> <ul style="list-style-type: none"> ● Преглед на всички дейности извършени от потребителя включително: <ul style="list-style-type: none"> ○ Информация за неговите поръчки. ○ Информация за неговите ревюта.
27	<p>Администраторът има права да променя данните на потребителите. Аминът може да посещава всички профили на различните потребители през Profile.</p> <ul style="list-style-type: none"> ● При наличие на проблем: <ul style="list-style-type: none"> ○ Админът натиска All users бутона, където се появява списък от всички потребители. При натискане на някой от тях, се отваря съответният профил. ○ Промяната се осъществява от Profile Details.
28	<ul style="list-style-type: none"> ● При прекратяване на взаимоотношенията между приложението и даден обект/бранд: <ul style="list-style-type: none"> ○ Администраторът трябва да може да премахне информацията за дадения обект/бранд от видимото пространство. ● При прекратяване на взаимоотношенията между даден бизнес и служител от съответния бизнес: <ul style="list-style-type: none"> ○ Собственикът на бизнеса или упълномощено от него лице трябва да може да отбележи промените в системата, без да ги изтрива с цел запазване на историята.

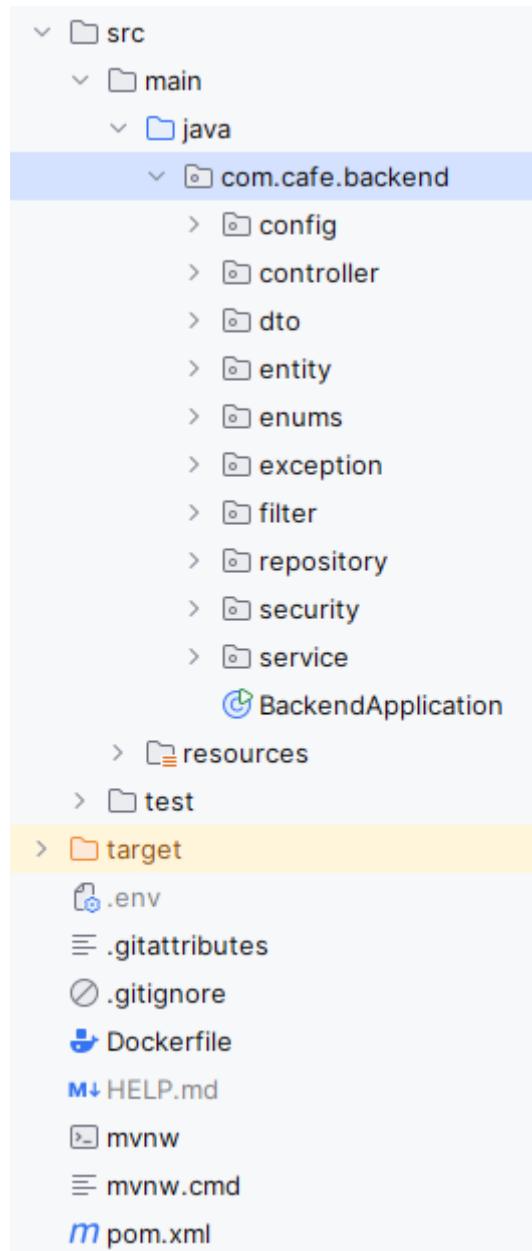
№ на user story	Критерии за приемане
29	<ul style="list-style-type: none"> ● При успешно логване: <ul style="list-style-type: none"> ○ Приложението връща валиден JWT токен в отговора. ○ Всички бъдещи заявки към защитените ресурси се изпращат с токена в Authorization header (Authorization: Bearer <token>). ● Токънът: <ul style="list-style-type: none"> ○ Съдържа информация за потребителя (ID, имейл и роля). ○ Съхранява се в frontend и информацията за токена не се пази на сървъра. ○ Има срок на валидност (например 1 ден), след което е нужно потребителят да се логне отново. ● Невалиден токен или истекъл: <ul style="list-style-type: none"> ○ Потребителят получава отговор с код 401 Unauthorized и е пренасочен към екрана за логин. ● Сървъра валидира: Токена при всяка заявка. ● При logout: Токънът трябва да се изтрива от клиента.
30	<p>При избор на клиента за плащане онлайн, с разплащателна карта и в случай на отказана поръчка от страна на обекта:</p> <ul style="list-style-type: none"> ● Транзакцията се анулира, ако още не е обработена. ● Паричните средства се възстановяват на клиента.

Backend реализация:

Технически спецификации:

- **Език за програмиране:** Java 17
- **Framework:** Spring Boot 3.x
- **ORM:** Hibernate (JPA)
- **База от данни:** PostgreSQL
- **Интеграция с frontend:** RESTful API
- **Security:** JWT Authentication, Spring Security
- **Обработка на заявки:** Spring Web (REST Controllers)
- **Тестове:** JUnit, Mockito
- **Инструменти за build и dependency management:** Maven
- **Външни зависимости:** Lombok, Starter-mail, JWT , Springdoc и др.

Файлова структура на backend:



Разделени са отделните функционалности в отделни packages. Следван е шаблона - MVC. Той е често практикуван и лесен за употреба. MVC е изграден от контролери, сървиси, репозиторита и модели.

Controller:

Съдържа REST контролери, които приемат и обработват HTTP заявки от клиента. Всеки контролер отговаря за конкретна функционалност. Пример за употребата му:

```
@RestController no usages ± Matrix2121 +1
@RequestMapping("/api/reviews")

public class ReviewController {
    @Autowired 4 usages
    private ReviewService reviewService;

    @PostMapping no usages ± Matrix2121
    @ResponseStatus(value = HttpStatus.CREATED)
    public ReviewDTO createReview(@RequestBody ReviewDTO reviewDTO) throws BadRequestException, NotFoundException {
        return reviewService.createReview(reviewDTO);
    }

    @PutMapping("/{id}") no usages ± Matrix2121
    @ResponseStatus(value = HttpStatus.OK)
    public ReviewDTO updateReview(@PathVariable("id") Long id, @RequestBody ReviewDTO updatedReviewDTO)
        throws NotFoundException, BadRequestException {
        return reviewService.updateReview(id, updatedReviewDTO);
    }

    @GetMapping("/{cafeteriaId}") no usages ± Matrix2121
    @ResponseStatus(value = HttpStatus.OK)
    public List<ReviewDTO> getReviewsByCafeteriaId(@PathVariable("cafeteriaId") Long id)
        throws NotFoundException, BadRequestException {
        return reviewService.getReviewsByCafeteriaId(id);
    }

    @GetMapping("/user/{userId}") no usages ± zapryan
    @ResponseStatus(value = HttpStatus.OK)
    public List<ReviewDTO> getReviewsByUserId(@PathVariable Long userId) throws BadRequestException, NotFoundException {
        return reviewService.getReviewsByUserId(userId);
    }
}
```

`@RestController` - Анотацията указва, че това е REST контролер. Spring автоматично сериализира връщаните обекти в JSON.

`@RequestMapping("/api/reviews")` - Това е пътя, по който всяка заявка ще започва.

`@Autowired` - Инжектира Service , който съдържа бизнес логиката.

`@PostMapping` - Методът ще се извика при POST заявка към /api/reviews

`@ResponseStatus(CREATED)` - HTTP статус 201 ще бъде върнат. Съответно в предоставения код има и други статуси.

`@RequestBody` - Очаква JSON тяло.

`@PathVariable("id")` - Взема id, може и да е друго поле не само id.

Другите контролери като стил са идентични на този. Основните разлики са методите, които се извикват от различните services. Друга основна разлика е пътя, по който се достъпват тези end point-ове.

Service:

```
public interface CafeteriaService { 7 usages 1 implementation ± Angel Stoynov +1
    CafeteriaDTO createCafeteria(CafeteriaDTO cafeteriaDTO) throws BadRequestException; 2 usages 1 implementation ± Angel Stoynov
    CafeteriaDTO getCafeteriaById(Long id) throws NotFoundException, BadRequestException; 6 usages 1 implementation ± Angel Stoynov
    List<CafeteriaDTO> getAllCafeterias() throws NotFoundException, BadRequestException; 3 usages 1 implementation ± Angel Stoynov
    CafeteriaDTO updateCafeteria(Long id, CafeteriaDTO cafeteriaDTO) throws NotFoundException, BadRequestException; 3 usages 1 implementation ± Angel Stoynov
    CafeteriaDTO updateCafeteriaReviewFields(Long cafeteriaId, Integer countReviews, Double rating) throws BadRequestException, NotFoundException; 1 usage 1 implementation ± Angel Stoynov
}
```

Използвани са интерфейси за съставянето на services, за да се раздели декларацията от имплементацията на методите. **CafeteriaServiceImpl** съдържа имплементацията на този интерфейс.

@Service - Анотацията, показва, че този компонент в себе си съдържа бизнес логика. Също така предоставя Singleton имплементацията, която съставя само една инициализация на класа.

@Transactional - Осигурява, че всички операции в методите на този клас ще бъдат извършени в една транзакция – ако нещо се провали, се прави rollback.

@Autowired - Изпълнява същата дефиниция, както е описана в controller layer.

Repository - описано в отделна подточка.

```
@Service 2 usages ± Angel Stoynov +3 *
@Transactional
public class CafeteriaServiceImpl implements CafeteriaService {

    @Autowired 5 usages
    private CafeteriaRepository cafeteriaRepository;

    @Override 2 usages ± Angel Stoynov +2
    public CafeteriaDTO createCafeteria(CafeteriaDTO cafeteriaDTO) throws BadRequestException {
        CafeteriaEntity cafeteria = CafeteriaMapper.mapToEntity(cafeteriaDTO);
        cafeteria.setId(null);
        CafeteriaEntity savedCafeteria = cafeteriaRepository.save(cafeteria);
        return CafeteriaMapper.mapToDTO(savedCafeteria);
    }

    @Override 6 usages ± Angel Stoynov +1 *
    public CafeteriaDTO getCafeteriaById(Long id) throws NotFoundException, BadRequestException {
        CafeteriaEntity cafeteria = cafeteriaRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("Could not find cafeteria with this id: " + id));
        return CafeteriaMapper.mapToDTO(cafeteria);
    }

    @Override 3 usages ± Angel Stoynov +1
    public List<CafeteriaDTO> getAllCafeterias() throws NotFoundException, BadRequestException {
        List<CafeteriaEntity> cafeterias = cafeteriaRepository.findAll();

        if (cafeterias.isEmpty()) {
            throw new ResourceNotFoundException("No cafeterias found");
        }

        List<CafeteriaDTO> results = new ArrayList<>();
        for (CafeteriaEntity entity : cafeterias) {
            results.add(CafeteriaMapper.mapToDTO(entity));
        }
        return results;
    }
}
```

```

@Override 3 usages ± Angel Stoynov +1
public CafeteriaDTO updateCafeteria(Long id, CafeteriaDTO cafeteriaDTO)
    throws NotFoundException, BadRequestException {
    CafeteriaEntity cafeteria = cafeteriaRepository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException("Could not find cafeteria with this id:" + id));
    CafeteriaEntity newUpdatedCafeteria = updateCafeteriaFields(cafeteriaDTO, cafeteria);
    return CafeteriaMapper.mapToDTO(newUpdatedCafeteria);
}

@Override 4 usages ± Matrix2121 +1
public CafeteriaDTO updateCafeteriaReviewFields(Long cafeteriaId, Integer countReviews, Double rating) throws BadRequestException, NotFoundException {
    CafeteriaDTO original = getCAFeteriaById(cafeeteriaId);
    validateUpdateCafeteriaFields(countReviews, rating);
    CafeteriaDTO updatedCafeteria = new CafeteriaDTO(
        original.id(),
        original.name(),
        original.brand(),
        original.location(),
        rating,
        countReviews,
        original.phoneNumber(),
        original.openingHour(),
        original.closingHour(),
        original.imageUrl()
    );
    return updateCafeteria(cafeeteriaId, updatedCafeteria);
}

private void validateUpdateCafeteriaFields(Integer countReviews, Double rating) throws ResourceNotFoundException, BadRequestException { 1 usage ± Angel S
    if (countReviews == null || rating == null) {
        throw new ResourceNotFoundException("Rating or count reviews were not found");
    }
    if (countReviews < 0 || rating < 0) {
        throw new BadRequestException("Rating or review cannot be less than 0");
    }
}

```

Това е начинът, по който са имплементирани CRUD методи:

При създаване на нов обект: Мап-ва се входящото request dto, като се превръща в entity. Не се задава id, понеже това се осъществява автоматично от ORM. Извика се repository, което добавя новият обект към таблицата в базата данни. Връща се вече запазения обект.

При търсене по id: Repository-то търси в базата запис отговарящ на посоченото id, ако такъв запис съществува връща съответната кафетерия или хвърля грешка, че не намерен такъв запис. Грешките се обработват в GlobalExceptionHandler. Връща се обратно мапна-тата кафетерия на потребителя.

При търсена на всички обекти от даден тип: Repository-то намира и връща всички обекти от дадена таблица в базата. Прави се проверка дали има изобщо някакви записи в базата, ако няма се хвърля **ResourceNotFoundException**. При наличие на записи се мапват в DTO и се връщат.

При update на обект: Repository-то търси в базата запис отговарящ на посоченото id, ако такъв запис съществува връща съответната кафетерия или хвърля грешка, че не намерен такъв запис. Вика се private метод, който просто сетва новите промени върху обекта, който се обновява. След това се запазват промените в базата и се връща резултата на потребителя.

При изтриване на обект: В cafe-dash се имплементирал само *soft delete*. НЕ се изтрива обекта от базата само се сменя статуса му. Това е идеално в нашия случай, понеже приложението ни е малко по размер и е удобно, ако искаме да възстановим обекта.

Останалите service са сходни като този. Единствените разлики, които могат да бъдат направени са в логиката, която имплементират и в custom извиквания на методи на дадено repository. За Repository-тата ще говорим сега.

Repository:

```
@Repository 4 usages  ± zapryan +2
public interface OrderRepository extends JpaRepository<OrderEntity, Long> {
    List<OrderEntity> findAllByIsDeletedFalse();  2 usages  ± zapryan
    List<OrderEntity> findByIdAndIsDeletedFalse(Long id);  2 usages  ± Matrix2121
    List<OrderEntity> findByUserIdAndIsDeletedFalse(Long id);  1 usage  ± Matrix2121
    List<OrderEntity> findByStatus(OrderStatusEnum status);  1 usage  ± Angel L Stoynov
}

JPA specific extension of org.springframework.data.repository.Repository.
Author: Oliver Gierke, Christoph Strobl, Mark Paluch, Sander Krabbenborg, Jesse Wouters, Greg
Turnquist, Jens Schauder

@NoArgsConstructor
public interface JpaRepository<T, ID> extends ListCrudRepository<T, ID>, ListPagingAndSortingRepository<T, ID>, QueryByExampleExecutor<T> {
```

Всяко наше repository имплементира JPARepository. JPA е вграден интерфейс, който ни осигурява тези методи, които отдолу са описани.

Използва се @Repository анотацията, която казва на Spring, че даден клас или интерфейс е компонент за достъп до базата данни – т.е. DAO слой.

Data-Access-Object(DAO) - дизайн шаблон (design pattern), който съдържа методи като save(), findById(), update(), delete() и отделя персистентната логика от бизнес логиката. Тоест както името му подсказва се използва за достъп на елементи помещаващи се в базата данни.

Имаме наши методи за достъп, които се използват в бизнес логиката. Например findByStatus - очаква се да се върнат всички поръчки, които са в базата с посочен от нас статус.

```
@Override 1 usage  ± Angel L Stoynov
public List<OrderDTO> getOrdersByStatus(OrderStatusEnum status) throws NotFoundException, DataMappingException {
    List<OrderEntity> orders = orderRepository.findByStatus(status);
    if (orders.isEmpty()) {
        throw new ResourceNotFoundException("No orders found with status: " + status);
    }
    List<OrderDTO> orderDTOS = new LinkedList<>();
    for (OrderEntity entity : orders) {
        orderDTOS.add(OrderMapper.mapToDTO(entity));
    }
    return orderDTOS;
}
```

Примерен код, в който е използван дефиниран от нас метод за достъп.

Mapper:

```
public class RoleMapper { 2 usages  ± Angel Stoynov +3 *  
  
    public RoleMapper() { no usages  ± Angel Stoynov *  
        throw new UnsupportedOperationException("Cannot initialize this class " + getClass().getSimpleName());  
    }  
  
    public static RoleDTO mapToDTO(RoleEntity role) throws DataMappingException { ± Angel Stoynov +2  
        if (role == null) {  
            throw new DataMappingException("RoleEntity cannot be null");  
        }  
  
        return new RoleDTO(  
            role.getId(),  
            role.getRoleName()  
        );  
    }  
  
    public static RoleEntity mapToEntity(RoleDTO roleDTO) throws DataMappingException { ± zapryan +2  
        if (roleDTO == null) {  
            throw new DataMappingException("RoleDTO cannot be null");  
        }  
  
        return RoleEntity.builder()  
            .id(roleDTO.id())  
            .roleName(roleDTO.roleName())  
            .build();  
    }  
}
```

Мапърите се използват най-вече, при наличие на различни слоеве. Например, за да преобразуване от Entity в DTO или обратното. [Повече за това в DTO/Entity подточката.](#)

Това са наши custom мапери. Не е използвано например: **map-struct**, който автоматично генерира мапърите.

Понеже това е utils клас, тоест в себе си съдържа само static методи не трябва да може да се инстанцира този клас. За това към конструктора е добавена UnsupportedOperationException, който показва, че класа е “static”.

В метода mapToDTO се проверява дали role не е null. Ако не, се хвърля custom грешка DataMappingException, тя се обработва глобално.

За създаване на нов обект по този начин се използва @Builder анотацията в самото DTO - [повече за това в DTO/Entity подточката.](#) Същото нещо се случва и в mapToEntity, където процесът е същия само че се връща entity.

Останалите мапъри са базирани също на този принцип.

DTO/Entity:

DTO:

```
@Builder 52 usages ± Angel Stoynov +1
public record ProductDTO(
    Long id, 1 usage
    String name, 8 usages
    double price, 2 usages
    ProductTypeEnum productType, 2 usages
    String imageUrl, 1 usage
    Long cafeteriaId 3 usages
) {}
```

@Builder - Анотацията ни позволява да създаваме обекти по този начин:

```
return CafeteriaEntity.builder()
    .name("cafe")
    .brand("star")
    .location("pz")
    .rating(2)
    .openingHour(LocalTime.parse(text: "06:00"))
    .closingHour(LocalTime.parse(text: "18:00"))
    .isDeleted(false)
    .countReview(10)
    .phoneNumber("+359890553312")
    .imageUrl("url")
    .build();
```

В този пример е използвано Entity, но в DTO е същия синтаксис.

Да се върнем на DTO.

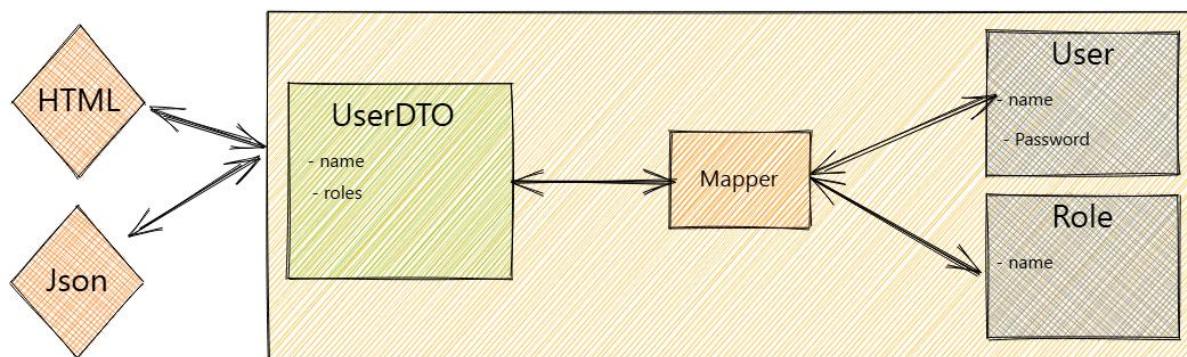
@Builder 52 usages · Angel Stoynov +1

```
public record ProductDTO(
    Long id, 1 usage
    String name, 8 usages
    double price, 2 usages
    ProductTypeEnum productType, 2 usages
    String imageUrl, 1 usage
    Long cafeteriaId 3 usages
) {}
```

Използвано е record, вместо клас, защото record е immutable, няма нужда от конструкути и от getter/setter.

DTO - Data transfer object, както името подсказва DTO се използва, за да пренася данни между слоевете. В него можем да ограничава какво искаме да изнесем и какво да бъде достъпно на потребителя.

Presentation Layers



Entity:

```
@Entity 30 usages  ↳ Angel Stoynov +1
@Table(name = "role")
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor|
```

```
public class RoleEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Column(name = "role_name", length = 100, nullable = false)
    private String roleName;

    @Column(name = "is_deleted")
    private boolean isDeleted;
}
```

@Entity - Анотацията показва, че класа е Entity, автоматично го прави таблица в базата данни, управлявана чрез JPA/Hibernate.

@Table - Името на таблицата

@Data - Генерира автоматично гетъри, сетъри, toString(), equals(), hashCode() за всички полета.

@Builder - вече говорихме за него.

@AllArgsConstructor/NoArgsConstructor както имената подсказват създават празен и конструктор с всички полета.

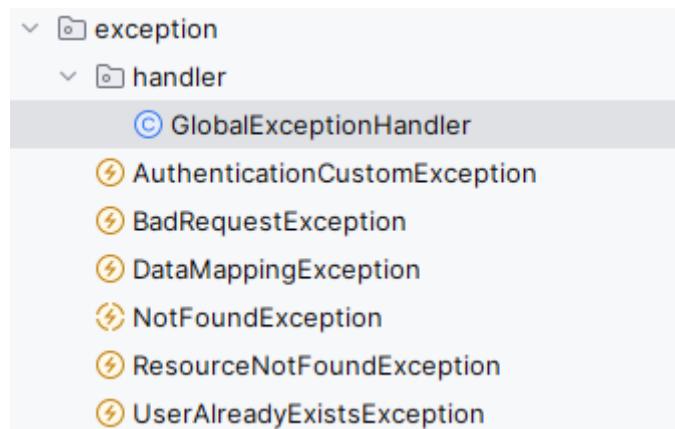
@Id - Маркира id като първичен ключ на таблицата.

@GeneratedValue(strategy = GenerationType.IDENTITY) - id-то ще бъде автоматично генерирано от базата – чрез auto-increment.

@Column(name = "role_name", length = 100, nullable = false) - показва как ще се казва колоната, дължината и че не може да е null.

Използвани са Entity класове, за да се опише структурата на базата от данни чрез Java код. Всеки @Entity клас представлява таблица в базата, а всяко негово поле – колона. Останалите entity са на същия принцип.

Exceptions:



GlobalExceptionHandler е отговорен за обработката на всички грешки. Изброените са нашите custom exceptions.

```
@RestControllerAdvice no usages Angel Stoynov
public class GlobalExceptionHandler {

    @ExceptionHandler(BadRequestException.class) no usages Angel Stoynov
    @ResponseStatus(value = HttpStatus.BAD_REQUEST)
    public String handleBadRequestException(BadRequestException ex) { return ex.getMessage(); }

    @ExceptionHandler(NotFoundException.class) no usages Angel Stoynov
    @ResponseStatus(value = HttpStatus.NOT_FOUND)
    public String handleNotFoundException(NotFoundException ex) { return ex.getMessage(); }

    @ExceptionHandler(AuthenticationCustomException.class) no usages Angel Stoynov
    @ResponseStatus(value = HttpStatus.FORBIDDEN)
    public String handleNotFoundException(AuthenticationCustomException ex) { return ex.getMessage(); }

    @ExceptionHandler(ConcurrentModificationException.class) no usages Angel Stoynov
    @ResponseStatus(value = HttpStatus.INTERNAL_SERVER_ERROR)
    public String handleConcurrentModificationException(ConcurrentModificationException ex) { return ex.getMessage(); }

    @ExceptionHandler(Exception.class) no usages Angel Stoynov
    @ResponseStatus(value = HttpStatus.INTERNAL_SERVER_ERROR)
    public String handleGlobalException(Exception ex) { return "An unexpected error occurred: " + ex.getMessage(); }
}
```

@RestControllerAdvice - Spring анотация, catch-ва всички грешки възникнали в RestController.

@ExceptionHandler() - Указва, че този метод обработва грешки от даден тип.

@ResponseStatus() - Задава какъв статус код искаме да върнем.

Съответно тук са описани всяка custom грешка какво да върне като статус код. Ако някоя грешка не е уловена накрая се хващат всички грешки, като се връща статус код 500 - INTERNAL SERVER ERROR.

Security:

Реализацията на сигурността в приложението е силно базирана на технологията **JWT (JSON Web Token)** – стандартен механизъм за stateless за удостоверяване и управление на достъпа.

JWT е токен, който съдържа **кодирана информация за потребителя**, като например неговия **уникален идентификатор (ID)** и ролите, които притежава. Когато потребителят се впише успешно в системата, **сървърът му издава JWT**, който се съхранява **локално** (в паметта на устройството или браузъра) и се изпраща автоматично **с всяка следваща заявка**, чрез HTTP Authorization header.

- **Authorization: Bearer <JWT-токен>** - изпращане на токен с HTTP заявка

Сървърът валидира токена при всяка заявка, като извършва следните проверки:

1. **Проверка на подписа (signature):**

Сървърът използва **тайния ключ**, с който е подписан токена, за да сравни подписа и да установи дали токенът е бил подправен.
Ако подписът не съвпада – токенът е невалиден.

2. **Проверка на валидността на съдържанието:**

JWT съдържа допълнителни параметри (claims), чрез които сървърът установява дали токенът е все още валиден:

- exp (expiration) – време, до което токенът е валиден.
- iat (issued at) – време, в което токенът е бил създаден

Ако **всички условия са изпълнени**, токенът се счита за **валиден** и достъпът се разрешава. В противен случай, сървърът връща **HTTP статус 401 Unauthorized**, което означава, че потребителят трябва да се **аутентицира отново**.

JWT позволява изграждане на **сигурна и скалируема система**, без необходимост от поддържане на сървърна сесия за всеки потребител, което го прави предпочтено решение за съвременни мобилни и уеб приложения.

Избран backend framework за сигурност Spring Security

В приложението Cafe-Dash **Spring Security** е използван заедно с **JWT (JSON Web Token)**, за да осигури **сигурна и stateless** автентикация и контрол на достъпа.

1. Удостоверяване (Authentication)

- Когато потребителят се впише в приложението (чрез login заявка), Spring Security валидира въведените **потребителско име и парола** чрез имплементация на UserDetailsService.
- Ако данните са валидни, се създава **JWT токен**, който съдържа информация за потребителя и се подписва с **таен ключ**.
- Този токен се връща на клиента и **се съхранява локално**

2. Authorization - След като потребителят бъде успешно удостоверен и предоставеният JWT токен бъде валидиран, **Spring Security използва класа UsernamePasswordAuthenticationToken**, за да създаде обект, представляващ текущо

автентицирания потребител. Този обект съдържа информация като **потребителско име, ID, роли и права за достъп**.

Създаденият UsernamePasswordAuthenticationToken се **поставя в SecurityContextHolder** – централен, статичен контейнер, който съхранява информация за сигурността по време на текущата заявка. **SecurityContextHolder е валиден само за продължителността на една HTTP заявка** – след нейното приключване, контекстът се изчиства автоматично.

При всяка следваща заявка с валиден JWT токен, процесът се повтаря – токенът се валидира, информацията за потребителя се извлича и се създава нов Authentication обект, който се записва отново в SecurityContextHolder.

Този механизъм позволява на Spring Security да провери дали текущият потребител има необходимите роли за достъп до даден метод или ресурс

Ако ролята или правата на потребителя **не съответстват на изискванията**, Spring Security автоматично прекратява изпълнението на заявката и връща **грешка с HTTP статус код 403 Forbidden**, като по този начин блокира достъпа до защитения ресурс.

Този подход осигурява централизирана, гъвкава и сигурна проверка на достъпа, базирана на реалната самоличност и роля на потребителя във всяка индивидуална заявка

Имплементация на security на сървъра

За да се интегрира тази технология в приложението са използвани следните компоненти:

- **JwtAuthenticationFilter** - филтър, който валидира и сетва данните в **SecurityContextHolder**
 - **AuthController** - контролер, който участва в регистрирането/логването
 - **JwtUtils** - използва пакета io.jsonwebtoken и обработва всички операции, свързани с JWT, като генериране на токени, валидиране, извлечане на claims и получаване на потребителска информация.
 - **SecurityConfig** - конфигурация за Spring Security
 - **SecurityRoleHelper** - статичен клас, който проверява дали в **SecurityContextHolder** съществува дадена роля
 - **CustomUserDetailsService** - имплементация на **UserDetailsService**(интерфейс нужен на Spring Security), която намира потребителят по неговото име
-
- **JwtAuthenticationFilter** - специализиран филтър от Spring Security, който се изпълнява веднъж за всяка входяща HTTP заявка и отговаря за JWT-базирана автентикация в приложението
- Основна цел:**
- Да извлече JWT токена от заглавката Authorization.

- Провери дали токенът е валиден и дали съдържа валидна информация за потребителя.
Зареди потребителските данни от базата чрез `UserDetailsService`.
- Да настрои контекста на сигурност (`SecurityContext`) със съответните роли и права, ако токенът е валиден.

Основни методи:

doFilterInternal(...) - Този метод се изпълнява автоматично за всяка заявка:

- Извлича токена от HTTP заглавката `Authorization`.
- Проверява дали токенът е валиден:
- Извлича потребителското име и ID.
- Зарежда потребителя чрез `CustomUserDetailsService`.
- Валидира токена чрез `JwtUtil.validateToken(...)`.

Ако всичко е наред, създава обект `UsernamePasswordAuthenticationToken`, който представлява автентицирания потребител.

Настройва `SecurityContextHolder` с новата автентикация — това позволява по-късно да се правят проверки за роли и права по време на изпълнение на заявката. При липса на токен или при невалиден токен връща подходящ HTTP статус код (403 или 401).

shouldNotFilter(HttpServletRequest request) - Определя кои пътища трябва да бъдат пропуснати от този филтър (напр. публични маршрути):

- `/api/auth/**` – маршрути за вход/регистрация
- `/api/password/**` – възстановяване на парола
- `/swagger-ui/**, /v3/api-docs/**` – документация
- HTTP метод `OPTIONS` – CORS заявки

getJWTFromRequest(HttpServletRequest request) Извлича JWT токена от HTTP заглавката `Authorization`

- **AuthController** - REST контролер, отговорен за управлението на потребителската **автентикация** (вход) и **регистрация**. Той е достъпен под пътя `/api/auth` и предоставя два основни публични крайни точки: `/login` и `/register`.
- **Основна роля**
 - Валидиране на потребителски идентификационни данни (потребителско име и парола),
 - Регистриране на нови потребители,
 - Генериране на **JWT токен**, който се използва за удостоверяване в последващи заявки.

Контролерът е част от реализацията на **stateless security**, при която сървърът не пази състояние на сесията, а използва валидиран токен за идентификация на всеки потребител при всяка заявка.

- **Основни зависимости:**

- **JwtUtil** – Утилити клас за създаване и валидиране на JWT токени.

- **UserService** – Сървис слой, който управлява операциите, свързани с потребители (регистрация, проверка за съществуване и др.).
- **AuthenticationManager** – Интерфейс на Spring Security, който валидира входните данни. Тук се използва имплементацията на интерфейса UserDetailsService
- **PasswordEncoder** – Отговаря за безопасното хеширане на потребителски пароли преди запис в базата. Конкретният обект се създава в SecurityConfig в се инжектира в този интерфейс.
- **JwtUtil** - компонент от слоя за сигурност, отговорен за всички операции, свързани с **JWT** – създаване, валидиране, парсване на claims и извлечане на потребителска информация от токена.

Той играе **централна роля** в осигуряването на **stateless authentication** в приложението, като позволява на сървъра да аутентицира потребителите **без необходимост от сесии**.

Основни функционалности:

generateToken(CustomUserDetails customUserDetails)

- Създава подписан JWT токен, съдържащ:
 - ID на потребителя
 - роля/роли
 - потребителско име
- Използва HMAC SHA-512 (HS512) за подписване.
- Валидността на токена е 24 часа (86400000 милисекунди).

validateToken(String token, CustomUserDetails userDetails)

- Проверява дали:
 - потребителското име от токена съвпада с това на подадения CustomUserDetails
 - ID-то от токена съвпада
 - токенът не е истекъл
 - потребителят не е заключен и е активен
- Връща true, ако токенът е валиден и потребителят има права за достъп.

getUsernameFromToken(String token)

- Връща потребителското име (subject) от токена.

getUserIDFromToken(String token)

- Връща ID-то на потребителя, закодирано в токена.

getClaimsFromToken(String token, Function<Claims, T> claimsResolver)

- Извлича желаната информация (claim) от токена чрез функция.
- Пример: използва се за извлечане на subject, expiration date, ID и други.

isTokenExpired(String token)

- Проверява дали токенът е изтекъл по стойността на exp (expiration).
- Използва текущата дата за сравнение.
- Хваща и ExpiredJwtException при нужда.

createToken(Map<String, Object> claims, String subject)

- Създава нов JWT токен с подадените claims и subject (обикновено потребителско име).
- Добавя текуща дата за issuedAt и автоматично изчислява expiration.

Вътрешна логика:

- Токените се подписват с предварително зададен таен ключ (SECRET) и са защитени от промяна.
- При всяка заявка токенът се проверява чрез валидиране на подписа и съдържанието.
- Всички claims се записват в полето payload на JWT и са кодирани, но **не криптираны**, затова не трябва да съдържат чувствителни данни (като пароли).

• SecurityConfig

SecurityConfig е конфигурационен клас, който настройва **Spring Security** за приложението Cafe-Dash. Той дефинира основните правила за сигурност, обработката на заявки, използването на JWT токени и криптирането на пароли.

Основни отговорности:

- Конфигурира stateless сигурност чрез JWT (JSON Web Token).
- Осигурява криптиране на пароли чрез BCryptPasswordEncoder.
- Позволява публичен достъп до определени публични маршрути (например /api/auth/**, /swagger-ui/** и др.).
- Изисква аутентикация за всички останали заявки.
- Деактивира сесии и CSRF защита (тъй като се използва JWT, а не cookies).
- Добавя JwtAuthenticationFilter **преди** UsernamePasswordAuthenticationFilter в security filter chain-a, за да прихваща и валидира токени още в началото на заявката.

Основни компоненти:

- **@EnableWebSecurity** - Активация на Web-базираната сигурност в Spring.
- **JwtAuthenticationFilter** - филтър за вземане на JWT, декриптиране и променяне на **SecurityContextHolder** със аутентикирания потребител при всяка защитена заявка
- **PasswordEncoder passwordEncoder()** - Създава BCryptPasswordEncoder, който:
 - Хешира пароли сигурно.
 - Използва salt автоматично за всяка парола.
 - Предпазва от атаки чрез речници и brute force.
- **AuthenticationManager authenticationManager(...)** - Конфигурира и предоставя AuthenticationManager, който се използва при логин, за да валидира потребителското име и парола. При всяко използване на метода authenticate(Authentication authentication) автоматично се използва конкретният клас на **PasswordEncoder** конфигуриран в SecurityConfig.
- **SecurityFilterChain filterChain(HttpSecurity http)** - Това е сърцето на сигурността в приложението. Тук се дефинират:
 - **CSRF**: изключена, защото не се използват сесии.
 - **Сесии**: STATELESS – няма сесии
 - **Правила за достъп**:
 - Публичен достъп: /api/auth/**, /swagger-ui/**, /v3/api-docs/**, и др.
 - Защитени маршрути: всички останали заявки изискват JWT.
 - Добавяне на JwtAuthenticationFilter преди Spring-овия UsernamePasswordAuthenticationFilter, за да може токенът да бъде обработен преди останалата част от security логиката

Swagger

Инструмент за автоматична генерация на документация за REST API. В нашето приложение е използван, за да:

- предоставя **интерактивна документация**, в която могат да се тестват API заявки директно от браузъра;
- улесни **разработчиците и тестерите** при разбирането и използването на сървърните ендпоинти;
- гарантира, че документацията винаги е в **синхрон с реалната имплементация**, тъй като се генерира автоматично от анотации в кода.

The screenshot shows the 'Cafe-Dash' API documentation. At the top, it says '1.0.0' and 'CAS 3.0'. Below that is a 'Servers' dropdown set to 'http://localhost:8080 - Generated server url' and an 'Authorize' button. The main area is titled 'user-controller' and contains four API endpoints:

- GET /api/users/{id}
- PUT /api/users/{id} (highlighted in orange)
- GET /api/users
- POST /api/users

Below this is another section titled 'review-controller'.

Изглед на Swagger

The screenshot shows a 401 error response. The status is '401' and the message is 'Error: response status is 401'. It's labeled 'Undocumented'. The 'Response body' section contains the text: 'No token found in the request. You must login first'.

Грешка при липса на JWT в Swagger

The screenshot shows a 403 error response. The status is '403' and the message is 'Error: response status is 403'. The 'Response body' section contains the text: 'User does not have any of the required roles: [admin]'.

Грешка при липсваща роля

Използван е **Springdoc OpenAPI**, който интегрира Swagger със Spring Boot. Swagger е единственото място, което е рисково за security(frontend частта не позволява на потребителя да изпълнява забранени за него действия). Swagger трябва да е строго защитен, защото е лесно достъпен и заявките са лесни за правене

- **Използване на Swagger и Spring Security съвместно**

Swagger ползва “/swagger-ui” и “/v3/api-docs”, но тези пътища трябва да бъдат конфигурирани да не се филтрират от **JwtAuthenticationFilter**, точно за това в метод shouldNotFilter тези конкретни пътища са конфигурирани да бъдат пропускани от филтъра

- **SwaggerConfig** - Тази конфигурация задава как работи Swagger (OpenAPI) в приложението. Основната ѝ цел е да направи документацията на REST API-то достъпна и разбираема, като визуализира всички налични ендпоинти, тяхната структура и параметри директно в уеб интерфейс.
 - Конфигурацията включва всички пътища на приложението, за да могат да се тестват през Swagger UI. Освен това, тя добавя поддръжка за JWT удостоверяване чрез „Bearer“ токени. Това означава, че ако един ендпоинт изисква авторизация, Swagger ще показва бутон „Authorize“, чрез който потребителят може да въведе своя токен.
 - Добавянето на авторизация е важно, защото много от функционалностите в приложението (като създаване на поръчка, достъп до административни панели и др.) са защитени и изискват потребителят да бъде вписан. Без авторизация, тези ендпоинти не могат да бъдат достъпвани или тествани през Swagger. Затова конфигурацията позволява валиден токен да се въведе в интерфейса, след което всички заявки, изпращани от Swagger, ще включват този токен автоматично. Това улеснява разработчиците и тестерите при проверка на защитени функционалности.



Available authorizations

bearerAuth (http, Bearer)

Value:

[Authorize](#)

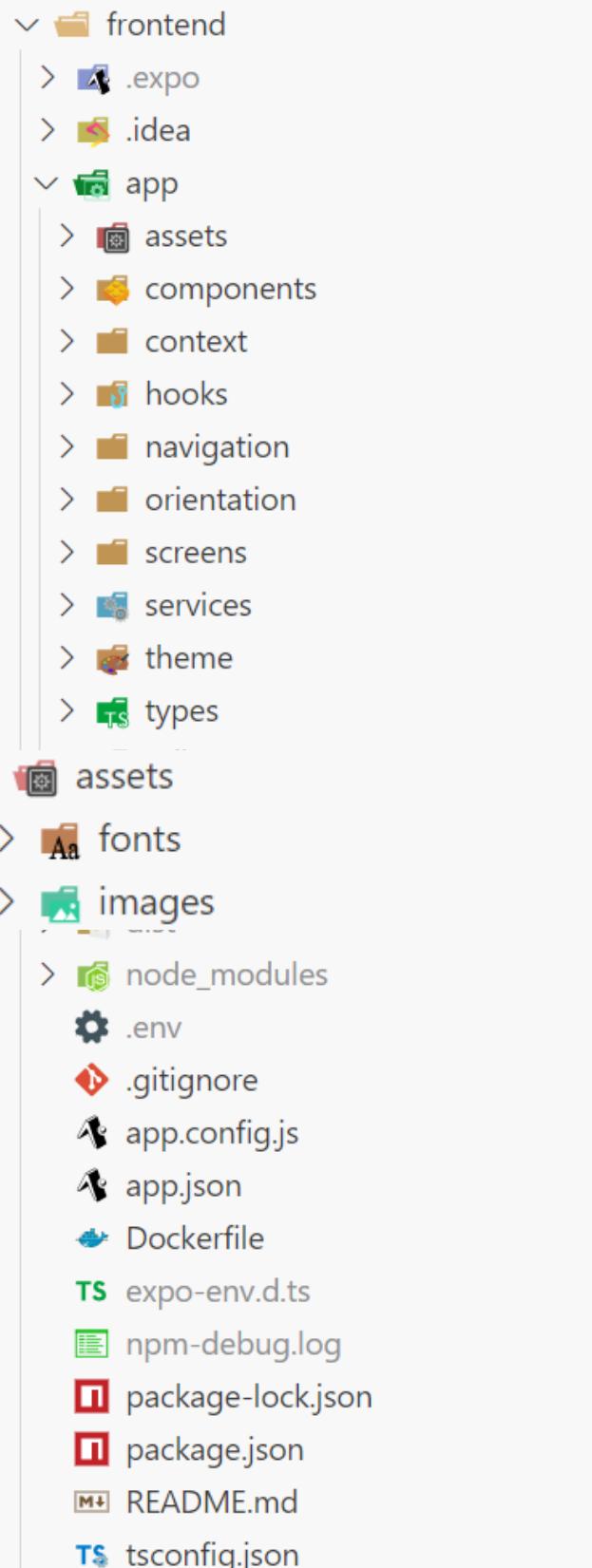
[Close](#)

[Повече информацията за структурата на backend, както и пълно описание на програмния код може да намерите в javadoc документацията. Тя се намира в репозиторио.](#)

Frontend реализация:

Файлова структура:

Архитектурния стил, който е използван е модерна модулна архитектура по функционалности, където всяка папка отговаря за дадена функционалност и има своя роля.

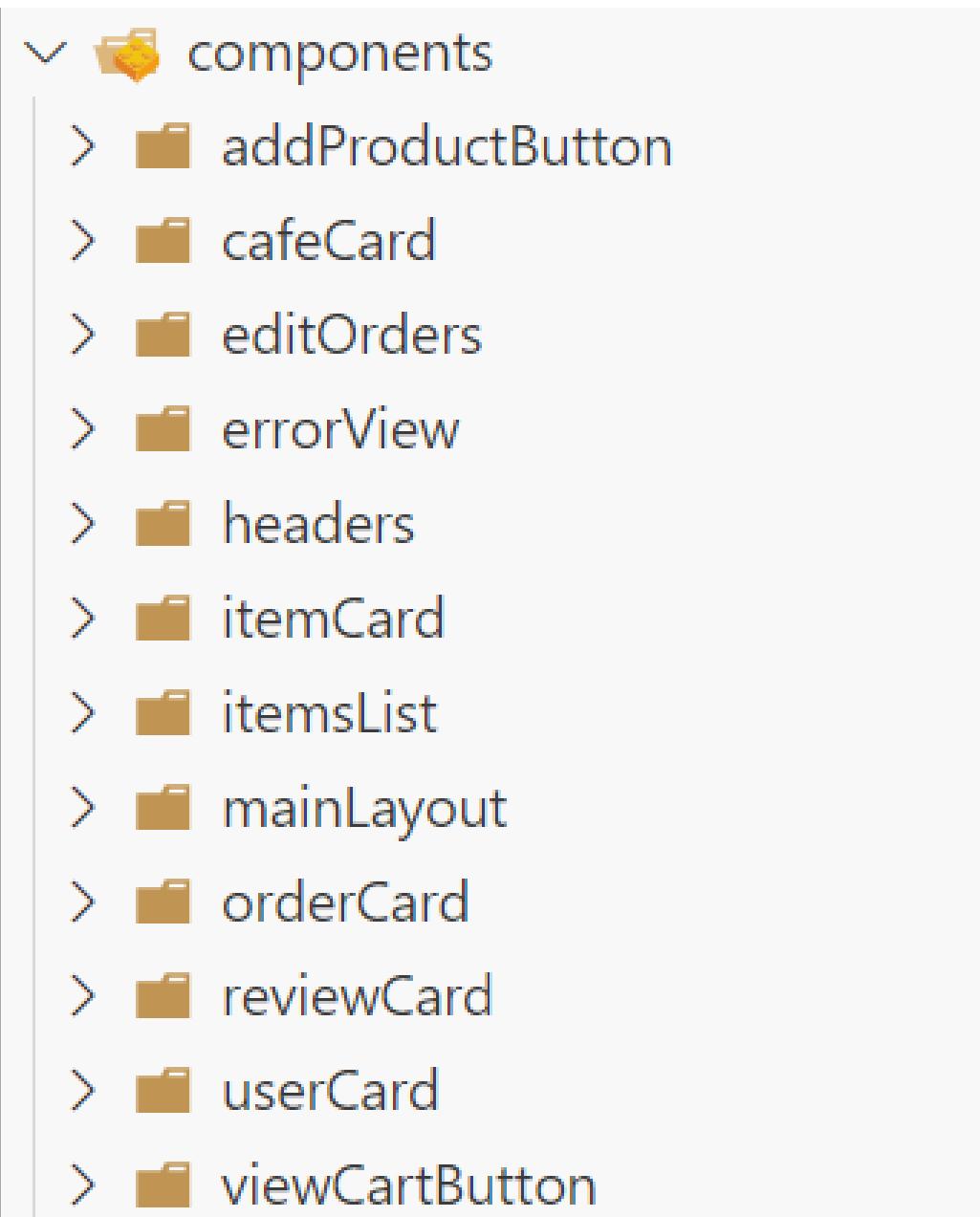


Assets:

Тук се съдържат всички снимки и шрифтове, които се използват за визуализация в приложението. Те не се зареждат от сървъра, ами си винаги налични, дори без връзка с интернет.

Components:

Тази папка съдържа важни компоненти, които изграждат екраните на по-късен етап.



- **AddProductButton:** Съдържа бутона, който се визуализира за добавяне на нов продукт. Той служи повече като прям път, защото се намира в екрана, където се намират всички предлагани продукти от съответното кафене. Бутона се визуализира само, ако потребителят има правата за тази операция.

```
interface CurrentCafeCartButtonProps {  
    cafe: Cafeteria;  
}
```

- **CafeCard:** Съдържа карта, която визуализира информация за кафене в списъка с кафенета. Компонентът показва изображение на кафенето, името, марката, работното време, местоположението и средната оценка. При натискане на картата потребителят се пренасочва към менюто на кафенето.

```
interface CafeCardProps {
  cafe: Cafeteria;
}
```

- **ItemCard:** Съдържа картата, която визуализира продукт в приложението. Това е компонент, който показва изображение на продукта, името и цената му, както и бутон за добавяне в кошницата. Картата се визуализира в списъка с продукти от съответното кафене.

```
interface ItemCardProps {
  product: Product;
}
```

- **LoadingErrorView:** Съдържа компонент, който се визуализира в състояния на зареждане, грешка или липса на данни. Той служи за показване на индикатор за зареждане, съобщения за грешки или алтернативно съдържание, когато липсват данни. Компонентът се използва в различни екрани на приложението.

```
interface Props {
  loading: boolean;
  error?: string | null | undefined;
  dataAvailable: boolean;
}
```

headers: Тази директория съдържа header-и, които се използват от еcranите, които се нуждаят:

- **CafeMenuSubheader:** Съдържа подзаглавен компонент, който се визуализира над менюто на кафене. Той показва името на кафенето и бутон за преглед на детайлите му. Компонентът се използва в екрана с меню на кафене и предоставя бърз достъп до детайлния изглед.

```
interface MenuSubheader {
  cafe: Cafeteria;
```

```
}
```

- **CommonHeader:** Съдържа стандартен заглавен компонент, който се използва в различни екрани на приложението. Компонентът показва заглавие и бутона за връщане назад. Той осигурява консистентен изглед на навигацията в приложението.

```
interface CommonHeaderProps {  
    title: String;  
}
```

- **DetailsHeader:** Съдържа заглавен компонент, който се визуализира на екрана с детайли за кафене. Той показва името на кафенето и бутона за връщане назад. Компонентът предоставя контекст за текущото избрано кафене.

```
interface DetailsHeader {  
    cafeName: string;  
}
```

- **ReviewsHeader:** Съдържа заглавен компонент, който се визуализира на екрана с отзиви. Той показва средната оценка, максималната оценка и броя отзиви за съответното кафене. Компонентът предоставя бърз преглед на рейтинга и бутона за връщане назад.

```
interface ReviewHeaderProps {  
    rating: number;  
    totalReviews: number;}
```

- **ItemsList:** Съдържа компонент, който визуализира списък с продукти, категоризирани по секции. Компонентът групира продуктите по тип (напитки, промоции, храна) и ги показва в отделни секции. Компонентът се използва в екрана с меню на кафене.

```
interface ItemsList {  
    products: Product[];  
}
```

mainLayout: Директорията съдържа header и footer, които се използват за главния экран на приложението:

- **Footer:** Съдържа долната навигационна лента на приложението. Компонентът използва навигация с табове за превключване между различните основни секции на приложението - списък с кафенета и кошица. Всеки таб има икона и етикет, които помагат на потребителя да се ориентира в приложението.
- **Header:** Съдържа горната лента на приложението, която се визуализира на основните екрани. Компонентът показва информация за текущия потребител, включително аватар и потребителско име. Съдържа също бутон за излизане от профила и бутон за административен панел, който се вижда само за потребители с администраторски права.
- **OrderCard:** Съдържа карта, която визуализира информация за поръчка. Компонентът показва номер на поръчката, дата, статус, списък с продукти и тяхното количество, както и общата сума. Статусът на поръчката се визуализира с цветен индикатор, който променя цвета си според текущото състояние на поръчката (доставена, обработваща се, отказана, отложена).

```
interface OrderCardProps {  
    order: Order;  
}
```

- **ReviewCard:** съдържа карта, която визуализира отзив за кафене. Компонентът показва заглавието на отзива, съдържанието, рейтинга (визуализиран със звезди) и датата на създаване. Картата използва стилизация, която я отделя визуално от останалите елементи на страницата и предоставя ясен и структуриран начин за преглед на потребителските мнения.

```
interface ReviewCardProps {  
    review: Review;  
}
```

viewCartButton: директорията съдържа два бутона с минимални разлики помежду им:

- **CurrentCafeCartButton:** визуализира бутон, който служи за пренасочване към количката на текущото кафене. Той се показва, когато потребителят е добавил продукти в количката и остава в същото кафене. Бутонът съдържа брой продукти

и общата им цена. При натискане се извиква “CommonActions.reset”, който нулира навигационния стек и пренасочва потребителя директно към экрана Cart. Това осигурява бърз достъп до количката, без да се натрупват множество екрани в историята на навигацията.

```
interface CurrentCafeCartButtonProps {  
  
    totalPrice: number;  
  
    productsCount: number;  
}
```

- **DifferentCafeCartButton:** визуализира се, когато потребителят се намира в различно кафене от това, от което вече е добавил продукти в количката. Бутона ѝ показва името на текущото кафене, от което са продуктите, както и общия брой артикули и тяхната цена. Подобно на “CurrentCafeCartButton”, натискането му пренасочва директно към экрана Cart чрез “CommonActions.reset”. Служи за напомняне, че потребителят има активна количка от друго заведение.

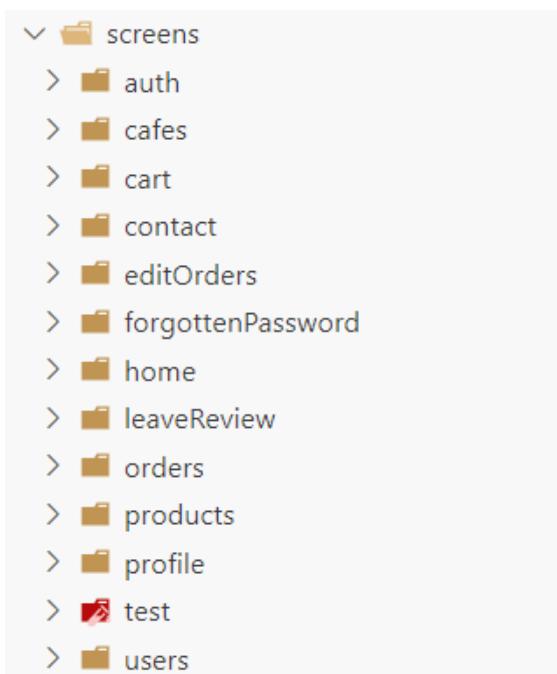
```
interface DifferentCafeCartButton {  
  
    currentCafeteriaName: string;  
  
    totalPrice: number;  
  
    productsCount: number;  
}
```

Screens:

Тази директория съдържа компонентите, които служат като екрани. Теса разделени на подкатегории според тяхната употреба. Екраните се сглобяват с помощта на компоненти от директорията components.

auth: Директорията съдържа 2 компонента, един за вписване и един за регистрация:

- **login:** Екран за вход на потребителите в системата.
 - Позволява въвеждане на потребителско име и парола чрез **TextInput** полета.



- Ако не са попълнени полетата, се визуализира грешка.
- При успешно логване, потребителят се пренасочва към екрана "home".
- Използва `useAuth()` hook, който предоставя метода `login`.
- Има и бутон за пренасочване към "register", както и за „забравена парола“.
- Дизайнът включва фон изображение и лого.

```
const [username, setUsername] = useState("");
const [password, setPassword] = useState("");
const { login } = useAuth();
```

- **register:** Еcran за създаване на нов потребителски профил.
 - Съдържа форми за въвеждане на потребителско име, имейл и парола. Включва валидации (напр. дължина на потребителско име, правилен имейл, минимална дължина на парола).
 - Ако регистрацията е успешна, се пренасочва към "home".
 - При неуспешна – се показва съобщение за грешка.

```
await register(username, email, password);
```

cafes: Директорията съдържа основните екрани, свързани с различните кафенета:

- **cafeCreateScreen:** Форма за създаване на ново кафене (само за администратори).
 - Съдържа валидации за всички полета (часове, телефон, изображение и т.н.).
 - След успешно създаване на кафене се показва съобщение за успех.
 - Визуализира се с фон и scroll-контейнер.

```
const newCafeteria: CreateCafeteriaDTO = { name, brand, location, ... }
```

- **cafeDetailScreen:** Детайлен экран за кафене.
 - Включва информация за име, бранд, адрес, рейтинг, телефон и работно време.
 - Има два бутона: един за контакт и един за писане на ревю.
 - Навигацията се извършва чрез `navigation.navigate(...)`.

```
onPress={() => navigation.navigate("cafereviews", { cafe })}
```

- **cafeListScreen:** Детайлен экран за кафене.
 - Зарежда се чрез `useCafes()` hook.
 - Показва бутона за създаване на кафене, ако потребителят има роля `admin`.
 - Всеки елемент е `CafeCard`.

```
{cafes.map((cafe: Cafeteria) => (<CafeCard cafe={cafe}>))}
```

- **cafeMenuScreen:** Екран с менюто (продуктите) на дадено кафене.
 - Зарежда списък с продукти за текущото кафене чрез hook-а `useProducts()`.
 - Ако няма продукти или има грешка – показва `LoadingErrorView`.
 - Включва бутон за добавяне на продукт `AddProductButton`, който се визуализира само за администратори. Използва `ItemsList`, за да визуализира всички продукти.
 - Показва бутон към количката:
 - `CurrentCafeCartButton`, ако потребителят добавя от текущото кафене.
 - `DifferentCafeCartButton`, ако е в различно кафене от това, в което има продукти.

```
useEffect(() => {
```

```
    fetchAllProductByCafeteriaId(cafe.id);
```

```
}, [cafe.id]);
```

- **cafeReviewsScreen:** Екран с потребителски ревюта за дадено кафене.
 - Зарежда всички ревюта на избраното кафене с `useReviews()`.
 - Извиква `refreshCafeteria()` при фокусиране, за да опресни информацията.
 - Използва `ReviewCard`, за да визуализира всяко ревю.
 - Ако няма ревюта – се визуализира съобщение.
 - Включва `ReviewHeader`, който показва рейтинг и брой ревюта.
 - Плаващ бутон в долната част води към екран за писане на ново ревю:

```
<TouchableOpacity  
    onPress={() => navigation.navigate("leavereview", { cafe })}  
>  
  <Text>Leave a review</Text>  
</TouchableOpacity>
```

- **cart:** Екранът с количката на потребителя.
 - Визуализира всички добавени продукти.
 - Позволява промяна на количествата чрез `updateQuantity()`.
 - Позволява изтриване на продукти с `removeFromCart()`.
 - Има два бутона:
 - `Clear Cart`: изчиства количката.
 - `Proceed to Checkout`: изпраща поръчката чрез `postOrder()` и нулира количката.
 - При действия се използва `expo-haptics` за тактилна обратна връзка.

```
<TouchableOpacity  
    onPress={async () => {
```

```

        await postOrder(currOrder());
        clearCart();
    }
>
<Text>Proceed to Checkout</Text>
</TouchableOpacity>

```

- **contactUs:** Компонентът визуализира формуляр за връзка с администраторите на приложението чрез EmailJS. Потребителят попълва email, име и съобщение и при натискане на бутона „Submit“ информацията се изпраща.
 - Три задължителни полета: Email, Name, Message
 - Изпращане чрез `@emailjs/react-native`
 - Валидират се празни полета
 - Успешно/неуспешно известие чрез `Alert`
 - Състояние на зареждане с променящ се текст на бутона

- **editOrder:** Този компонент е административен интерфейс за преглед и промяна на статуса на поръчки със статус `PROCESSING`. Извлича поръчки и свързаните с тях продукти, позволява редакция чрез `Picker` и визуализира общата сума и списък с артикули.
 - Извличане на всички поръчки със статус `PROCESSING`
 - Преглед на информация за всяка поръчка: дата, статус, продукти, обща цена
 - Селекция на статус чрез `<Picker>` — промяна се изпраща с `updateOrderStatus()`
 - Всеки продукт от поръчката се показва с име, цена, количество и сума
 - Визуален индикатор за статус чрез цветна точка

forgottenPassword: В тази директория има два компонента, които позволяват на потребителя да възстанови достъп до акаунта си, в случай на забравена парола:

- **ForgotPassword:** Екран за заявка за възстановяване на парола.
 - Съдържа поле за въвеждане на имейл адрес и два бутона – за изпращане на линк и за въвеждане на токен.
 - При натискане на "Send Reset Link", се изпраща POST заявка към `/api/password/resetToken/:email`.
 - Използва `customAPI.post(...)` и навигира към „resetpassword“ при натискане на втория бутон.
 - Проверява дали полето за имейл е попълнено.

- **ResetPassword:** Екран за въвеждане на токен и нова парола.
 - Съдържа три полета – токен, нова парола и потвърждение.
 - Включва валидация.

- При успешна валидация, се изпраща POST заявка към `/api/password/reset-password` с токен и новата парола.
 - При успех се визуализира съобщение, при грешка – подходящо съобщение.
 - Използва `customAPI.post(...)`.

- **home:** Това е основния екран, който е съставен от header-a и footer-a от mainLayout и съответния екран, който е избран във footer-a.
 - Съдържание:


```
const Home = () => {
  return (
    <ProtectedRoute>
      <View style={styles.container}>
        <Header/>
        <Footer/>
      </View>
    </ProtectedRoute>
  );
};
```

- **LeaveReview:** Екран за добавяне на ревю към конкретно кафе. Позволява на потребител да въведе оценка (със звезди), заглавие и коментар.
 - Валидации: изисква заглавие и оценка, проверява дали потребителят е логнат.
 - След успешно изпращане: ревюто се подава чрез `postReview(...)`, връща потребителя назад.
 - Използва `AirbnbRating`, `CommonHeader`, `Haptics`.


```
await postReview({ title, body, rating, cafeteriaId, userId });
```

- **Orders:** Екран за преглед на направени поръчки на текущия потребител.
 - Зарежда поръчките с hook-a `useOrders(user?.id)`.
 - Ако няма поръчки, показва съобщение.
 - Ако има – визуализира всяка чрез `OrderCard`.


```
const { orders, loading, error } = useOrders(user?.id);
```

- **CreateProduct:** Екран за администраторско добавяне на нов продукт към кафе.
 - Полета: име, цена, тип продукт (dropdown), изображение (URL).
 - Включва валидации и визуализира грешки при липсващи или невалидни стойности.
 - При успех създава нов продукт чрез API и връща назад.


```
await customAPI.post("/api/products", {
  name, price, productType, imageUrl, cafeteriaId
```

```
});
```

- **Profile:** Потребителски профил с възможност за редакция на username и email чрез модален прозорец.
 - Извлича информация за потребителя чрез `useUser`.
 - Позволява редакция на потребителските данни и изпращане с `updateUser(...)`.
 - Съдържа навигация към `Orders` еcran и достъп до всички потребители (ако е админ).
 - Използва `Modal`, `CommonHeader`, `HasRoles`, `List.Accordion`.

Users: Тук се съдържат няколко прозореца, използвани от потребители със администраторски права, които служат като панел за работа на администраторите

- **CreateUser:** Екран за администраторско създаване на нов потребител.
 - Въвеждат се username, email, парола и роли.
 - Съдържа валидации: минимални дължини, формат на имейл, поне една роля (ако е админ).
 - След създаване: формата се нулира и показва потвърждение.

```
await customAPI.post('/api/users', {  
    username, email, passwordHash, roleNames  
});
```
- **UserEdit:** Администраторски екран за редакция на съществуващ потребител.
 - Зарежда потребител по ID с `useUser(...)`, попълва формата.
 - Възможна е промяна на username, email и роли.
 - При грешка показва съобщение, при успех — потвърждение.

```
await updateUser({ id, username, email, roles }, id);
```
- **UserList:** Екран за преглед на всички потребители (без текущия).
 - Извлича списък с всички потребители чрез `useAllUsers()`.
 - Филтрира текущия потребител чрез `useAuth()`.
 - Ако няма други потребители, показва съобщение.
 - Всеки потребител се рендерира с `UserCard`.

```
const filteredUsers = users.filter(u => u.id !== currentUser.id);
```
- **UserReviewList:** Екран за всички ревюта, написани от конкретен потребител.
 - Взима `userId` от `route.params`.
 - Извлича ревюта с `useUserReviews(userId)`.
 - Ако няма ревюта, показва съобщение "No reviews".
 - Използва `ReviewCard` за визуализация на всяко ревю.

Navigation:

В тази директория се намират 2 компонента, които служат за навигация и ограничаване на контрол до определени функционалности:

- **ProtectedRoute:** този компонент служи като спирачка, ако потребителя по някакъв начин успее да достъпи home. Той проверява, дали има вписан потребител и отхвърля достъпа, ако няма.
- **Navigation:** Този файл използва [@react-navigation/native-stack](#), за да дефинира всички екрани в приложението и маршрутизира потребителите между тях. Всички екрани са регистрирани в `Stack.Navigator`, а навигацията е типизирана чрез `RootStackParamList`.
 - Чрез `RootStackParamList` се дефинира какви параметри приема всеки еcran, което помага за:
 - безопасно предаване на `props`;
 - TypeScript автодовършване;
 - лесно проследяване на всички навигационни пътища в приложението.
 - Пример за навигиране с параметри:
`navigation.navigate("cafemenu", { cafe });`

Services:

В тази директория има само един компонент, който е:

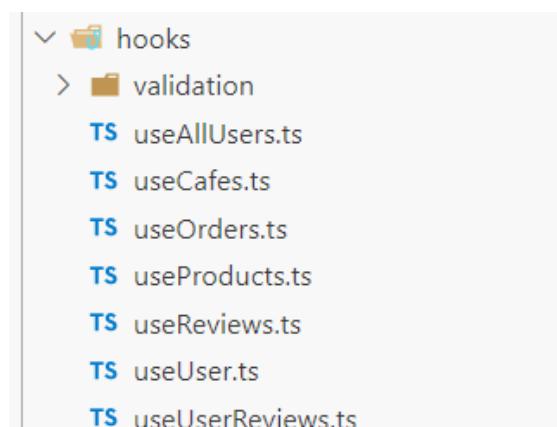
- **apiService:** този компонент създава HTTP клиент, който комуникира със зададения адрес във .env файла, освен това той добавя jwt token във header-а на всяка заявка към сървъра:

```
const apiUrl = Constants.expoConfig?.extra?.apiUrl;
const customAPI = axios.create({
  baseURL: apiUrl
});
```

Hooks:

Тук се намират компонентите, които изпращат заявки към сървъра и приемат данните от него. Те използват `apiClient.ts`. Тъй като те са еднотипни ще разгледаме само един примерен hook:

- **useUser:** този hook, може да приема като параметър число или нищо, след което изпраща една от две заявки (зависи според нуждите) към сървъра, като след обработката на заявката от сървъра той връща отговор.



```

const useUser = (id?: number) => {
  const [user, setUser] = useState<User | null>(null);
  const [loading,  setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  const fetchUserById = async (id: number) => {
    try {
      const response = await customAPI.get(`api/users/${id}`);
      setUser(response.data);
    } catch (error: any) {
      setError(error?.response?.data?.message || error.message || 'Something went wrong');
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    if (id != null) {
      fetchUserById(id);
    }
  }, [id]);
}

const updateUser = async (updatedUser: UserUpdate, id: number) => {
  try {
    const response = await customAPI.put(`api/users/${id}`, updatedUser);
    setUser(response.data);
  } catch (error: any) {
    setError(error?.response?.data?.message || error.message || 'Something went wrong');
  }
};

return { user, updateUser, loading, error };
};

```

Context:

Тук имаме два компонента от тип context, чиято цел е да предоставя информация на всички компоненти намиращи се в тялото на тези. Те се инициализирани възможно най-високо.

- **AuthContext:** този context предоставя на компонентите информация за потребителя, а именно: id, username, roles. Тук също така се извършват и операциите по вписване и регистриране

```
const login = async (username: string, passwordHash: string) => {
  const response = await customAPI.post('api/auth/login', { username, passwordHash });
  const token = response.data;
  await AsyncStorage.setItem('jwt', token);
  const decoded = decodeToken(token);
  console.log(decoded.id);
  setUser(decoded);
};
```

- **CartContext:** този context съдържа информация за продуктите в количката на потребителя. Освен това, той съдържа и методи за модифициране (добавяне, премахване на продукти, изчистване на количката и др.)

●

```
type CartContextType = {
  cartItems: CartItem[];
  addToCart: (product: Product) => void;
  updateQuantity: (productId: Product, quantity: number) => void;
  removeFromCart: (product: Product) => void;
  clearCart: () => void;
  currentCafeteria: Cafeteria | null;
  productsCount: number;
  totalPrice: number;
  currOrder: () => Order;
};
```

Theme:

Тук имаме само един файл:

- **theme:** това е компонент със често използвани теми, които са характерни за това приложение. Съдържа:
 - Цветове
 - Отстояния

Types:

В тази директория има само един файл, чието име е:

- **types:** този файл съдържа типовете на данните, чрез които приложението комуникира със сървъра
- **пример:**

```
export type User = {
```

```
id: number;  
username: string;  
email: string;  
roles: Role[];  
orders: Order[];  
reviews: Review[];  
}
```

Utils:

В тази директория има един компонент:

- **HasRoles:** съдържа единствено метод, който проверява дали потребителът, който използва приложението има съответната желата роля, която се подава чрез prop.

```
interface HasRolesProps {  
  roles: string[];  
  children: React.ReactNode;  
}  
  
const HasRoles = ({ roles, children }: HasRolesProps) => {  
  const { user } = useAuth();  
  
  if (!user) return null;  
  
  // Check if the user has at least one of the specified roles  
  const hasAnyRole = roles.some(requiredRole =>  
    user.roles.some(role => role.authority === requiredRole)  
  );  
  if (!hasAnyRole) return null;  
  return <>{children}</>;  
};
```

Допълнителни функционалности:

```
export const OrientationContext : Context<Orientation> = createContext<Orientation>('UNKNOWN'); Show usages ▾ Angel LS

export const OrientationProvider : ((children : { children: ReactNode }) => { children : { children: ReactNode } }) => {
  const [orientation, setOrientation] = useState<Orientation>('UNKNOWN');

  useEffect(() => {
    const updateOrientation : (orientation: Orientation) => void = (orientation: ScreenOrientation.Orientation) => { Show
      if (
        orientation === ScreenOrientation.Orientation.LANDSCAPE_LEFT ||
        orientation === ScreenOrientation.Orientation.LANDSCAPE_RIGHT
      ) {
        setOrientation('LANDSCAPE');
      } else if (
        orientation === ScreenOrientation.Orientation.PORTRAIT_UP ||
        orientation === ScreenOrientation.Orientation.PORTRAIT_DOWN
      ) {
        setOrientation('PORTRAIT');
      } else {
        setOrientation('UNKNOWN');
      }
    };
    ScreenOrientation.getOrientationAsync().then(updateOrientation);

    const subscription : EventSubscription = ScreenOrientation.addOrientationChangeListener((evt : OrientationChangeEvent
      | updateOrientation(evt.orientationInfo.orientation);
    ));

    return () => {
      ScreenOrientation.removeOrientationChangeListeners();
    };
  }, []);

  return (
    <OrientationContext.Provider value={orientation}>
      {children}
    </OrientationContext.Provider>
  );
};

const App : () => Element = () => { Show
  return (
    <GestureHandlerRootView>
      <OrientationProvider>
        <AuthProvider>
          <CartProvider>
            <Navigation />
          </CartProvider>
        </AuthProvider>
      </OrientationProvider>
    </GestureHandlerRootView>
  );
};

export default App; no usages ▾ Matri
```

Cafe-Dash разполага с orientation, което позволява по-удобен начин за интеракция в приложението. Ако ориентацията е портретната и завъртим телефона, тя ще се сменя на пейзажна. OrientationProvider е Wrapper компонент, който обхваща цялото приложение.

Security:

Сигурността на приложението е **изцяло централизирана в Backend**, където се извършва реалната проверка на токена, валидирането на потребителя и ролите му, както и контролът на достъпа до защитени ресурси чрез Spring Security.

useAuth

Custom hook-ът useAuth изпълнява ключова роля в управлението на сигурността и потребителската сесия във frontend частта на приложението. Макар че всички проверки за автентичност и права се извършват от сървъра чрез Spring Security, useAuth предоставя необходимата логика от клиентска страна за локален контрол и визуално ограничаване на достъпа. Основните му отговорности включват:

- **Съхраняване и декодиране на JWT токена:** След успешен вход или регистрация, токентът се записва в AsyncStorage, декодира се и се извлича информацията за потребителя (ID, потребителско име, роли).
- **Управление на локалното състояние на потребителя:** Декодираната информация се записва в useState и се използва навсякъде в приложението за показване/скриване на компоненти.
- **Предоставяне на глобален достъп до потребителя:** Чрез AuthContext hook-ът прави информацията за текущия потребител достъпна във всяка част на приложението чрез useAuth().
- **Функции за вход, регистрация и изход:** Hook-ът предоставя централизирани методи (login, register, logout), които управляват токена и състоянието на потребителя съобразно действието.

useAuth не извършва реална защита на ресурсите, но осигурява сигурно потребителско изживяване чрез контрол над това какво да се визуализира в интерфейса спрямо ролите на логнатия потребител.

Използване на useAuth hook примери:

```
const { user, logout } = useAuth(); - зарежда се стойността на потребителя, запазена в AuthContext. Компонентът , който ползва този код трябва да се намира задължително в рамките на AuthContext.Provider
```

Във **Frontend** частта, целта е да се предотврати визуализация на компоненти и функционалности, до които потребителят **няма роля** или **няма нужните права**. Това се реализира чрез компонент наречен **HasRoles.tsx**

HasRoles.tsx

- Този компонент действа като **визуална защита (UI Guard)**. Той скрива или показва даден React елемент в зависимост от ролите на текущия потребител, заредени чрез **AuthContext**.

- **Основна логика:**
 - Ако потребителят **не е логнат** – нищо не се рендерира.
 - Ако потребителят **няма нито една от подадените роли** – нищо не се рендерира.
 - Ако потребителят **има поне една съвпадаща роля** – децата на компонента (children) се визуализират.

Deployment реализация:

Проектът е деплоинат на Microsoft Azure, използвайки Azure Virtual Machine (VM) за backend сървъра и Azure Blob Storage за съхранение на изображения.

Архитектура:

Backend:

Git repository е копирано върху самата VM. Необходимият порт за публичен достъп е **9999**. Това е осъществено чрез Azure Network Security Group (NSG) правила, за да се позволи външен достъп до backend.

Priority ↑	Name	Port	Protocol	Source	Destination	Action
▼ Inbound port rules (8)						
300	▲ SSH	22	TCP	Any	Any	Allow
320	HTTPS	443	TCP	Any	Any	Allow
340	HTTP	80	TCP	Any	Any	Allow
360	AllowAnyCustom9999Inbound	9999	TCP	Any	Any	Allow
370	AllowAnyCustom8081Inbound	8081	TCP	Any	Any	Allow
65000	AllowVnetInBound ⓘ	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalancerInbound ⓘ	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInbound ⓘ	Any	Any	Any	Any	Deny
➤ Outbound port rules (3)						

База данни:

Хоства се локално на същата VM, с подходящи настройки за сигурност и достъп.

```
astoynov@tu-sofia.bg@azure-backend:~/Cafe-Dash$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
2c2bd1e65926        cafe-dash_backend   "java -jar app.jar ..."   3 hours ago       Up 3 hours          0.0.0.0:9999->8080/tcp, ::ffff:9999->8080/tcp   cafe-dash_backend_1
73aaaf4fc2c00        postgres:15-alpine    "docker-entrypoint.s..."   3 hours ago       Up 3 hours (healthy)  0.0.0.0:5432->5432/tcp, ::ffff:5432->5432/tcp   cafe-dash_db_1
astoynov@tu-sofia.bg@azure-backend:~/Cafe-Dash$ make up_sql
Start database
docker exec -it cafe-dash_db_1 psql -U postgres -d cafe
psql (15.12)
Type "help" for help.

cafe=# SELECT * FROM users;
 id |      email       | is_deleted |                                password | username
---+----------------+-----+-----+
 2 | koceto@gmail.com | f           | $2a$10$4p.EoCq/yrQuKdAA4/ds0.P.3Lx0TNk1WVGDV7bst6jrc9Wc1MWUG | koceto
 3 | client@gmail.com | f           | $2a$10$n4UA6fx.U0KJxURd94V.8uCwY09HbDs4izi.ul.5BOLkwNi5AixGS | client
 1 | angelstoinov@gmail.com | f           | $2a$10$zpgpisu8p6x1hPYW10smzEemLwhLFRNuoir1WmyoVSDj70jeK/EldDu | angel
 4 | push@gmail.com   | f           | $2a$10$aILBmXWiVWxo10oS4UpKZeBihjWgJBbw/2dSueNCJWFXY6Tab2oe. | push123
(4 rows)

cafe=#

```

Съхранение на изображения:

Използва се Azure Blob Storage. Разрешен е достъп до изображенията чрез конфигуриране на CORS правила, за да се избегнат проблеми при достъп от frontend.

CORS настройките за Blob Storage съдържат разрешени HTTP методи (GET, OPTIONS) и wildcard (*) или конкретен origin за frontend приложението.

Name	Last modified	Anonymous access level	Lease state	...
Logs	4/2/2025, 11:09:47 AM	Private	Available	...
cafeteria-image-blob	3/31/2025, 12:07:33 PM	Blob	Available	...
product-image-blob	3/31/2025, 2:22:44 PM	Blob	Available	...
svgs	3/31/2025, 2:30:39 PM	Blob	Available	...

Конфигурация:

Всички чувствителни данни, като ключове за blob достъп или конфигурации, са пазени в .env файл или системни променливи на VM.

Docker конфигурация:

Използвано е docker-compose.yml, за да мениджираме отделните контейнери. Съответно имаме и dockerFile в backend, който е конфигуриран да build-не.

```
version: '3.8'

services:
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    env_file:
      - backend/.env
    ports:
      - "9999:8080"
    depends_on:
      db:
        condition: service_healthy
    networks:
      - app-network
    restart: always

  db:
    image: postgres:15-alpine
    restart: always
    environment:
      POSTGRES_DB: cafe
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: 12345
    volumes:
      - pgdata:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    networks:
      - app-network
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 10s
      timeout: 5s
      retries: 5
    volumes:
      pgdata:
        name: pgdata
    networks:
      app-network:
        driver: bridge

FROM openjdk:17-jdk

WORKDIR /app
COPY target/backend-*.jar app.jar
EXPOSE 8080

CMD ["java", "-jar", "app.jar", "--server.address=0.0.0.0"]
```

Makefile:

За улеснение при разработка и деплоимент е създаден Makefile, който включва често използвани команди.

```
SHELL=/bin/bash

up:
    docker-compose down
    docker-compose up -d
    @echo Docker containers are up and running

up_build:
    docker-compose up -d --build
    @echo Docker containers are build and running

down:
    docker-compose down
    @echo Docker containers are down

logs:
    docker logs -f cafe-dash_backend_1
    @echo Logs comming from the spring boot

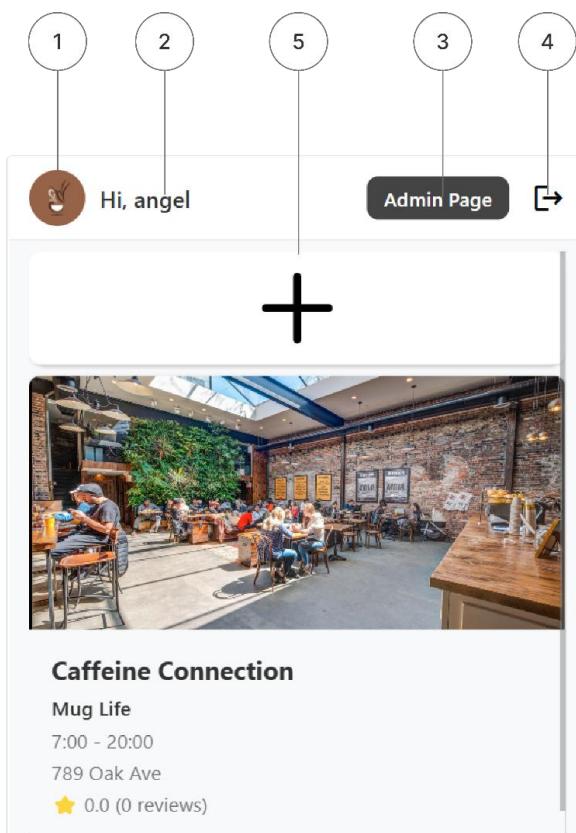
up_sql:
    @echo Start database
    docker exec -it cafe-dash_db_1 psql -U postgres -d cafe
```

Тези команди са често използвани в пускането и мениджирането на контейнерите. Улеснява работата, понеже целият този процес се осъществява през конзолата. За всяка команда е оставено съобщение, което се извежда в конзолата всеки път, когато се използва някоя от командите.

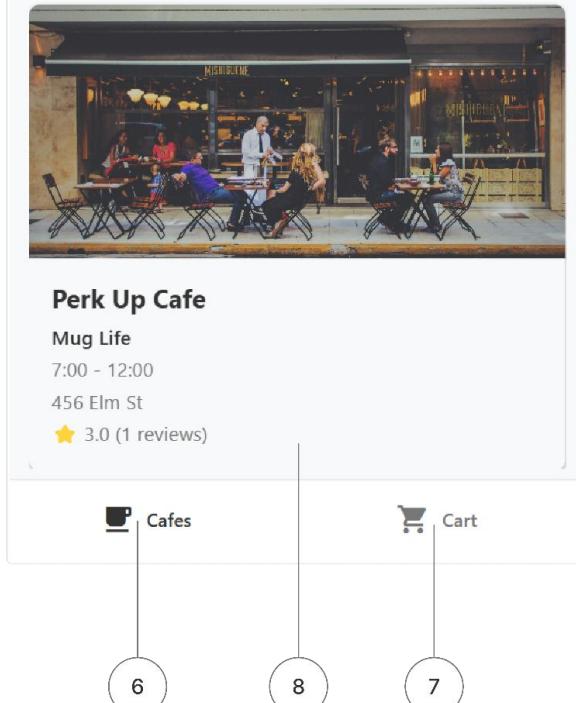
Потребителско ръководство:

Общ преглед на някои екрани:

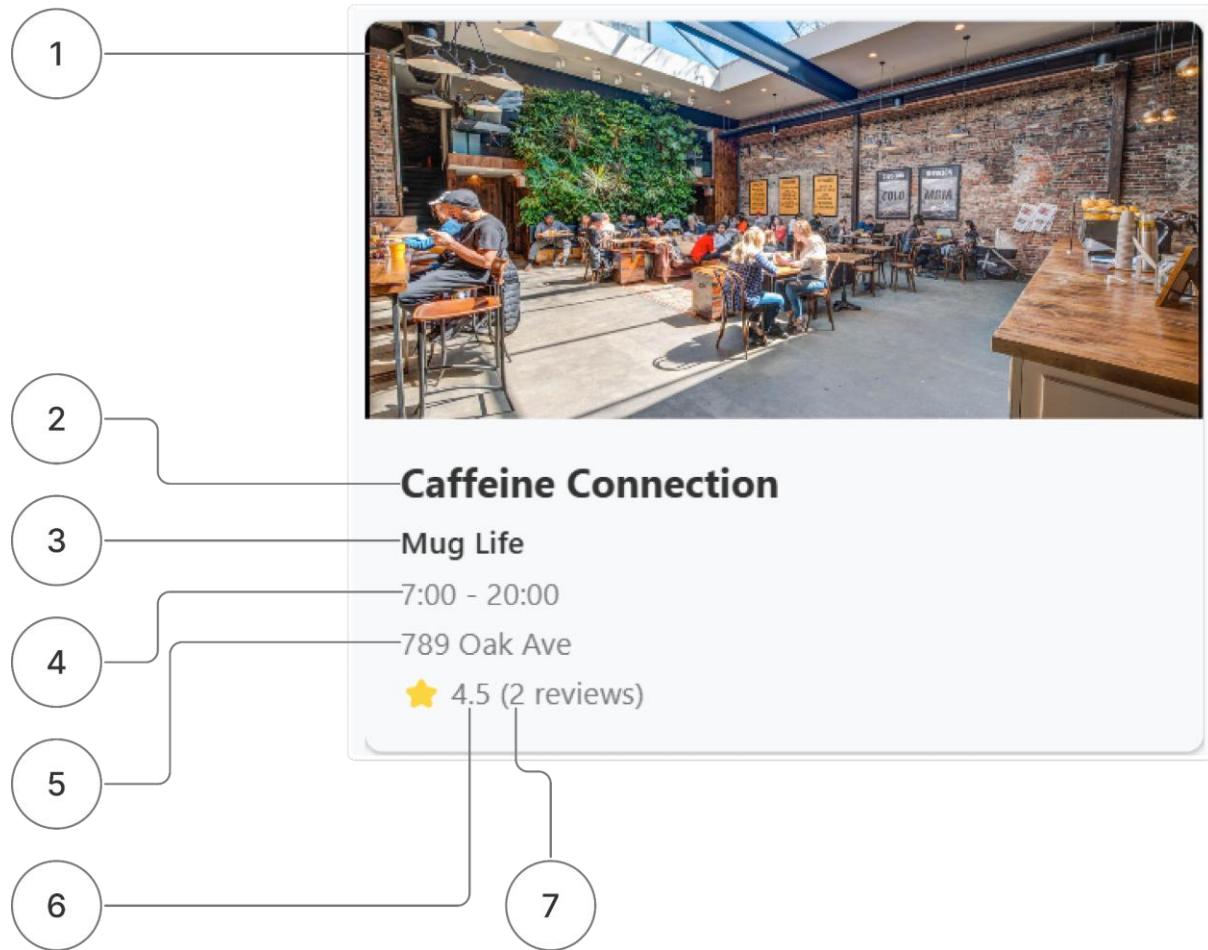
Home:



- 1 - Лого на приложението
- 2 - Username на потребителя
- 3 - Бутон, за достъп до администраторски функционалности (само потребителите със специални роли имат достъп)
- 4 - Бутон за изход
- 5 - Бутон за добавяне на кафене
- 6 и 7 - Toggle за превключване между екрана със кафенета и екрана на количката
- 8 - CafeCard - Карта с кратка информация за кафенето

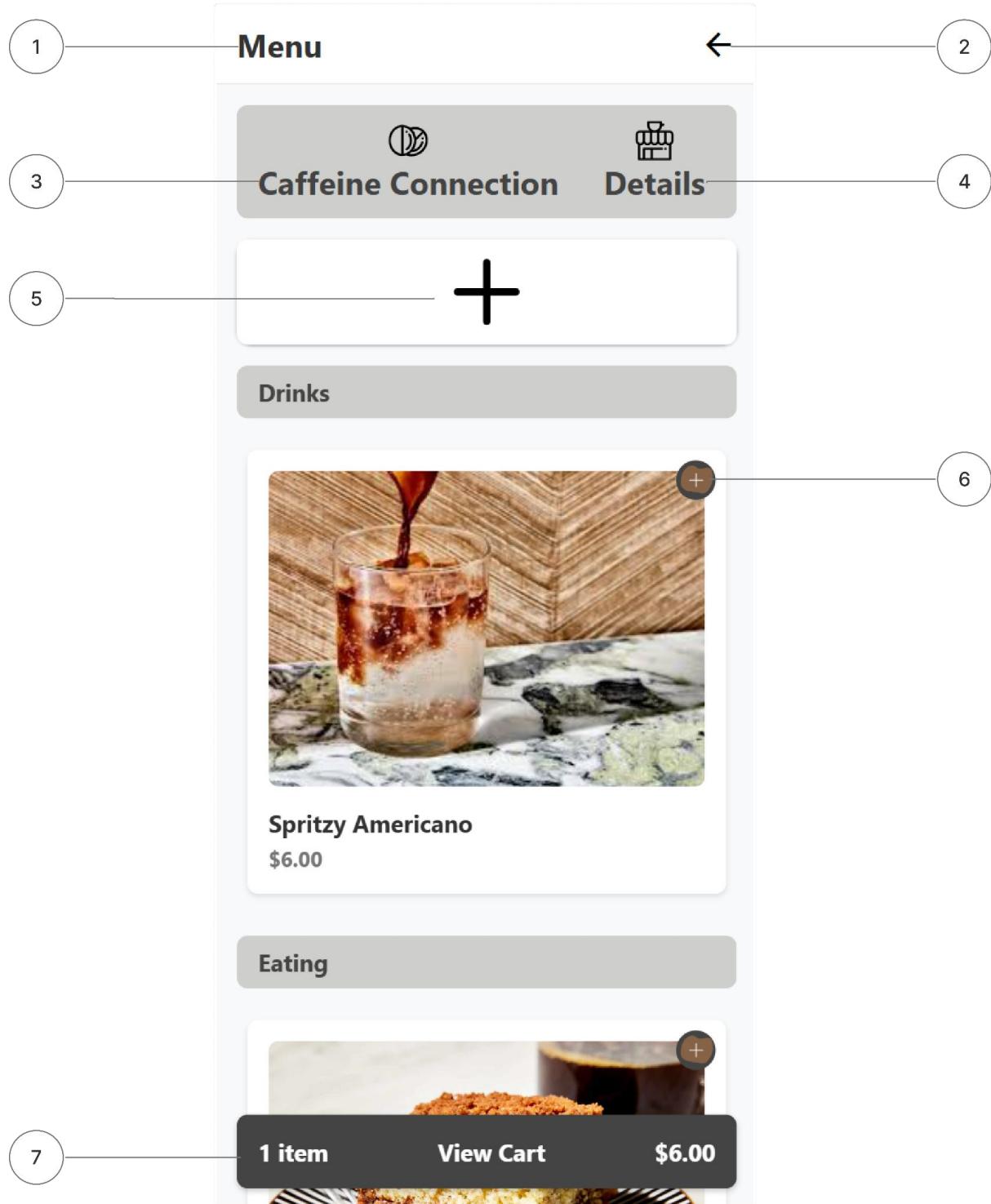


CafeCard:



- 1 - Снимка на обекта
- 2 - Име на обекта
- 3 - Бранд на обекта (ако има)
- 4 - Работно време
- 5 - Адрес на обекта
- 6 - Средна оценка на ревюта за този обект
- 7 - Брой ревюта

Cafe Menu Screen:



Cafe Details Screen:

1

2

3

4

5

Caffeine Connection

Brand:
Mug Life

Address:
789 Oak Ave

Rating:
0.0 (0 Reviews)

Opening Hours:
7:00 - 20:00

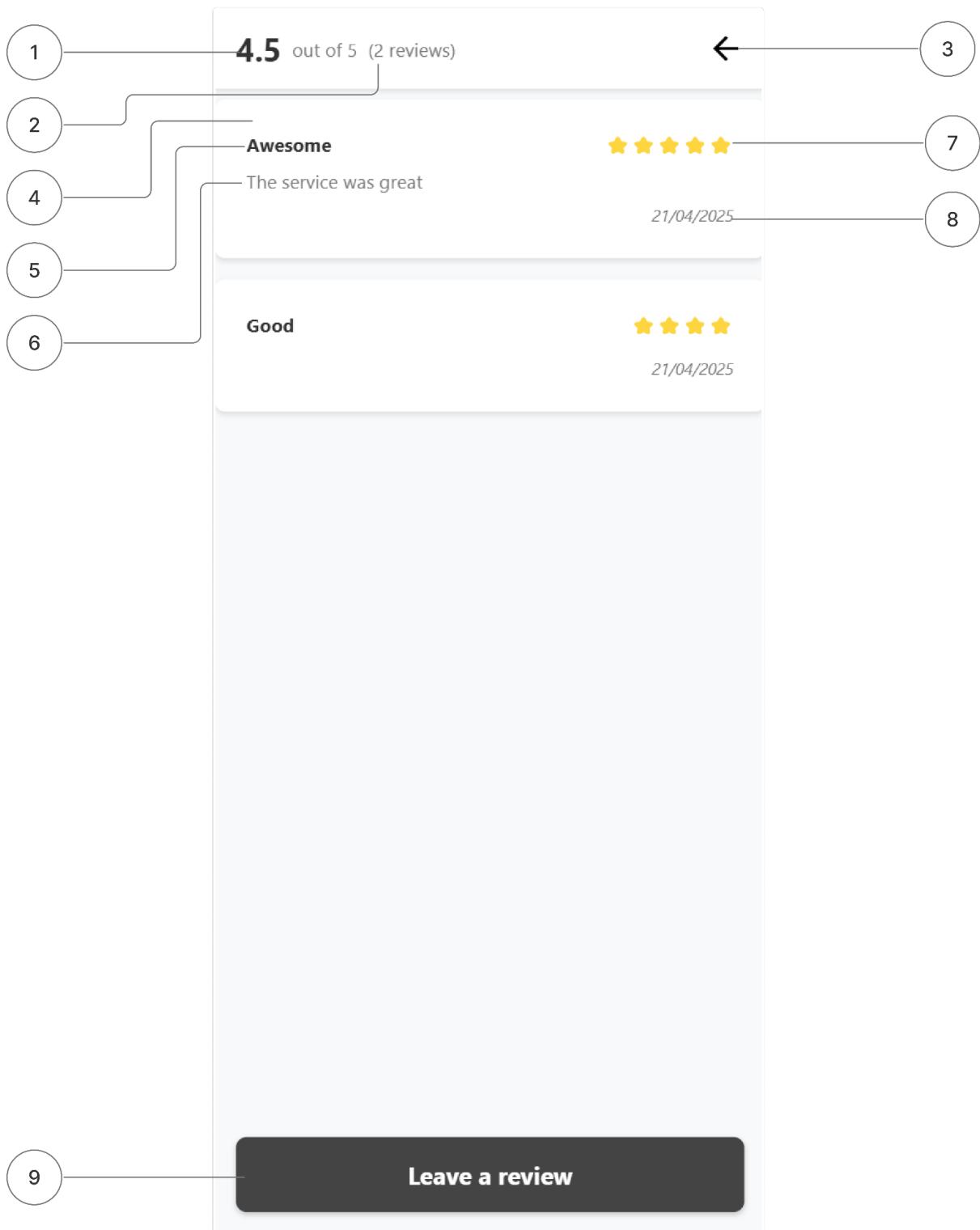
Phone:
18066958703

Contact us here

Leave Caffeine Connection a review ★

- 1 - Име на кафенето
- 2 - Бутон, който води към предното меню
- 3 - Информация за кафенето
- 4 - Бутон, който препраща потребителя към контактна форма, с цел започване на лична комуникация с кафенето
- 5 - Бутон, който води към екрана с отзиви

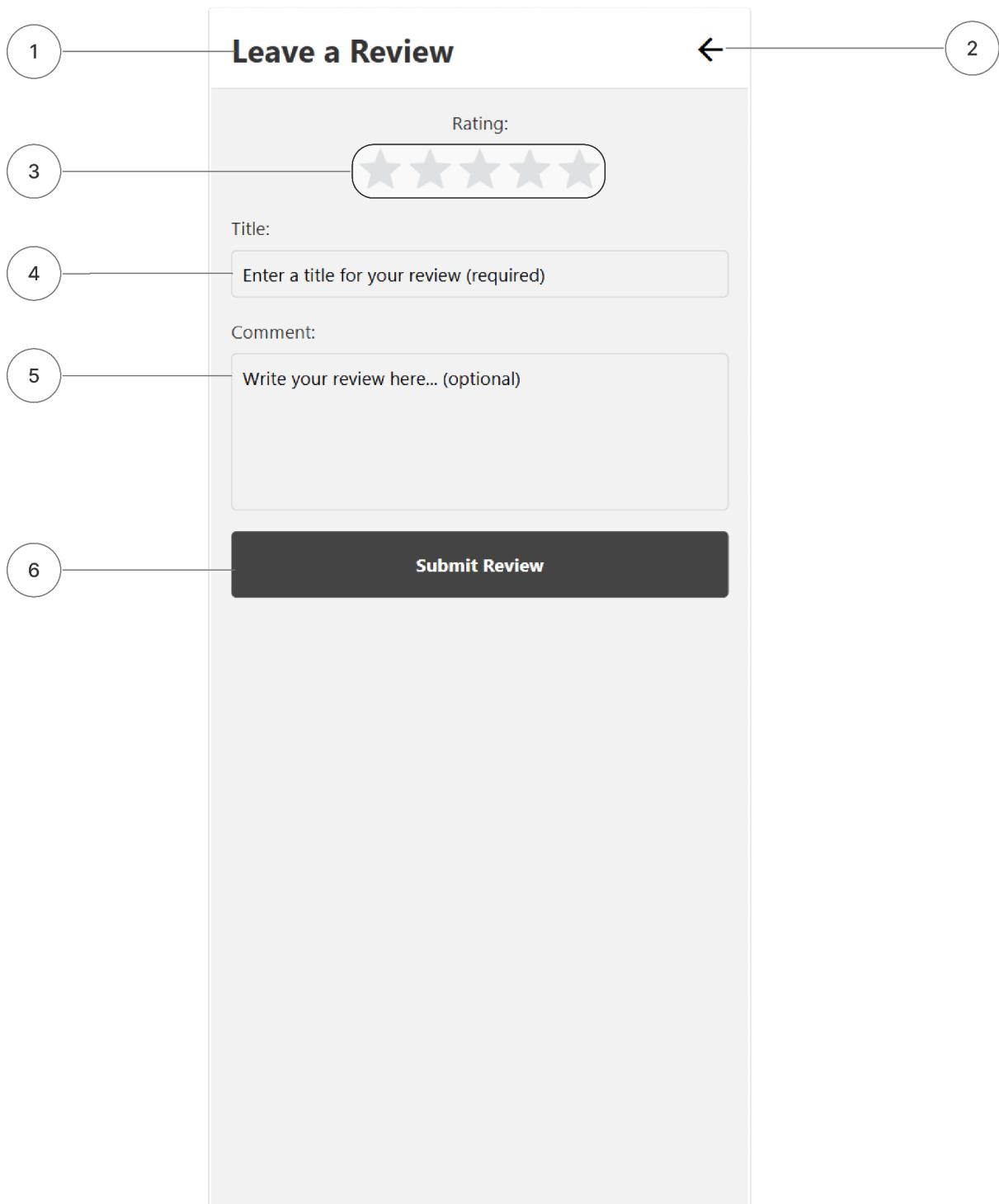
Cafe Reviews Screen:



- 1 - Средно ниво на отзивите
- 2 - Брой отзиви
- 3 - Бутон за назад
- 4 - Карта, съдържаща отзив
- 5 - Заглавие на отзива
- 6 - Тяло на отзива

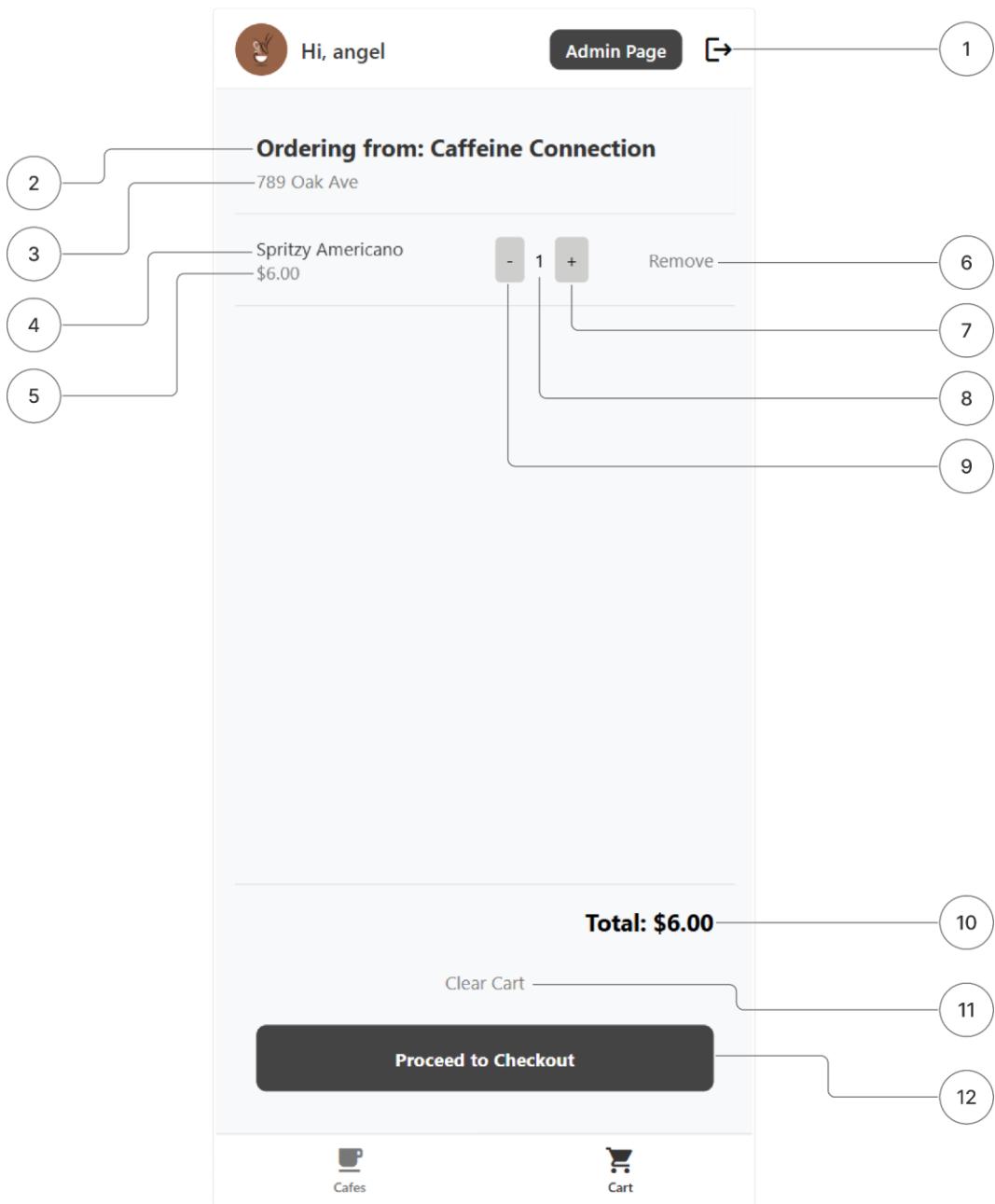
- 7 - Рейтинг на отзива (от 1 до 5)
- 8 - Дата на оставяне на отзива
- 9 - Бутон за оставяне на отзив

Leave Review Screen:



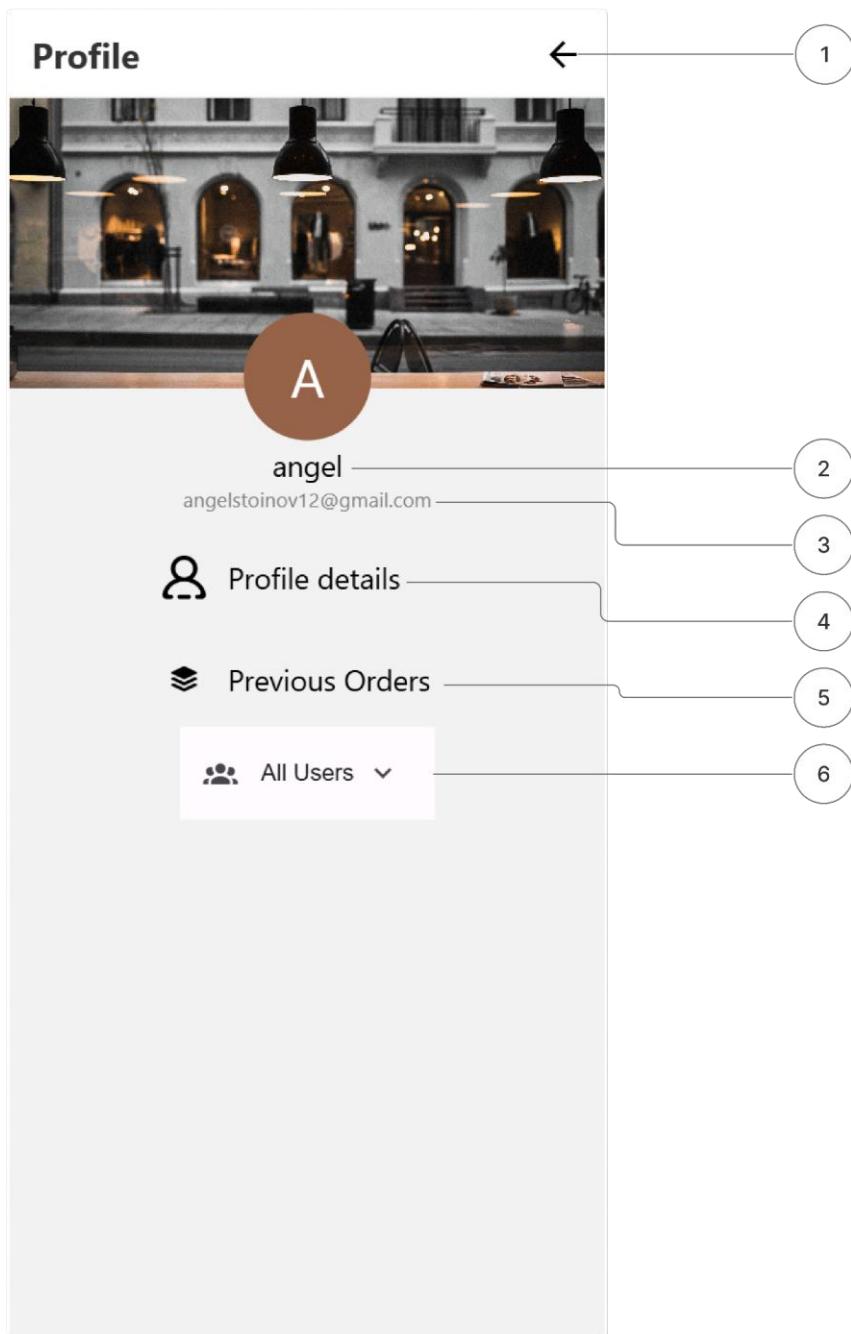
- 1 - Заглавие на екрана
- 2 - Бутон за връщане назад
- 3 - Поле за оставяне на рейтинг на отзива
- 4 - Поле за попълване на заглавие на отзива
- 5 - Поле за оставяне на тяло на отзива
- 6 - Бутон за оставяне на отзив

Cart Screen:



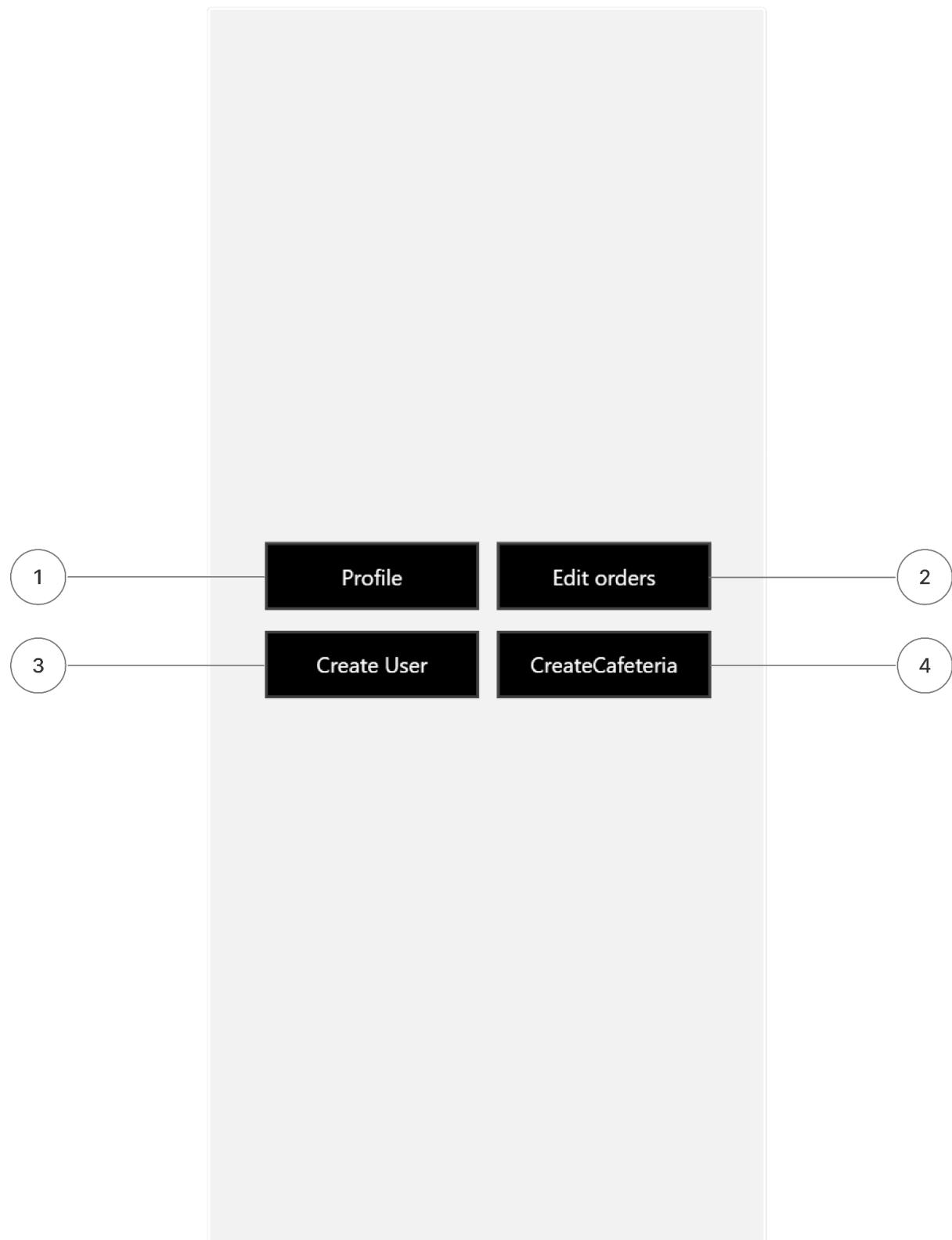
- 1 - Бутон за излизане от акаунт
- 2 - Името на кафето, от където се поръчва
- 3 - Адрес на кафето, от където се поръчва
- 4 - Име на продукта в количката
- 5 - Цена за продукта (единична цена * брой)
- 6 - Бутон за премахване на продукта от количката
- 7 - Бутон за добавяне на един брой от същия продукт
- 8 - Брояч на съответния продукти
- 9 - Бутон за премахване на един брой от съответния продукт
- 10 - Обща стойност на всички продукти в количката
- 11 - Бутон за премахване на всички продукти в количката
- 12 - Бутон за пускане на поръчка

Profile Screen:



- 1 - Бутон за връщане назад
- 2 - Потребителско име на влезналия потребител
- 3 - Имейл на влезналия потребител
- 4 - Бутон, който отваря модал, където потребителя може да смени данните
- 5 - Бутон, който отваря еcran със всички поръчки от логнатия потребител
- 6 - Drop-down меню, със всички потребители, което позволява промяна на данните на потребителите (достъпно само от администратори)

Admin Panel:



1 - Бутон, който води до профил менюто

2 - Бутон, който води до еcran, където администратор (или профил със превилните права) може да променя статуса на поръчката

3 - Бутон, който води до форма за създаване на потребител от администратор

4 - Бутон, който води до форма за създаване на кафене от администратор

Начин на употреба:

Логин:

- 1 Navigate to <http://localhost:8081/login>



При стартиране на приложението, това е първата страница, която се зарежда.

- 2 Click this username field.



- 3 Type "username"

Ако имате акаунт, попълвате потребителско име, ако нямате акаунт натиснете Sign Up.
Необходимо потребителското име да е поне 6 символа.

- 4** Click this password field.



Popълнете паролата си.

Необходимо тя да бъде поне 8 символа.

- 5** Click "Logging in..."



Натиснете Login, ако всичко е наред, системата ще ви отведе в страницата с кафетерии.

Ако има някакъв проблем с автентификацията ви, системата ще ви даде подробна информация за проблема. При забравено поле, грешно попълнено грешно или ако системата временно не работи, ще бъдете уведомени.

6 Welcome to Cafe-Dash

The screenshot shows a user interface for managing cafe profiles. At the top, there's a header with a profile picture, the name "Hi, angel", and buttons for "Admin Page" and "Logout". Below the header, there's a large plus sign icon, likely for adding new cafes. Two cafe profiles are listed:

- Caffeine Connection**
Mug Life
7:00 - 20:00
789 Oak Ave
0.0 (0 reviews)
- Perk Up Cafe**
Mug Life
7:00 - 12:00
456 Elm St
3.0 (1 review)

Честито, успешно се логнахте, разгледайте нашите продукти.

Правене на поръчка:

- 1 Navigate to <http://localhost:8081/home/Cafes>

The screenshot shows a mobile application interface with a light gray background. At the top, there is a large white button with a black 'T' icon. Below it, there are two cards for different cafes.

Caffeine Connection
Mug Life
7:00 - 20:00
789 Oak Ave
★ 0.0 (0 reviews)

Perk Up Cafe
Mug Life
7:00 - 12:00
456 Elm St
★ 3.0 (1 reviews)

Трябва да сте логнати преди това. Акаунтът, с който сте логнати няма значение с каква роля е.



Perk Up Cafe

Mug Life

7:00 - 12:00

456 Elm St

★ 3.0 (1 reviews)

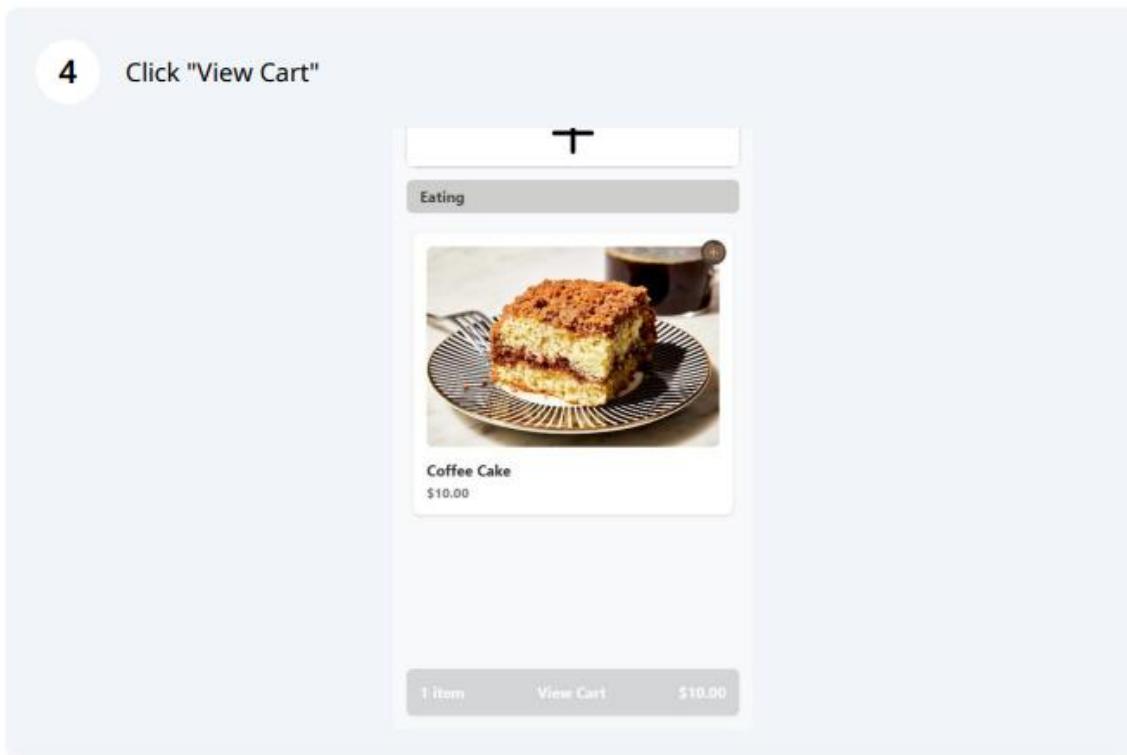
Изберете кафетерия, от която искате да направите поръчка.



Coffee Cake
\$10.00

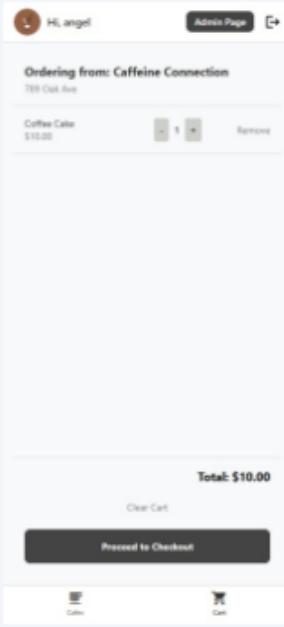
Добавете продукта, който искате да закупите. Това се осъществява в горния десен ъгъл на бутончето “+”.

4 Click "View Cart"



Когато добавите продукт, ще се появи поръчката ви в долния ъгъл на телефона ви. Показано е броят артикули тяхната цена и бутонче “View Cart”, цъкнете го, за да отидете на checkout.

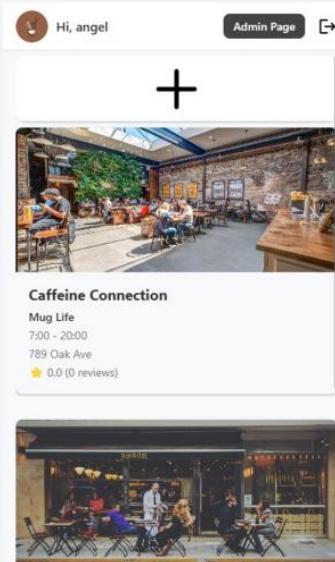
5 Click "Proceed to Checkout"



В checkout, ще видите продукта/ите, които сте избрали, тяхната цена и количеството. Може да променяте количеството с бутончетата “-”, “+”, съответно те са за декрементация и инкрементация. Ако не искате да правите поръчка, може да кликнете Clear Cart и да изчистите количката. Ако сте сигурни в поръчката си, цъкнете Proceed to checkout, за да финализирате вашата поръчка.

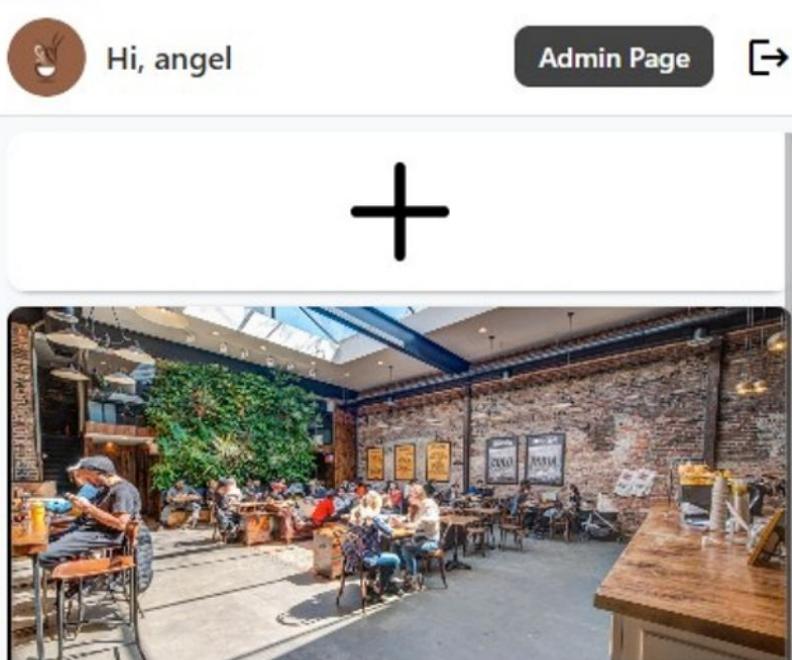
Обработка на поръчка:

- 1 Navigate to <http://localhost:8081/home/Cafes>



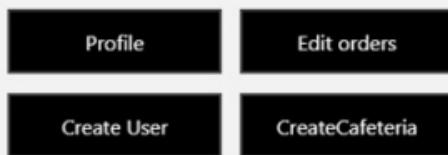
При правилно логване с акаунт, който разполага с permission, се показва бутона Admin Page. Ако нямате достъп, този бутон не се визуализира.

- 2 Click "Admin Page"



При натискането на бутона се отваря админското меню, което съдържа в себе си - Profile, Edit orders, Create User and CreateCafeteria.

3 Click "Edit orders"



Натискате Edit Orders и отивате в страницата, където са всички необработени поръчки.

Order #7	19/04/2025, 09:02:54	
● PROCESSING		
Processing	▼	
Coffee Cake	\$10.00 x 1	\$10.00
Total: \$10.00		

Order #8	19/04/2025, 09:03:11	
● PROCESSING		
Processing	▼	
Coffee Cake	\$10.00 x 1	\$10.00
Total: \$10.00		

Оттук можете да променяте статус на поръчките. Предоставена е информация за датата, име на продукта, цена, количество и общата сума. Промяната се осъществява чрез натискане на dropdown менюто. Избирате статуса и поръчката се премахва от необработени.

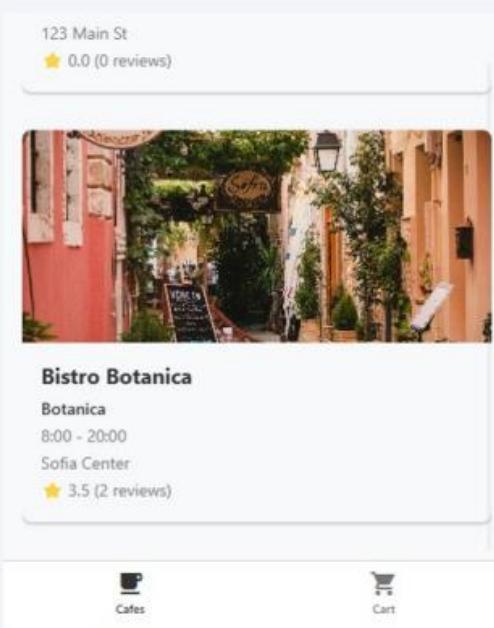
Оставяне на ревю:

- 1 Navigate to <http://localhost:8081/home/Cafes>

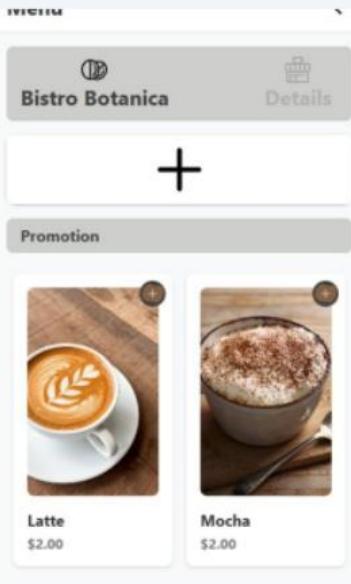


Селектирате кафене по ваш избор.

- 2 Select a cafeteria.



3 Click "Details"



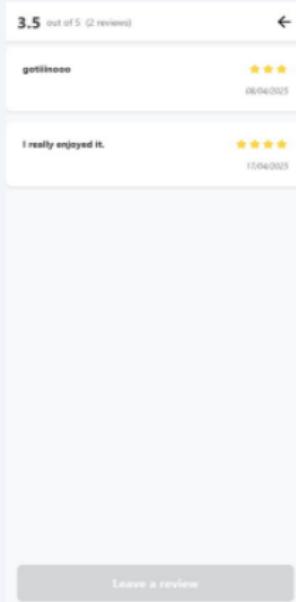
Кликвате на детайли. Там ще намерите подробна информация за кафенето, как да се свържете и оставите ревю.

4 Click on Leave a review



Кликвате на оставяне на ревю.

5 Click "Leave a review"



Тук може да видите броя ревюта, средноаритметичната оценка, както и самото ревю. Ако искате да оставите ревю кликнете на Leave a review.

6 Fill the form.

Rating:

★ ★ ★ ★ ★

Title:

Enter a title for your review (required)

Comment:

Write your review here... (optional)

Submit Review

Попълвате бланката като избирате броя звезди, заглавие на ревюто ви и коментар, ако искате.

7 Click the "Enter a title for your review (required)" field.

Rating:

★★★☆☆

Title:

Enter a title for your review (required)

Comment:

Write your review here... (optional)

Submit Review

8 Type a title

9 Click the "Write your review here... (optional)" field.

Rating:

★★★☆☆

Title:

Its good

Comment:

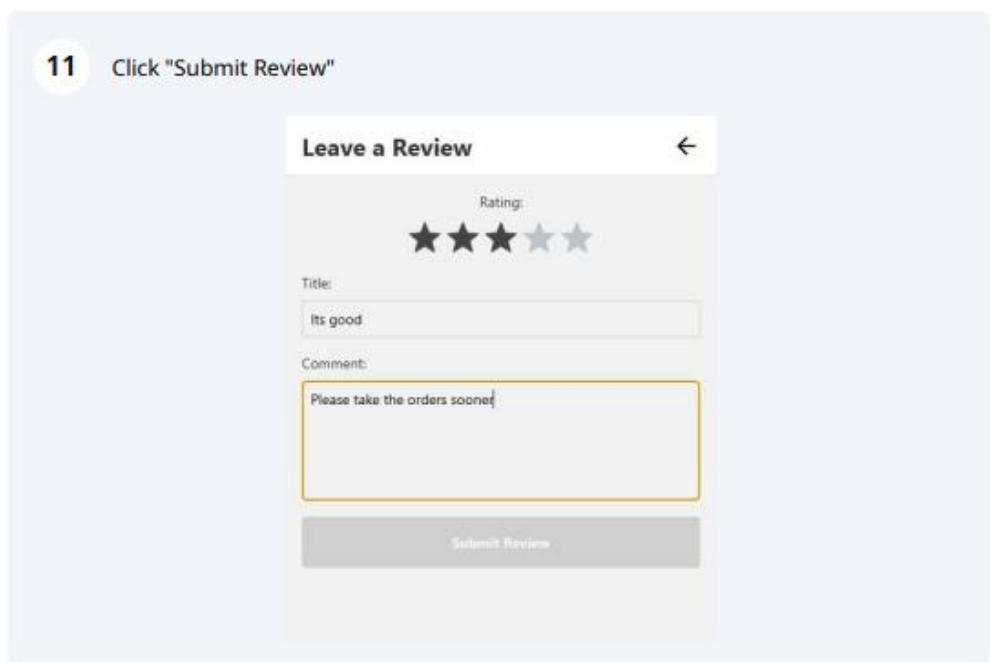
Write your review here... (optional)

Submit Review

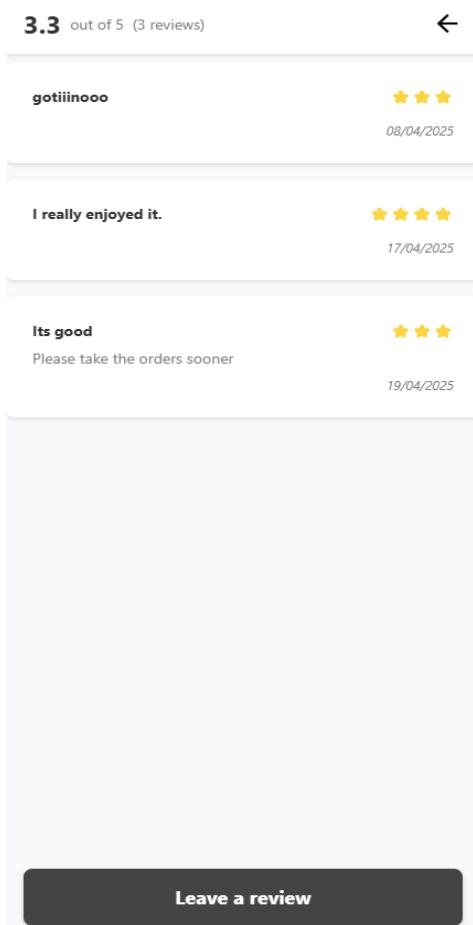
10 Type a feedback

Попълнете коментар, може да го пропуснете.

11 Click "Submit Review"



Цъкнете Submit Review, когато сте готови.



Ако правилно сте попълнили бланката, трябва да видите вашето ревю веднага.

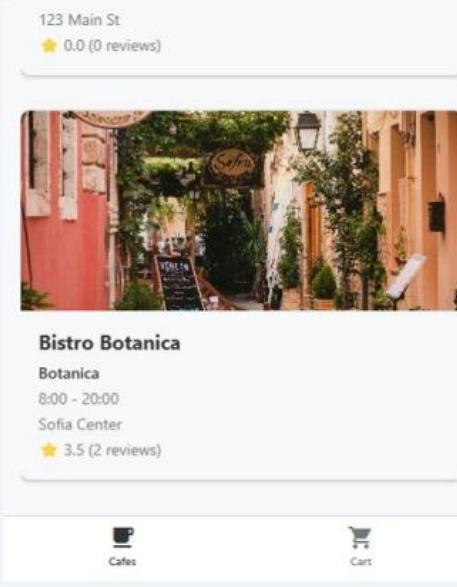
Свържи се с кафетерията:

- 1 Navigate to <http://localhost:8081/home/Cafes>



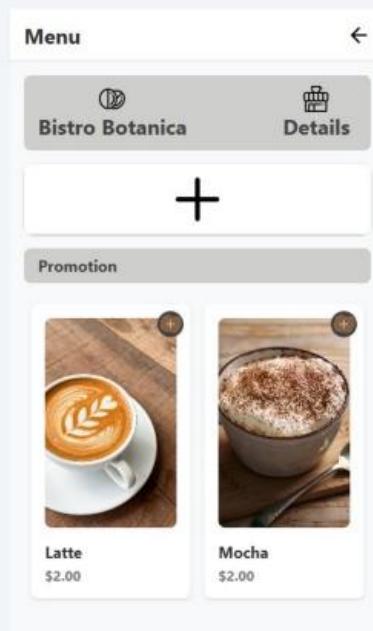
Отидете в страницата Кафенета.

- 2 Select a cafeteria.



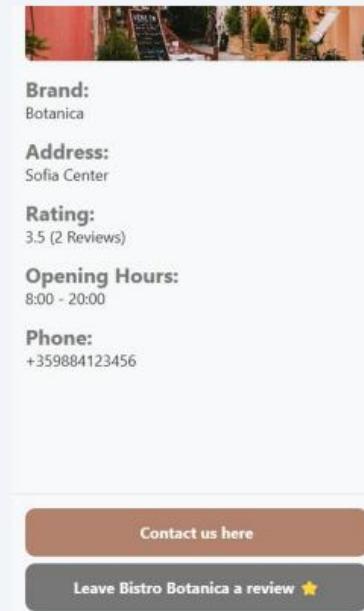
Изберете кафенето, с което искате да се свържете. Натиснете върху него.

3 Click details



Вие се намирате в менюто на самото кафене, за да се свържете с кафенето, трябва да натиснете бутона “Details”. Там ще намерите подробна информация.

4 Click "Contact us here"



В тази страница е подробно описана информацията за кафенето, като има и телефонен номер за връзка. Ако искате да се свържете по имейл, натиснете Contact us here.

- 5 Click the "Email*" field.

Contact Us

Email*

Name*

Your Message*

SUBMIT

- 6 Type your email.

Popълнете имейл адрес, то е задължително поле, за да направите връзка с кафенето. При липса или грешен имейл, кафенето няма да получи вашето съобщение.

- 7 Click the "Name*" field.

Contact Us

Email*

Name*

Your Message*

SUBMIT

- 8 Type your name

Попълнете името си или може да го пропуснете, ако желаете да останете анонимен.

- 9** Click the "Your Message*" field.

Contact Us

angelstoinov12@gmail.com

Angel

Your Message*

SUBMIT

- 10** Type your message

Остава да попълните вашето съобщение преди да можете да из pratите.

- 11** Click "Sending..."

Contact Us

angelstoinov12@gmail.com

Angel

There was a mistake in the price.

SUBMIT

Когато сте готови, натиснете Submit. Системата ще ви уведоми, ако има някакъв проблем с получаването на съобщението.

New message from Angel

C

User <cafedash25@gmail.com>
to codeninjastu ▾

You got a new message from Angel - angelstoinov12@gmail.com

Message:

There was a mistake in the price.

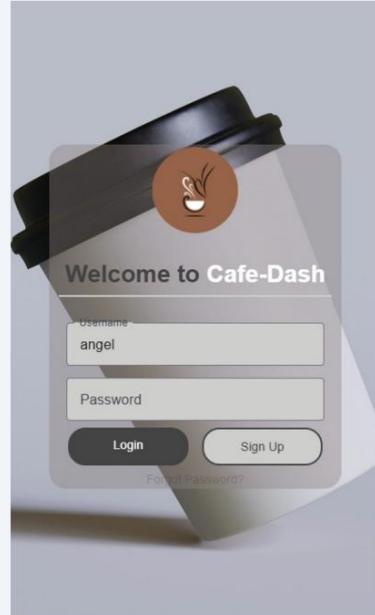
Reply to: angelstoinov12@gmail.com

Email sent via [EmailJS.com](#)

При успешно изпращане, кафенето получава вашето съобщение във този вид.

Промяна на паролата:

- 1 Navigate to <http://localhost:8081/login>



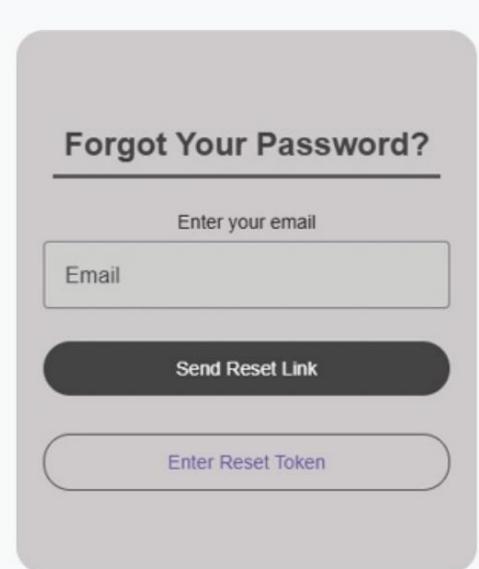
Ако сте забравили паролата си, Cafe-Dash предоставя опцията да я промените.

- 2 Натиснете "Forgot Password?"



За да промените паролата, натиснете бутона “Forgot Password”.

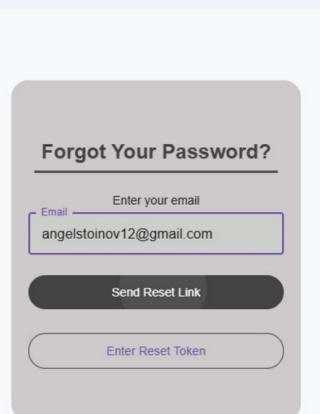
3 Въведи имейл адрес



The image shows a "Forgot Your Password?" form. At the top, it says "Forgot Your Password?". Below that is a placeholder "Enter your email". A text input field contains the email address "Email". Below the input field is a dark button labeled "Send Reset Link". At the bottom of the form is another button labeled "Enter Reset Token".

Въведете имейл адрес, с който сте се регистрирали.

4 Натисни "Send Reset Link". Ако вашият имейл адрес е валиден, вие ще получите имейл от нас.

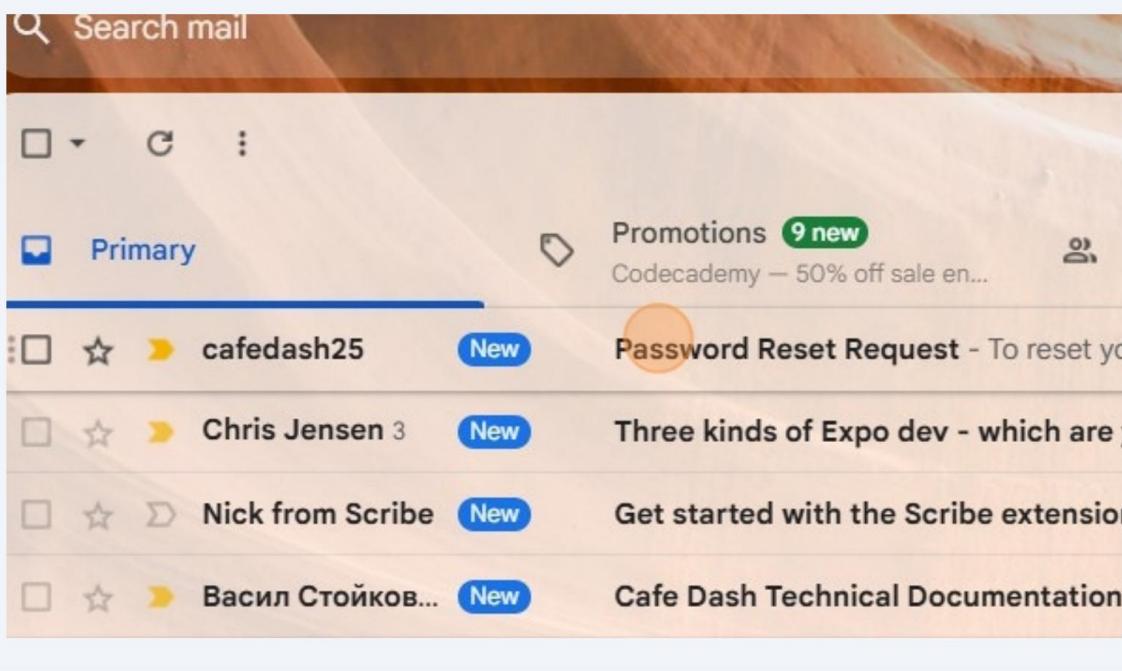


The image shows the same "Forgot Your Password?" form as above, but with the email input field populated with "Email angelstoinov12@gmail.com". The rest of the interface remains the same.

Системата винаги ви уведомява, че ако вашият имейл адрес е валиден, ще получите имейл от нас.

5

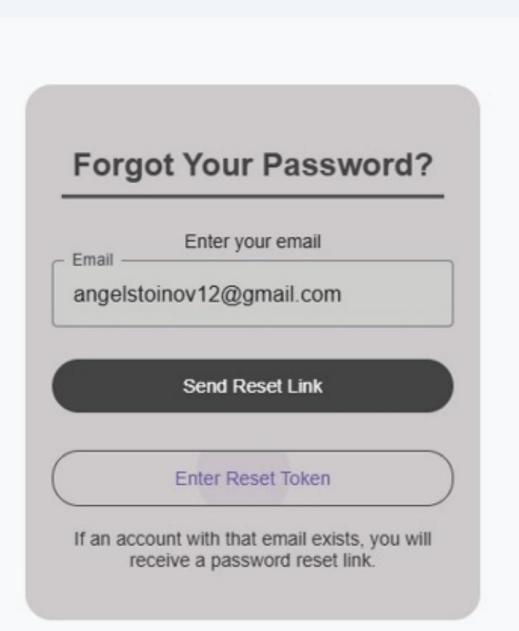
Отворите имейла, там ще намерите токен, който ще поставим обратно в приложението.



Имейлът, който да очаквате от нас е от адрес: cafedash25. **ВНИМАВАЙТЕ: всеки друг имейл адрес различен от този се представя за нас и не носим отговорност при хакерска атака.**

6

Натисните "Enter Reset Token"



Върнете се обратно в приложението, натиснете бутона Enter Reset Token, той ще ви отведе в друга страница.

7

Попълнете бланката. Поставете копирания токен, въведете нова парола два пъти, за да потвърдите.

Enter Reset Token and
New Password

Reset Token
43bda6b4-75ec-4e6e-b5e3-1b608c4f

New Password

Confirm Password

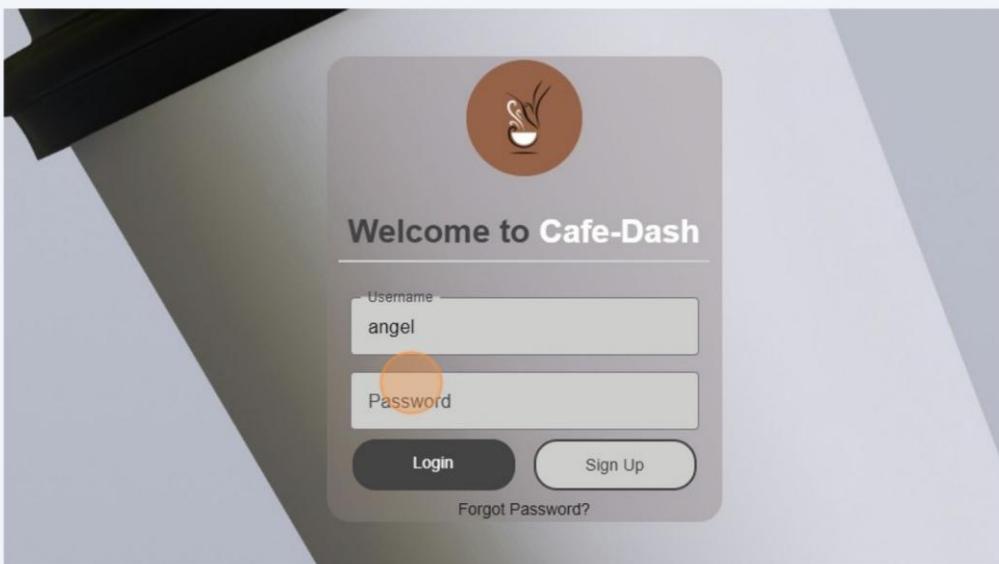
Reset Password

Password has been reset successfully.

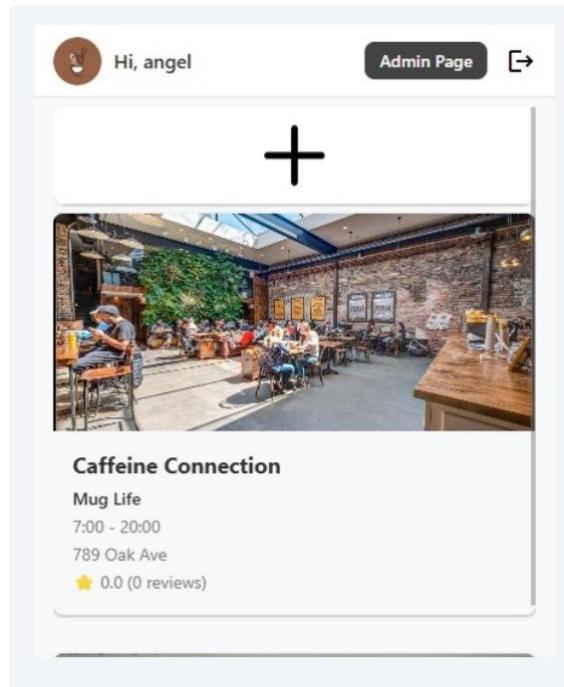
При правилен токен и два пъти правилна потвърдена парола. Ще видите съобщението, което е на екрана (отгоре). При грешен или изтекъл токен (30 мин продължителност) ще ви се изведе съобщение, че токенът е невалиден или съответно изтекъл. Паролата трябва да е поне 8 символа. Системата ще потвърди това, като ви даде обратна връзка. Също, ако паролата не съответства, ще получите специално съобщение, че паролите не са еднакви.

8

Попълнете вече логина с новата парола.



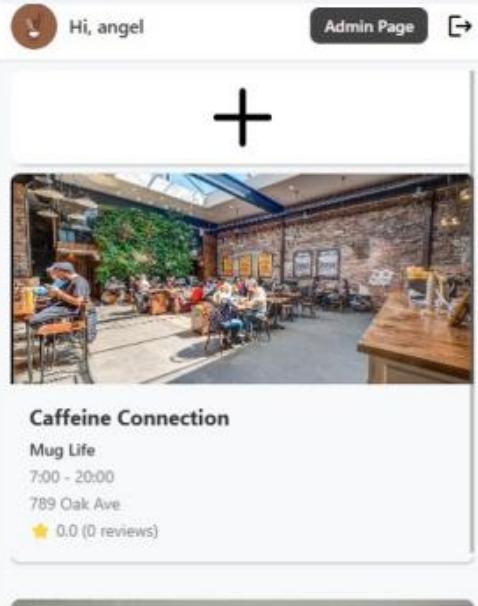
Системата ви изпраща обратно в логин менюто, където трябва да въведете новата парола.



Ако всичко е наред, системата ще ви изпрати в страницата с кафетерии. При грешка системата ще ви информира и ще останете в логин страницата, докато проблемът не бъде решен.

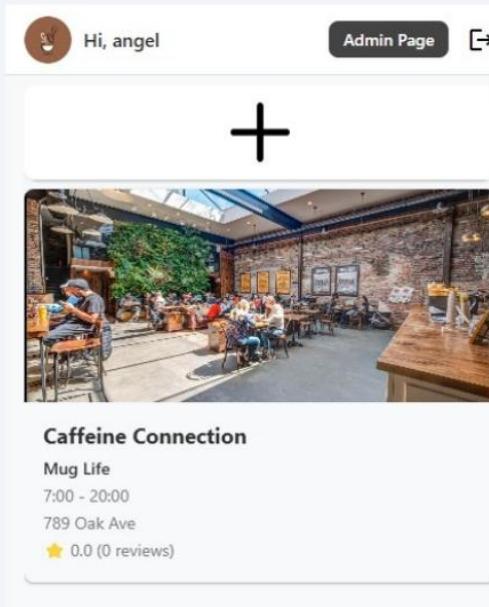
Добавяне на продукт в кафетерия:

- 1 Отидете на <http://localhost:8081/home/Cafes>



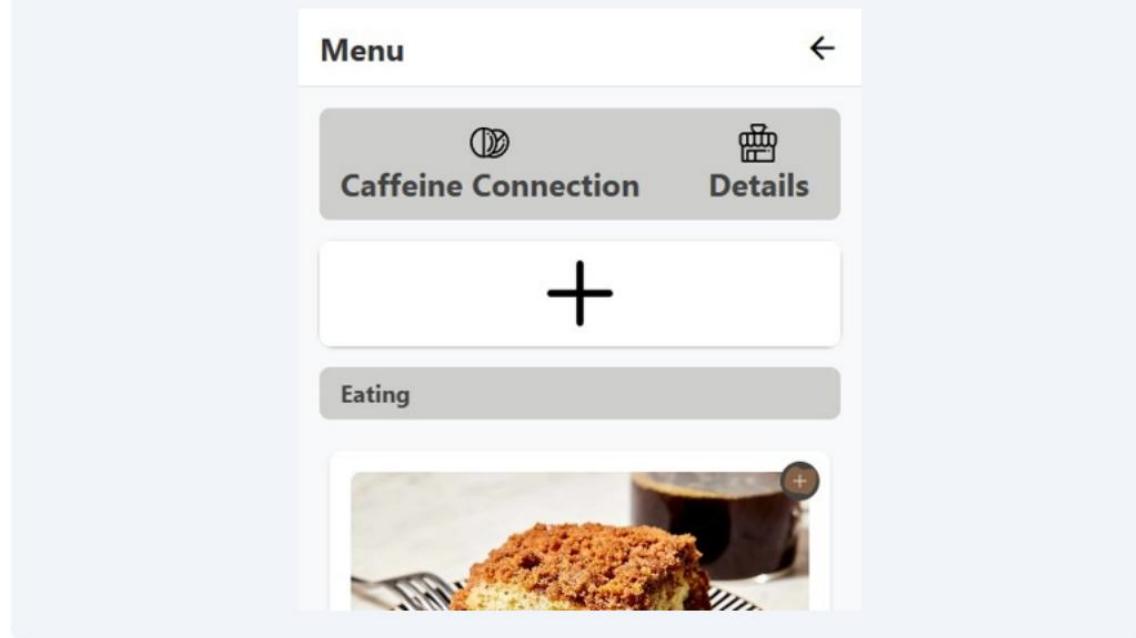
За да добавите нови продукти, първо трябва да сте логнати с авторизиран акаунт. Стигнете до кафетерия страницата.

- 2 Изберете кафетерия, в която искате да добавите продукт.



Натиснете кафетерията, към която искате да добавите продукта.

3 Кликнете на големия знак: плюс



При натискане на бутона +, ще ви се отвори нова форма, която трябва да попълните, за да добавите нов продукт.

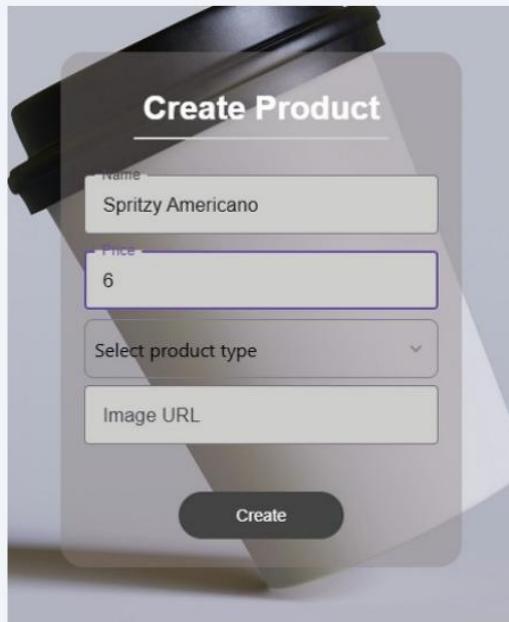
4 За да добавите продукт, трябва да попълните бланката.



5 Напишете името на продукта

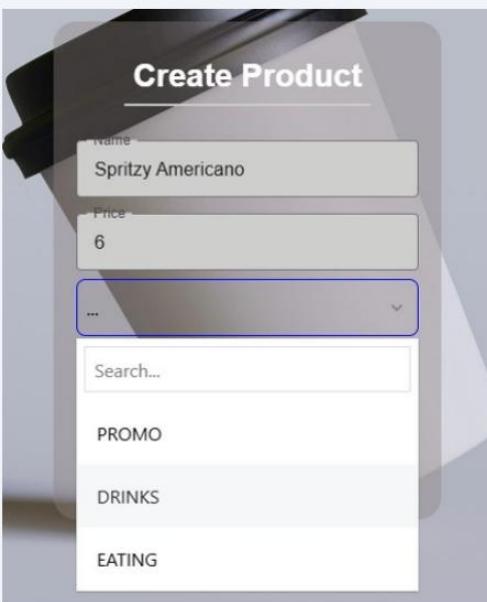
Всяко поле е задължително.

6 Въведете цена



Въведете положително число, ако въведете 0 или отрицателно, системата ще ви уведоми.

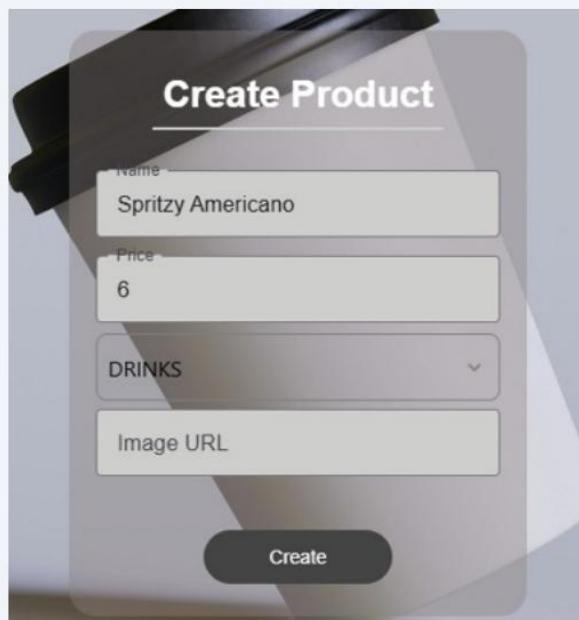
7 Изберете категория от dropdown менюто.



Ако не изберете нищо, системата ще ви уведоми, да го направите.

8

Поставете пътя на снимката



За да вземете адреса на снимката, трябва преди това да имате достъп до storage blob системата.

Spritzy Americano
\$6.00

Ако всичко е наред ще ви се визуализира добавеният продукт.

Добавяне на кафетерия:

- 1 Navigate to <http://localhost:8081/home/Cafes>



Трябва да сте логнати и да имате коректната авторизация.

- 2 Натиснете тук



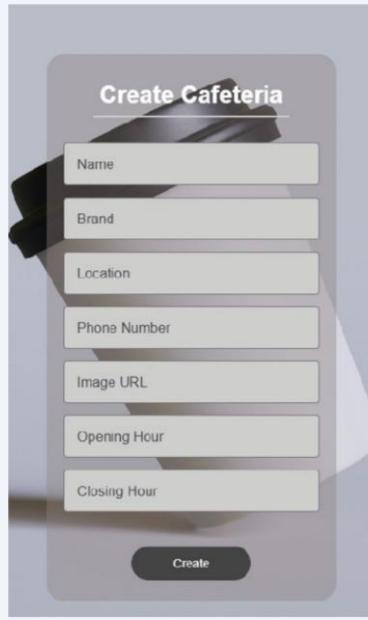
Hi, angel

Admin Page



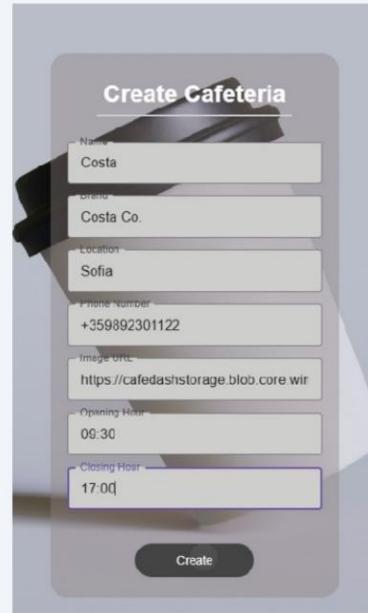
Натиснете бутона +, за да добавите нова кафетерия. Там ще трябва да попълните форма.

3 Попълнете формата



Всяко поле има валидация, която ще ви уведоми, ако сте допуснали грешка. Всяко поле е задължително. Вземете пътя на снимката от azure storage blob.

4 Натиснете "Creating..."



Натиснете бутона Create, ако сте готови. Системата ще ви уведоми, ако успешно се добавили кафетерия или не.



Costa

Costa Co.

9:30 - 17:00

Sofia

0.0 (0 reviews)

Новата кафетерия ще излезне при останалите, като средноаритметичният рейтинг и броя ревюта са 0 по default.

Принос и отговорност:

Ние, екипът на Cafe-Dash, взехме решението всеки един от нас да участва пълноценно във всички основни аспекти от разработката на приложението. Идеята на приложението бе да натрупаме нови знания по съставяне на full-stack приложение, както и работата с version control в екипна среда. Описаната документация е съставена от всеки един от нас.

В процеса на разработка:

Всеки член от екипа се включи в изграждането на потребителския интерфейс, използвайки React-Native за съставянето му.

Всеки от нас участва в разработката на сървърната логика, реализирана със Spring Boot, където се изградиха контролери, услуги и модели.

Работихме съвместно и при проектирането и реализирането на релационната база данни – създадохме ER диаграми, моделирахме таблиците в PostgreSQL и се погрижихме за коректната връзка между обектите.

Темата и целта на приложението е съгласувана с всеки един член от екипа. Основен UI/UX дизайнер е **Васил Стойков**. Валидацията, аутентикацията и цялостното security бе разработена от **Запрян Запрянов**. За Deployment частта отговаря **Ангел Стойнов**, както и за съхраняването на изображенията на azure storage.

Редно е да се спомене още веднъж, че всеки е работил както и по backend така и по frontend.

Бъдещи промени:

Това, което предстои в бъдещата разработка на приложението, е предимно неговото публикуване в Google Play с цел достигане до реални потребители и събиране на обратна връзка.

Необходими промени:

- Оптимизация на производителността – допълнително тестване на приложението в реални условия, премахване на излишни заявки и ненужно зареждане на компоненти.
- Добавяне на Unit и Integration тестове.
- Добавяне на Push notification.
- Добавяне на гео-локации.
- Добавяне на допълнителни методи за автентикация - чрез Google account.

След приключване на горепосочените дейности, приложението ще бъде подгответо за създаване на APK, подписване със сертификат и публикуване в Google Play Console, в съответствие с изискванията на платформата.

Заключение:

Разработката на информационната система „Cafe-Dash“ постигна основната си цел – създаване на цялостно решение за управление на кафетерии, което обединява интуитивен потребителски интерфейс, надеждна сървърна логика и добре организирана база от данни.

Приложението предоставя възможност на потребителите да правят поръчки, служителите да обработват поръчките, а администраторите и собствениците да управляват цялостния процес. Вярваме, че с малко допълнителни подобрения, системата ще бъде конкурентоспособна на съществуващите решения в сектора.

Github:

Repository - <https://github.com/Matrix2121/Cafe-Dash>

Използвана литература и материали:

- [ScribeHow](#)
- [EmailJs](#)
- [Figma](#)
- [Azure](#)
- [Spring Documentation](#)
- [Baeldung](#)
- [React Native documentation](#)
- [Expo documentation](#)