Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное   
образовательное учреждение высшего образования   
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе №0

Дисциплина: «Технологии сетевого программирования»

Выполнил: Кузьмин Д.О.

Группа: 6302-010302D

Самара, 2025

**Задание на лабораторную работу**

1. **Проектирование архитектуры**:
   * Разработать схему взаимодействия компонентов (сервер, БД, клиент);
   * Разработать логическую схему базы данных (использование, как минимум, 3-х сущностей и 1-ой связи many-to-many);
   * Определить структуру API (REST, методы, URL, параметры, форматы запросов и ответов).
2. **Создание Git-репозитория**:
   * Инициализировать репозиторий (рекомендуется использовать GitHub);
   * Настроить базовую структуру проекта.
   * Добавить .gitignore для исключения ненужных файлов (виртуальное окружение, логи, артефакты сборки и т. д.).

**Проектирование архитектуры и концепция приложения**

Приложение предназначено для продажи автозапчастей и управления заказами. Пользователи смогут просматривать каталог товаров, добавлять их в корзину, оформлять заказы, а администраторы — управлять товарами, заказами и клиентами.

**Для пользователей:**

Просмотр каталога автозапчастей  
Поиск и фильтрация товаров  
Добавление товаров в корзину  
Оформление заказа  
Личный кабинет (история заказов, управление профилем)  
Авторизация и регистрация

**Для администраторов:**

Добавление, редактирование, удаление товаров  
Управление категориями товаров  
Просмотр и обработка заказов  
Управление пользователями (список клиентов, их заказы)

**Клиент (Frontend) ⟷ Сервер (Backend) ⟷ База данных (Database)**

1. **Клиент (Frontend)**

Отправляет запросы к серверу через REST API

Получает данные и отображает их

Написан на **React**

1. **Сервер (Backend)**

Обрабатывает запросы от клиента

Взаимодействует с базой данных

Написан на **Node.js (Express)**

1. **База данных (Database)**

Хранит информацию о пользователях, услугах, товарах, заказах

Использует **PostgreSQL**

**Клиентская часть** будет взаимодействовать с сервером через REST API.

**Сервер** обработает запросы.

**База данных** -хранит данные и предоставляет необходимую информацию клиенту (админу).

**Стек технологий:**

Будет использоваться популярный стек технологий PERN.

Стек технологий: Frontend

* 1. React JS
  2. React bootstrap, чтобы не тратить время на верстку страниц (возможно будет сделана вертка без этого, либо переделана, для получения навыков)
  3. Axios - — клиент-серверная библиотека для выполнения HTTP-запросов для браузера и Node.js
  4. React-router-dom

Стек технологий: Backend

* 1. NODE JS
  2. Express
  3. PostgreSQL
  4. Sequelize для запросов к БД (чтобы не писать select напрямую к бд, а мы вызывали необходимые функции)

Nodejs (платформа для написания бэкенда на js), в связке с express (фреймворк для написания бэкенда на node)

**Логическая схема базы данных**

1. **User (Пользователь)**
   * id (PK)
   * name (Имя пользователя)
   * email
   * password\_hash (Хеш пароля)
   * phone\_number (Телефон покупателя)
   * role — роль пользователя (клиент или администратор).
2. **Product (Товар)**
   * id (PK)
   * name (Название товара)
   * description (Описание товара, которое может содержать дополнительную информацию)
   * price (Цена товара в рублях)
   * stock (Количество товара на складе.)
   * category\_id (FK) — идентификатор категории товара.
3. **Categories (Категория товара)**

* category\_id (PK) — уникальный идентификатор категории.
* category\_name — название категории (например, "Тормоза", "Двигатели" и т.д.).

1. **Order (Заказ)**
   * id (PK)
   * user\_id (FK → User.id)
   * total\_price
   * status (Статус заказа: 'pending', 'shipped', 'delivered' и т. д.)
2. **Cart (Связь M:Mмежду Order и Product)**
   * id (PK)
   * order\_id (FK → Order.id)
   * product\_id (FK → Product.id)
   * quantity

**Связи:**

User → Order: Один пользователь может сделать несколько заказов (связь "один ко многим").

Product ↔ Categories: связь "один ко многим" (одна категория может включать много товаров, но каждый товар относится к одной категории).

Order → Cart: Один заказ может включать несколько товаров, а каждый товар может быть в нескольких заказах (связь many-to-many через таблицу Cart).

Product → Cart: Один товар может быть добавлен в несколько заказов, и каждый заказ может включать несколько товаров (связь many-to-many через таблицу Cart).

Order и Product находятся в связи M:M через таблицу Cart. Это позволяет одному заказу включать несколько товаров и одному товару быть включенным в несколько заказов, с указанием количества каждого товара в корзине.

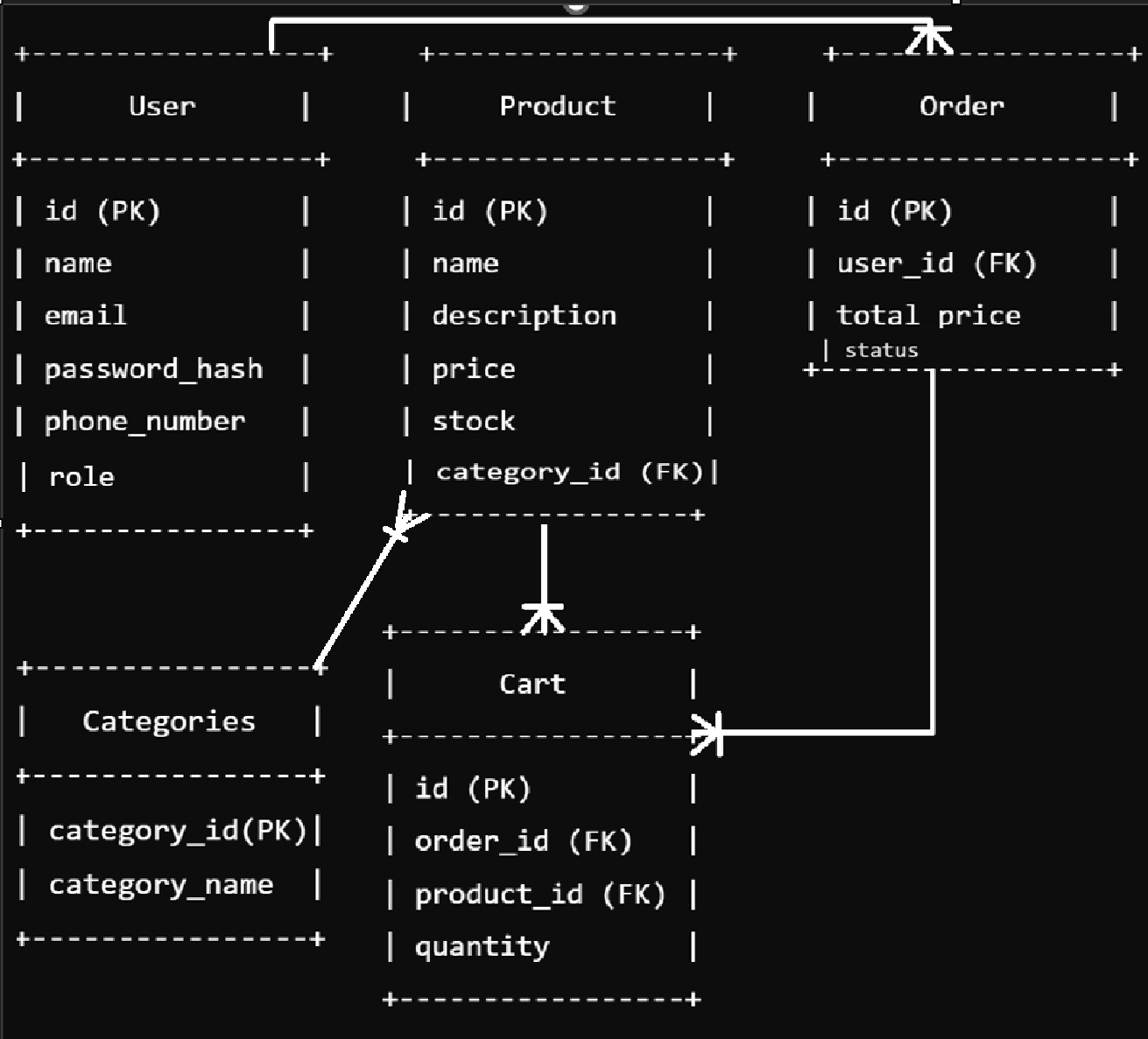


Рисунок 1 – Схема БД

**Структура API (REST)**

**Авторизация**

* POST /auth/register – регистрация
* POST /auth/login – вход (возвращает токен)
* GET /auth/me – проверка авторизации

**Каталог товаров**

* GET /products – получить список товаров
* GET /products/{id} – получить информацию о товаре
* POST /products – добавить новый товар (для админа)
* PUT /products/{id} – обновить товар (для админа)
* DELETE /products/{id} – удалить товар (для админа)

**Корзина и заказы**

* POST /cart – добавить товар в корзину
* GET /cart – получить содержимое корзины
* POST /order – оформить заказ
* GET /orders – получить список заказов пользователя
* GET /orders/{id} – получить информацию о заказе

**Формат запросов и ответов (JSON)**

Пример запроса на создание заказа:

POST /order

{

"products": [

{ "product\_id": 1, "quantity": 2 },

{ "product\_id": 3, "quantity": 1 }

]

}

Пример ответа:

{

"order\_id": 5,

"status": "new",

"total\_price": 2500

}