



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«МИРЭА – Российский технологический университет»

**РТУ МИРЭА**

**Институт информационных технологий (ИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**КУРСОВАЯ РАБОТА**

по дисциплине: Разработка серверных частей интернет-ресурсов

по профилю: Разработка программных продуктов и проектирование информационных систем

направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Веб-сервис доставки продуктов питания

Студент: Быченков Александр Константинович

Группа: ИКБО-32-21

Работа представлена к защите 05.03.24 (дата) А. Быченков / Быченков А.К.  
(подпись и ф.и.о. студента)

Руководитель: ст. преподаватель Волков Михаил Юрьевич

Работа допущена к защите 05.03.24 (дата) М. Волков / Волков М.Ю.  
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: хорошо

05.03.24 Волков М.Ю. / Волков М.Ю. /  
05.03.24 Волков М.Ю. / Волков М.Ю. /  
(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших  
защиту)



МИНОБРНАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА – Российский технологический университет»  
**РТУ МИРЭА**

**Институт информационных технологий (ИТ)**  
**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**ЗАДАНИЕ**  
**на выполнение курсовой работы**

по дисциплине: Разработка серверных частей интернет-ресурсов  
по профилю: Разработка программных продуктов и проектирование информационных систем  
направления профессиональной подготовки: Программная инженерия (09.03.04)

Студент: Быченков Александр Константинович

Группа: ИКБО-32-21

Срок представления к защите: 12.12.2023.

Руководитель: ст. преподаватель Волков Михаил Юрьевич

**Тема:** Веб-сервис доставки продуктов питания

**Исходные данные:** используемые технологии: HTML5, CSS3, JavaScript, Python, VS Code, SQL СУБД, наличие: межстраничной навигации, внешнего вида страниц, соответствующего современным стандартам веб-разработки, использование паттерна проектирования MVC. Нормативный документ: инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18.

**Перечень вопросов, подлежащих разработке, и обязательного графического материала:**

1. Провести анализ предметной области разрабатываемого веб-приложения. 2. Обосновать выбор технологий разработки веб-приложения. 3. Разработать архитектуру веб-приложения на основе выбранного паттерна проектирования. 4. Реализовать слой серверной логики веб-приложения с применением выбранной технологии. 5. Реализовать слой логики базы данных. 6. Разработать слой клиентского представления веб-приложения 7. Создать презентацию по выполненной курсовой работе.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: [подпись] /Р. Г. Болбаков/, « 18 » 09 2023 г.

Задание на КР выдал: [подпись] /М.Ю. Волков/, « 18 » 09 2023 г.

Задание на КР получил: [подпись] /А.К. Быченков/, « 18 » 09 2023 г.

Руководитель курсовой работы: старший преподаватель М.Ю. Волков

Быченков А.К., Курсовая работа направления подготовки «Программная инженерия» на тему «Веб-сервис доставки продуктов питания»: М. 2023 г., МИРЭА – Российский технологический университет (РТУ МИРЭА), Институт информационных технологий (ИИТ), кафедра инструментального и прикладного программного обеспечения (ИиППО) – 25 стр., 20 рис., 1 табл., 8 источн. (в т.ч. на английском яз.).

Ключевые слова: доставка продуктов питания, разработка клиент-серверного приложения, открытый протокол авторизации, контейнеризация веб-приложений, fullstack-разработка.

Целью работы является проектирование архитектуры веб-приложения на основе определенного паттерна проектирования, разработка клиентской и серверной логики веб-приложения. Разработана структура базы данных. Необходимые для функционирования системы веб-сервисы были развернуты в облаке.

Bychenkov A.K., Course work in the field of training "Software Engineering" on the topic "Food delivery web-service": M. 2023, MIREA – Russian Technological University (RTU MIREA), Institute of Information Technology (IIT), Department of Instrumental and Applied Software (IiPPO) – 25 p., 20 ill., 1 tabl., 8 ref. (in English).

Keywords: food delivery, client-server application development, open authorization protocol, web application containerization, fullstack development.

The aim of the work is to design the architecture of a web application based on a specific design pattern, develop client and server logic of a web application. The database structure has been developed. The web services necessary for the operation of the system have been deployed in the cloud.

РТУ МИРЭА: 119454, Москва, пр-т Вернадского, д. 78

кафедра инструментального и прикладного программного обеспечения (ИиППО)

Тираж: 1 экз. (на правах рукописи)

Файл: «ПЗ\_РСЧИР\_ИКБО-32-21\_БыченковАК.pdf», исполнитель Быченков А.К.

© А.К. Быченков

## СОДЕРЖАНИЕ

|   |    |
|---|----|
| ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ .....   | 5  |
| ВВЕДЕНИЕ .....  | 6  |
| 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ФОРМИРОВАНИЕ ОСНОВНЫХ<br>ТРЕБОВАНИЙ К ПРИЛОЖЕНИЮ .....                      | 7  |
| 2 ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЙ РАЗРАБОТКИ .....   | 11 |
| 2.1 ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, НЕОБХОДИМОЕ<br>ДЛЯ РАЗРАБОТКИ И ФУНКЦИОНИРОВАНИЯ ВЕБ-ПРИЛОЖЕНИЯ.. | 11 |
| 2.2 ЯЗЫКИ И ТЕХНОЛОГИИ РЕАЛИЗАЦИИ ВЕБ-ПРИЛОЖЕНИЯ .....  | 11 |
| 3 РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ .....   | 12 |
| 3.1 РАЗРАБОТКА АРХИТЕКТУРЫ ВЕБ-ПРИЛОЖЕНИЯ НА ОСНОВЕ<br>ВЫБРАННОГО ПАТТЕРНА ПРОЕКТИРОВАНИЯ .....           | 12 |
| ЗАКЛЮЧЕНИЕ .....  | 23 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....  | 24 |

## ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящем отчете применяются следующие сокращения и обозначения:

- |       |  |
|-------|--|
| OAuth | – Open Authorization (открытый протокол авторизации);  |
| OIDC  | – OpenID Connect (протокол аутентификации, основанный на протоколе OAuth2);  |
| JWT   | – JSON Web Token (открытый стандарт для создания токенов доступа, основанный на формате JSON);   |
| REST  | – Representational State Transfer (передача репрезентативного состояния);  |
| MVC   | – Model-View-Controller (схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер). |

## **ВВЕДЕНИЕ**

Развитие сферы услуг доставки продуктов питания непрерывно нарастает в контексте современных тенденций образа жизни. В условиях изменяющихся предпочтений потребителей и активного использования технологий возникает стремительная необходимость в создании эффективных веб-сервисов, обеспечивающих удобство, скорость и качество доставки.

Целью данной курсовой работы является разработка веб-сервиса по доставке продуктов питания, ориентированного на обеспечение клиентов возможностью удобного и оперативного заказа.

Актуальность данной работы определяется растущим спросом на услуги доставки продуктов питания среди потребителей, особенно в условиях изменений образа жизни, связанных с увеличением удаленной работы и ограничениями мобильности.

Объектом исследования является процесс разработки веб-сервиса доставки продуктов питания, а предметом - функциональность, архитектура и оптимизация процессов заказа и доставки через данный сервис.


Ожидаемый результат работы - создание работоспособного прототипа веб-сервиса, который будет демонстрировать эффективную работу механизмов заказа и доставки, удовлетворяя требованиям современных потребителей и обеспечивая оперативное выполнение заказов.

Для достижения поставленной цели будут рассмотрены современные технологии в сфере разработки веб-сервисов, а также анализированы основные аспекты, определяющие удовлетворенность клиентов услугами доставки продуктов питания.

## 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ФОРМИРОВАНИЕ ОСНОВНЫХ ТРЕБОВАНИЙ К ПРИЛОЖЕНИЮ

Для анализа предметной области было проведено сравнение двух популярных российских веб-сервисов доставки продуктов питания: СберМаркет (sbermarket.ru), Самокат (samokat.ru). Были выделены общие черты, функционал, который присутствует во всех сервисах.

Для возможности оформления заказа на каждом из сервисов клиентам необходимо выполнить вход: либо по номеру телефона с подтверждением по СМС, либо, используя Сбер ID OIDC, рисунки 1.1, 1.2. Так как отправка СМС сообщений платная, в разрабатываемом приложении будет использоваться вариант с OIDC, а именно identity-провайдер KeyCloak.



**Добро пожаловать**

Введите номер телефона, чтобы войти  
или зарегистрироваться

Номер телефона  
+7 ( ) \_ \_ - -

**Получить код в СМС**


 Войти по Сбер ID

Рисунок 1.1 – Окно авторизации на sbermarket.ru



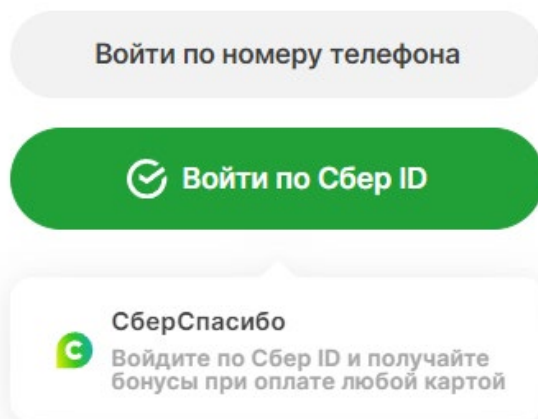


Рисунок 1.2 – Окно авторизации на samokat.ru

После авторизации клиентам показывается каталог с продуктами, разбитыми на категории, рисунок 1.3.

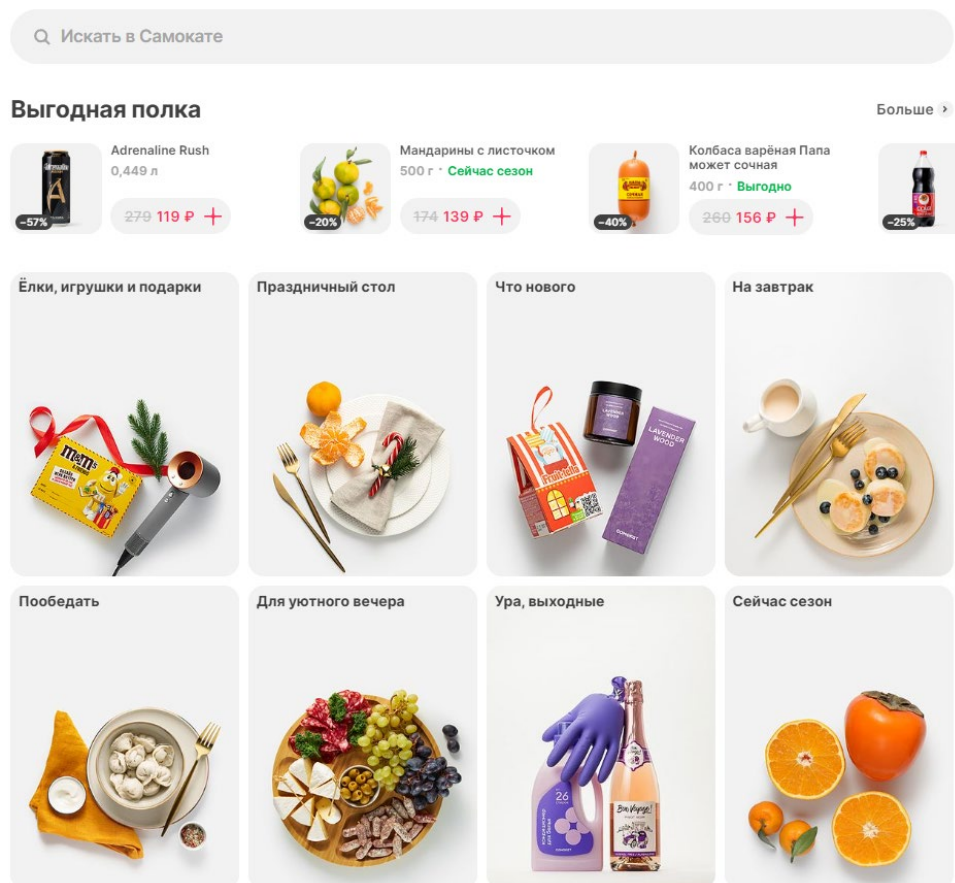


Рисунок 1.3 – Каталог продуктов на samokat.ru

При выборе определенного продукта отображается его краткое описание, рисунок 1.4.



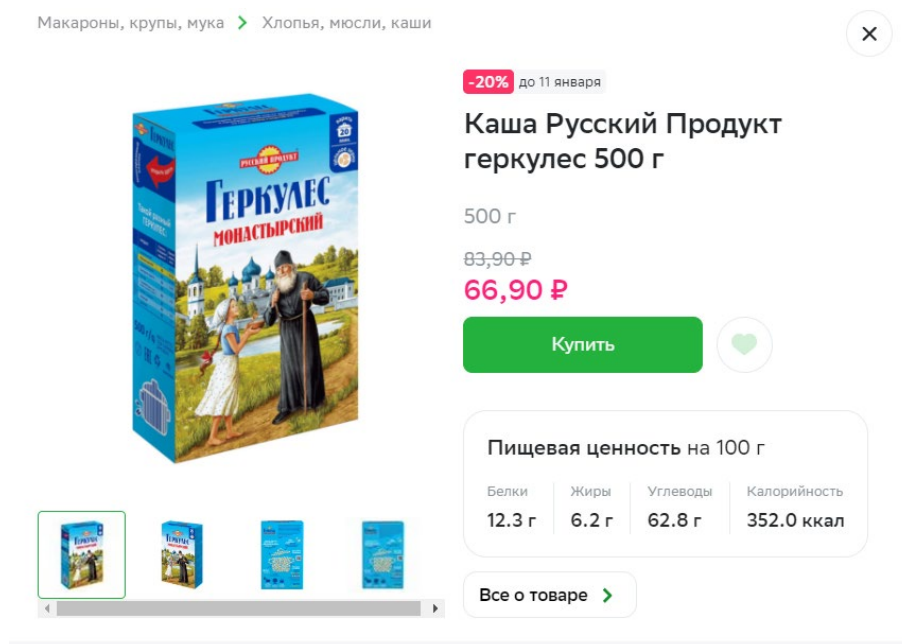


Рисунок 1.4 – Страница продукта на sbermarket.ru

Для оформления заказа на доставку нужные продукты необходимо добавить в корзину, рисунок 1.5.

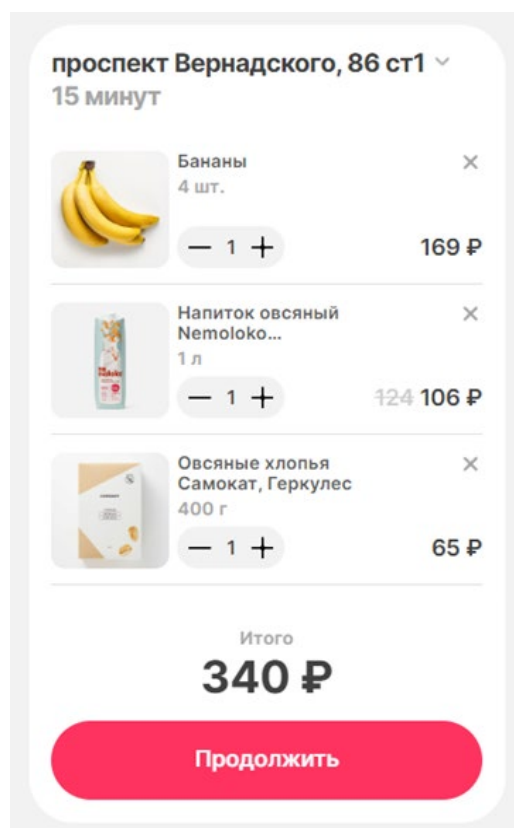


Рисунок 1.5 – Корзина с продуктами на samokat.ru

Для завершения оформления заказа требуется указать адрес доставки, рисунок 1.6.

← Адрес и оплата ×

проспект Вернадского, 86 ст1

Квартира/офис Этаж

Подъезд Домофон

Комментарий

+ Оплата новой картой >

---

Доставка 0 ₽

Итого 340 ₽

Оплатить

Рисунок 1.6 – Оформление заказа на samokat.ru

Изучив вышеперечисленные сервисы, были сформированы основные требования к разрабатываемому приложению:

1. Авторизация пользователей;
2. Просмотр каталога товаров;
3. Работа с корзиной;
4. Возможность оформления заказа с выбором адреса доставки.

## **2 ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЙ РАЗРАБОТКИ**

### **2.1 ПРИКЛАДНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, НЕОБХОДИМОЕ ДЛЯ РАЗРАБОТКИ И ФУНКЦИОНИРОВАНИЯ ВЕБ-ПРИЛОЖЕНИЯ**

Для написания программного кода использовалась среда разработки VS Code. Для функционирования приложения необходима программная платформа контейнеризации приложений Docker, которая в настоящее время является стандартом развертывания веб-приложений.

### **2.2 ЯЗЫКИ И ТЕХНОЛОГИИ РЕАЛИЗАЦИИ ВЕБ-ПРИЛОЖЕНИЯ**

Для разработки серверной логики веб-приложения используется язык программирования Python версии 3.12 в связке с фреймворком FastAPI. Данная связка технологий активно применяется в разработке микросервисов, благодаря своей гибкости и высокой скорости разработки.

Разработка клиентской логики веб-приложения проводилась с использованием языка TypeScript и веб-фреймворка Next.js. Рассмотренные ранее популярные веб-сервисы доставки продуктов питания используют именно этот фреймворк, т.к. он позволяет довольно быстро создавать сложные веб-приложения с серверным рендерингом и различными способами кэширования данных, оптимизации загрузки веб-страниц.

Данные приложения хранятся внутри СУБД PostgreSQL. СУБД PostgreSQL широко используется в настоящее время и является промышленным стандартом хранения данных.

Для авторизации пользователей используется KeyCloak – готовое SSO (Single Sign-On) решение, имеющее встроенную реализацию протокола OpenID Connect. KeyCloak активно применяется во многих российских компаниях, как надежный сервер авторизации.

## 3 РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ

### 3.1 РАЗРАБОТКА АРХИТЕКТУРЫ ВЕБ-ПРИЛОЖЕНИЯ НА ОСНОВЕ ВЫБРАННОГО ПАТТЕРНА ПРОЕКТИРОВАНИЯ

Серверная часть веб-приложения была разработана с использованием паттерна проектирования MVC, подразумевающего собой отделение бизнес-логики (модели) от её представления. На рисунке 3.1.1 показана структура FastAPI приложения. Имеются контроллеры (routers), которые обрабатывают входящие запросы и вызывают сервисный слой (services), рисунок 3.1.2. На сервисном слое создаются транзакции базы данных, внутри которых вызываются функции слоя репозитория (repositories).

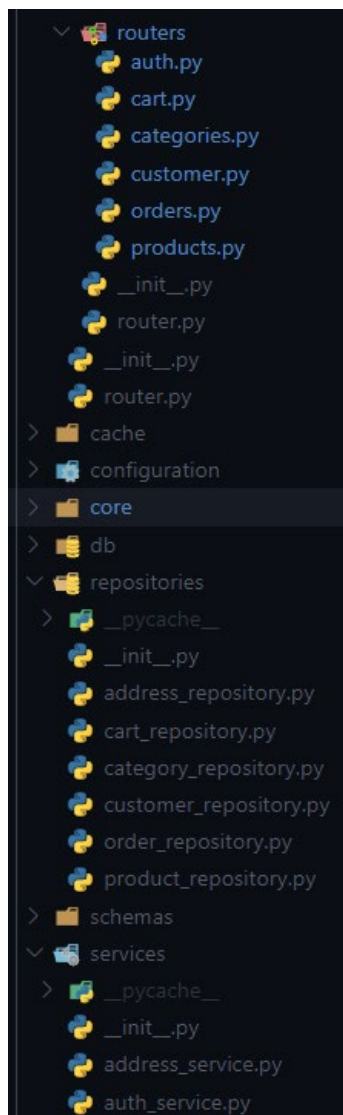


Рисунок 3.1.1 – Структура FastAPI приложения

```

class OrderService(Service):
    def __init__(
        self, repository: OrderRepository, cart_service: CartService, conn_pool: Pool
    ) -> None:
        self.repository: OrderRepository = repository
        self.cart_service: CartService = cart_service
        self.conn_pool: Pool[Any] = conn_pool

    @abstractmethod
    async def get_user_orders(self, user_id: UUID) -> list[OrderDTO]: ...

    @abstractmethod
    async def get_user_order_by_id(
        self, user_id: UUID, order_id: UUID
    ) -> Optional[OrderDTO]: ...

    @abstractmethod
    async def create_order_from_cart(self, user_id: UUID) -> Optional[OrderDTO]: ...

class OrderServiceImpl(OrderService):
    async def get_user_orders(self, user_id: UUID) -> list[OrderDTO]:
        async with self.conn_pool.acquire() as conn:
            return await self.repository.get_user_orders(user_id, conn)

```

Рисунок 3.1.2 – Сервис для работы с заказами

## 3.2 РЕАЛИЗАЦИЯ СЛОЯ СЕРВЕРНОЙ ЛОГИКИ ВЕБ-ПРИЛОЖЕНИЯ С ПРИМЕНЕНИЕМ ВЫБРАННОЙ ТЕХНОЛОГИИ

На рисунке 3.2.1 показан пример реализации HTTP-эндпоинтов для обработки входящих HTTP-запросов для работы с заказами. В них вызываются функции сервиса заказов, используя систему внедрения зависимостей. Показанные два HTTP-эндпоинта также требуют авторизации клиента по токenu доступа, указанному в заголовке Authorization, полученного от KeyCloak.

```

@router.get("/orders/{order_id}", name="Get user order by id")
async def get_user_order(
    order_id: UUID, user: User = Depends(get_user)
) -> OrderDTO | None:
    return await app_configuration.get_service(OrderService).get_user_order_by_id(
        user.id, order_id
    )

@router.post("/cart/create_order", name="Create order from cart")
async def create_order(user: User = Depends(get_user)) -> Optional[OrderDTO]:
    return await app_configuration.get_service(OrderService).create_order_from_cart(
        user.id
    )

```

Рисунок 3.2.1 – Контроллер для работы с заказами

Access и refresh токены можно получить либо, обратившись напрямую к сервису авторизации KeyCloak, либо через реализованную серверную часть. Второй вариант предпочтителен, так как более безопасен. На рисунке 3.2.2 показан метод сервиса аутентификации для обеспечения аутентификации пользователей по реквизитам для входа (логин и пароль или refresh token). Метод вызывает необходимые HTTP-эндпоинты самого KeyCloak и отправляет в теле запроса реквизиты, передаваемые пользователями. При успешной аутентификации сервис отдает пользователям access token, refresh token и время их действия.

```

async def signin(self, auth_request: AuthRequest) -> TokenResponse | None:
    async with AsyncClient() as client:
        response: Response = await client.post(
            f"{settings.keycloak.url}/realms/{settings.keycloak.realm}/protocol/openid-connect/token",
            data={
                "client_id": settings.keycloak.client_id,
                "client_secret": settings.keycloak.client_secret,
            }
        )
        if (
            {
                "grant_type": "refresh_token",
                "refresh_token": auth_request.refresh_token,
            }
            if auth_request.refresh_token
            else {
                "grant_type": "password",
                "username": auth_request.username,
                "password": auth_request.password,
            }
        ),
        match (response.status_code):
            case 200:
                return TokenResponse.model_validate(response.json())
            case 400:
                raise HTTPException(400, "Invalid request")
            case 401:
                raise HTTPException(401, "Invalid credentials")
    return None

```

Рисунок 3.2.2 – Реализованный сервис аутентификации

В таблице 1 показаны реализованные в серверной части HTTP-эндпоинты.

Таблица 1 — Реализованные HTTP-эндпоинты

| Аутентификация пользователей |      |   |                                 |
|------------------------------|------|---|---------------------------------|
| /auth/sign-in                | POST | username;<br>password;<br>refresh token | Аутентификация<br>пользователей |

Продолжение таблицы 1

|   |       |   |   |
|---|-------|---|---|
| /auth/sign-up   | POST  | username;<br>password                               | Регистрация<br>пользователей                                  |
| Работа с категориями продуктов (включает в себя информация обо всех<br>вложенных продуктах) |       |   |   |
| /categories/all_parent  | GET   | -   | Получение всех<br>корневых<br>категорий                       |
| /categories/{category_slug}   | GET   | -   | Получение<br>категории по ее<br>идентификатору                |
| Работы с каталогом продуктов  |       |   |   |
| /products/search?text={}  | GET   | -   | Поиск<br>продуктов по<br>строке в<br>названии или<br>описании |
| Работа с корзиной   |       |   |   |
| /cart   | GET   | Authorization<br>header                             | Получение<br>данных о<br>корзине                              |
| /cart/products  | POST  | Authorization<br>header;<br>product_id;<br>quantity | Добавление<br>товара в корзину<br>пользователя                |
| /cart/products/{product_id}   | PATCH | Authorization<br>header;<br>quantity;<br>is_active  | Изменение<br>количества и<br>состояния<br>товара в корзине    |



Продолжение таблицы 1

| Работа с заказами                |        |  |  |
|----------------------------------|--------|--|--|
| /orders                          | GET    | Authorization header                                     | Получение списка заказов                       |
| /orders/{order_id}               | GET    | Authorization header                                     | Получение заказа по идентификатору             |
| /cart/create_order               | POST   | Authorization header                                     | Создание заказа из содержимого корзины         |
| Работа с адресами                |        |  |  |
| /clean-address                   | POST   | address  | Валидация адресов, получение геолокации адреса |
| /delivery-addresses              | GET    | Authorization header                                     | Получение списка адресов доставки пользователя |
|                                  | POST   | Authorization header;<br>lame;<br>latitude;<br>longitude | Добавление нового адреса доставки              |
| /delivery-addresses/{address_id} | DELETE | Authorization header                                     | Удаление сохраненного адреса доставки          |

Окончание таблицы 1

| Работа с пользователями |     |                      |                               |
|-------------------------|-----|----------------------|-------------------------------|
| /customer               | GET | Authorization header | Получение данных пользователя |

### 3.3 РЕАЛИЗАЦИЯ СЛОЯ ЛОГИКИ БАЗЫ ДАННЫХ

На рисунке 3.3.1 показана разработанная для работы веб-приложения схема базы данных.



Рисунок 3.3.1 – Разработанная схема базы данных

В репозиториях серверной части над таблицами базы данных выполняются определенные SQL-запросы. На рисунке 3.3.2 показан один из методов репозитория заказов для получения списка заказов пользователя. Используются JSON-функции агрегации и операторы JOIN для получения данных о заказах вместе с необходимой информацией о товарах, полученной из смежных таблиц. Полученный JSON-объект затем валидируется и преобразуется к определенному Python-классу.

```

class OrderRepositoryImpl(OrderRepository):
    async def get_user_orders(
        self,
        user_id: UUID,
        conn: PoolConnectionProxy,
    ) -> list[OrderDTO]:
        query = "SELECT JSON_AGG( \
            JSON_BUILD_OBJECT( \
                'product', \
                JSON_BUILD_OBJECT( \
                    'id', p.id, \
                    'title', p.title, \
                    'price', p.price, \
                    'description', p.description \
                ), \
                'quantity', op.quantity \
            ) \
        ) AS order_info, \
        SUM(op.price * op.quantity) AS total_price, \
        o.id, \
        o.user_id, \
        o.created_at, \
        o.status, \
        o.delivery_address \
        FROM order_product AS op \
        LEFT JOIN product AS p ON p.id = op.product_id \
        LEFT JOIN orders AS o ON o.id = op.order_id \
        WHERE o.user_id = $1 \
        GROUP BY op.order_id, o.id, o.user_id, o.created_at, o.status, o.delivery_address"
        results: list[Record] = await conn.fetch(query, user_id)
        return [
            OrderDTO.model_validate(
                f

```

Рисунок 3.3.2 – Репозиторий для работы с заказами

На рисунке 3.3.3 показаны реализованные Python-классы для передачи данных о заказах, включающие необходимые поля и их типы, рисунок 3.3.3.

```

class OrderDB(BaseModel):
    id: UUID
    user_id: UUID
    delivery_address: str
    total_price: Decimal
    status: ORDER_STATUS
    created_at: datetime

class OrderDTO(OrderDB):
    products: list[OrderProductDTO]

```

Рисунок 3.3.3 – Python-классы для валидации данных из БД для их дальнейшей передачи

## 3.4 РАЗРАБОТКА СЛОЯ КЛИЕНТСКОГО ПРЕДСТАВЛЕНИЯ ВЕБ-ПРИЛОЖЕНИЯ

На рисунке 3.4.1 показан внешний вид клиентского представления веб-приложения.

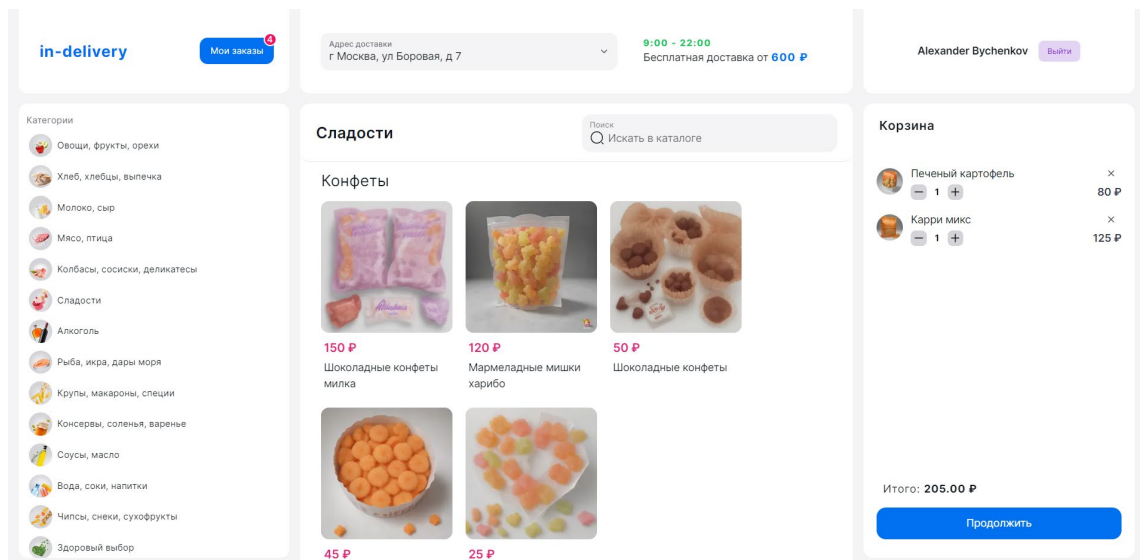


Рисунок 3.4.1 – Внешний вид веб-приложения

Клиентская часть реализована с использованием фреймворка React. Стили компонентов написаны с использованием CSS-фреймворка Tailwind CSS. На рисунке 3.4.2 показана реализация модального окна продукта.

```
const ProductModal = () => {
  const modalProduct = useStore($modalProduct);
  const cartData = useStore($cart);
  return (
    <Modal
      isOpen={modalProduct ? true : false}
      onClose={() => setModalProduct(null)}
      scrollBehavior="inside"
      placement="center"
      className="max-h-[90vh] overflow-hidden"
      classNames={{
        closeButton: "absolute top-4 right-4 bg-gray-200 hover:bg-gray-300"
      }}
    >
      {modalProduct && (
        <ModalContent>
          <ModalBody className="p-0">
            <Image
              className="z-[-1] rounded-none"
              quality={100}
              src={modalProduct.image}
            />
          </ModalBody>
        </ModalContent>
      )}
    </Modal>
  )
}
```

Рисунок 3.4.2 – Реализация модального окна продукта

Данные о выбранном продукте и корзине берутся из глобального состояния с использованием менеджера состояний *effector* и отображаются на странице, рисунок 3.4.3.

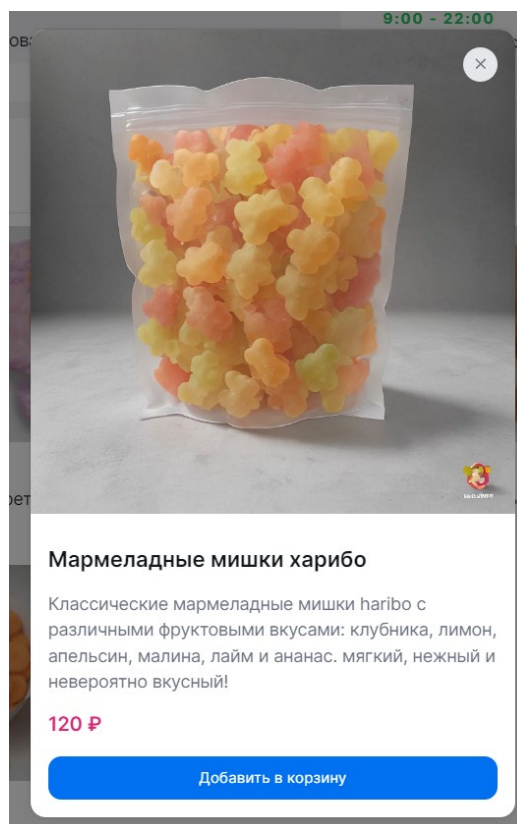


Рисунок 3.4.3 – Модальное окно продукта

Для возможности оформления заказа пользователю необходимо войти в учетную запись. Вход происходит через сервис авторизации KeyCloak, рисунок 3.4.4.

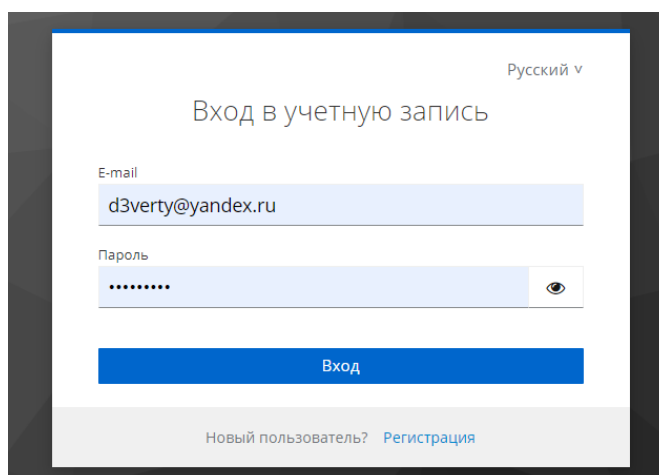


Рисунок 3.4.4 – Вход в учетную запись

После входа пользователь получает возможность работать с корзиной. На рисунке 3.4.5 показана корзина с добавленными продуктами. Продукты можно удалять или изменять их количество.

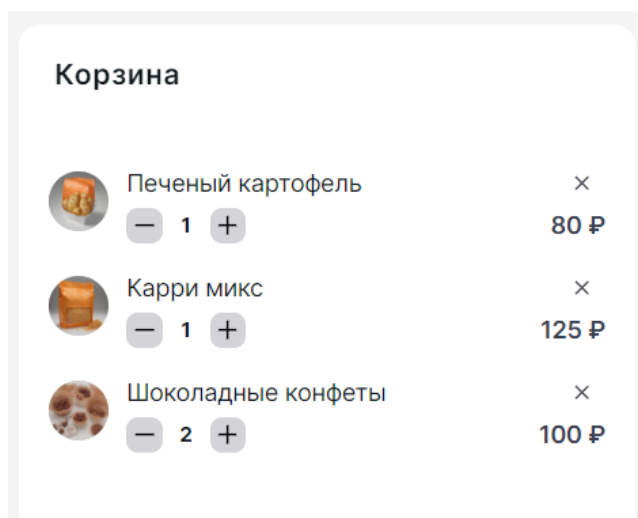


Рисунок 3.4.4 – Корзина с продуктами

Для оформления заказа необходимо ввести адрес доставки. При вводе показываются варианты автозаполнения. Для введенного адреса показывается точка на карте, рисунок 3.4.5

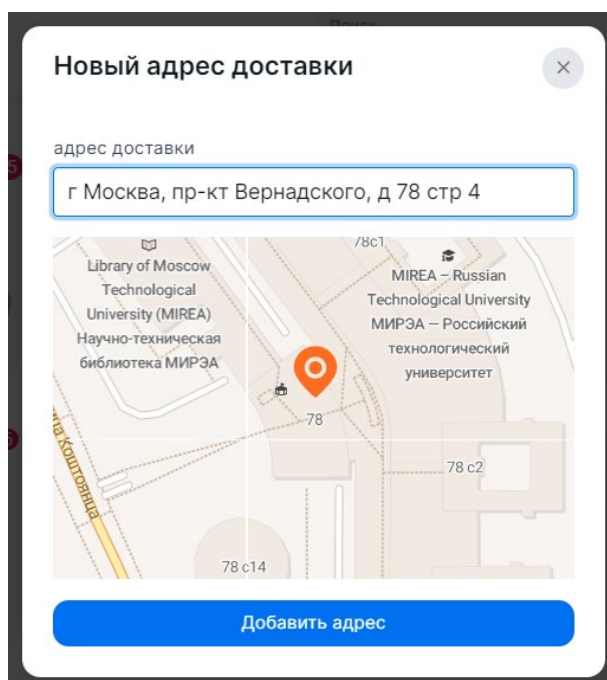


Рисунок 3.4.5 – Добавление нового адреса доставки

После оформления заказа информацию о нем можно просмотреть, нажав на кнопку «Мои заказы» и, выбрав соответствующий заказ. Пример информации о заказе показан на рисунке 3.4.6.

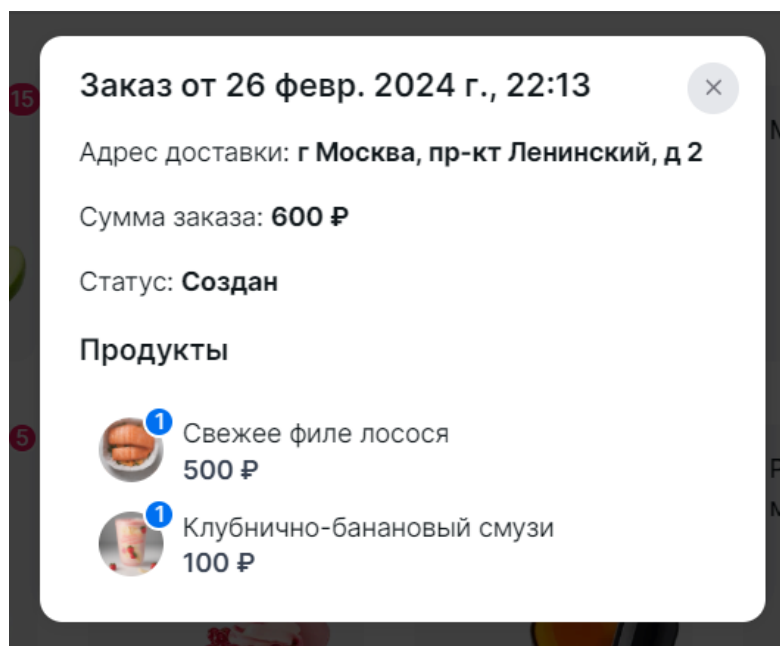


Рисунок 3.4.6 – Информация о заказе



## ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы были разработаны клиентская и серверная части веб-сервиса доставки продуктов питания, используя актуальные подходы к разработке современных веб-сервисов.

Средствами языка Python и веб-фреймворка FastAPI, применив соответствующие паттерны проектирования, была разработана серверная часть рассматриваемого веб-сервиса.

Веб-приложение было успешно развернуто в облаке, используя возможности контейнеризации.

Курсовая работа позволила на практике изучить востребованные веб-технологии для разработки современных веб-сервисов. В результате поставленные задачи были выполнены, все цели курсовой работы были достигнуты.

С клиентской и серверной частями разработанного веб-приложения можно ознакомиться в соответствующих GitHub-репозиториях:

<https://github.com/descenty/in-delivery-backend>

<https://github.com/descenty/in-delivery-frontend>

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация FastAPI [Электронный ресурс]. – URL: <https://fastapi.tiangolo.com/ru/> (дата обращения 20.02.2024).
2. Документация Next.js [Электронный ресурс]. – URL: <https://nextjs.org/docs> (дата обращения 20.02.2024).
3. Документация React [Электронный ресурс]. – URL: <https://react.dev/learn> (дата обращения 20.02.2024).
4. Документация Tailwind CSS [Электронный ресурс]. – URL: <https://tailwindcss.ru/docs/installation/> (дата обращения 20.02.2024).
5. Документация KeyCloak [Электронный ресурс]. – URL: <https://www.keycloak.org/documentation> (дата обращения 20.02.2024).
6. JSON-функции и операторы PostgreSQL [Электронный ресурс]. – URL: <https://www.postgresql.org/docs/9.3/functions-json.html> (дата обращения 20.02.2024).
7. Документация Effector [Электронный ресурс]. – URL: <https://docs.spring.io/spring-framework/docs/current/reference/html/web.html> (дата обращения 20.02.2024).
8. Документация NextUI [Электронный ресурс]. – URL: <https://nextui.org/docs> (дата обращения: 20.02.2024)