**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

на разработку

\_\_\_\_\_\_\_\_ информационной системы магазина по продаже \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ программного обеспечения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО |  | СОГЛАСОВАНО |
| Колледж ВятГУ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Потапов А.А.  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.  М.П. |  | Руководитель УП  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Долженкова М. Л.  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. |

|  |
| --- |
| СОГЛАСОВАНО |
| Колледж ВятГУ  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фалалеев Д.С.  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.  М.П. |

2024

**Содержание**

[1. Уточнение структуры данных 3](#_Toc182625507)

[1.1. Связи между сущностями 4](#_Toc182625508)

[1.2. Целостность сущностей 5](#_Toc182625509)

[1.3. ER диаграммы. 8](#_Toc182625510)

[2. Алгоритмы решения задачи 9](#_Toc182625511)

[3. Формы представления данных. 17](#_Toc182625512)

[4. Структура программы 23](#_Toc182625513)

[4.1. Бэкенд (FastAPI) 23](#_Toc182625514)

[4.2. Фронтенд (Nuxt.js SPA) 23](#_Toc182625515)

[4.3. Взаимодействие между FastAPI и Nuxt.js. 24](#_Toc182625516)

[4.4. Технические особенности 25](#_Toc182625517)

[5. Требования к техническим средствам 27](#_Toc182625518)

# Уточнение структуры данных

У веб-приложения должна быть база данных. База данных изображена в логической ER диаграмме и физической ER диаграмме на рисунках 1 и 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Сущность | Свойство | Тип | Уникальность | Обязательность заполнения | Ограничения |
| User(пользователь) | User ID | Number | Да | Да | -первичный ключ |
|  | Full name | varchar(50) | Нет | Да |  |
|  | Phone number | varchar(12) | Да | Да |  |
|  | E-mail | varchar(50) | Да | Да |  |
|  | Login | varchar(50) | Да | Да |  |
|  | Password | varchar(50) | Нет | Да |  |
|  | ISAdmin | bool | Нет | Да |  |
| Сategories(категории) | Сategory ID | Number | Да | Да | -первичный ключ |
|  | Name | varchar(50) | Да | Да |  |
|  | Description | text | Нет | Да |  |
| Goods(товары) | Good ID | Number | Да | Да | -первичный ключ |
|  | Name | varchar(50) | Да | Да |  |
|  | Сategory | Number | Нет | Да | -внешний ключ |
|  | Price | real | Нет | Да |  |
|  | Image | varchar(50) | Да | Да |  |
|  | Description | text | Нет | Нет |  |
| Order(заказ) | Order ID | Number | Да | Да | -первичный ключ |
|  | User | Number | Нет | Да | -внешний ключ |
|  | Good | Number | Нет | Да | -внешний ключ |
|  | Count | int | Нет | Да |  |
| Payment(оплата) | Payment | Number | Да | Да | -первичный ключ |
|  | Order | Number | Нет | Да | -внешний ключ |
|  | Payment metod | varchar(50) | Нет | Да |  |
|  | Paid | bool | Нет | Да |  |
|  | Amount | real | Нет | Да |  |
|  | Date | date | Нет | Да |  |

## Связи между сущностями

− «Категории» – «Товары»: один ко многим, в категории может быть множество товаров, но каждый товар в одной категории;

− «Товары» – «Заказ»: многие к одному, в заказе может быть множество товаров;

− «Оплата» – «Заказ»: один к одному, каждому заказу соответствует одна оплата и наоборот;

− «Клиент» – «Заказ»: многие к одному, каждому заказу соответствует один клиент и у клиента может быть много заказов;

Целостность ссылок:

User (пользователь):

User ID: Первичный ключ, который должен быть уникальным для каждого пользователя, обеспечивая целостность ссылок с другими таблицами, где он может использоваться в качестве внешнего ключа (например, в таблице Order).

Categories (категории):

Category ID: Первичный ключ, который также должен быть уникальным для каждой категории, обеспечивая целостность ссылок с другими таблицами, где он может использоваться в качестве внешнего ключа (например, в таблице Goods).

Goods (товары):

Good ID: Первичный ключ, обеспечивающий уникальность товара в таблице.

Category: Внешний ключ, связанный с Category ID в таблице Categories, обеспечивает целостность ссылок между товарами и категориями.

Order (заказ):

Order ID: Первичный ключ, обеспечивающий уникальность каждого заказа.

User: Внешний ключ, связанный с User ID в таблице User, обеспечивает связь между заказами и пользователями.

Good: Внешний ключ, связанный с Good ID в таблице Goods, обеспечивает связь между заказами и товарами.

Payment (оплата):

Payment ID: Первичный ключ, уникальный для каждого платежа.

Order: Внешний ключ, связанный с Order ID в таблице Order, обеспечивает связь между платежами и заказами.

## Целостность сущностей

User (пользователь):

Full name, Phone number, E-mail, Login, Password: Обязательные для заполнения поля обеспечивают целостность данных в рамках сущности пользователя.

ISAdmin: Булево значение, необязательное для заполнения, определяет, является ли пользователь администратором.

Categories (категории):

Name: Обязательное для заполнения поле, определяющее имя категории.

Goods (товары):

Name, Category, Price, Image: Обязательные для заполнения поля, обеспечивающие целостность данных, в сущности - товаров.

Description: Необязательное поле.

Order (заказ):

User, Good, Count: Обязательные для заполнения поля, определяющие пользователя, товар и количество товаров в заказе.

Payment (оплата):

Payment method, Paid, Amount, Date: Обязательные для заполнения поля, определяющие метод оплаты, статус оплаты, сумму и дату платежа.

Обеспечив целостность ссылок и целостность сущностей в базе данных, мы создаем структуру данных, которая минимизирует возможность ошибок, дублирования и обеспечивает согласованность данных.

Таблица описания полей базы данных.

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица | Атрибут | Тип | Размер |
| User(пользователь) | User ID | Number |  |
|  | Full name | varchar | (50) |
|  | Phone number | varchar | (12) |
|  | E-mail | varchar | (50) |
|  | Login | varchar | (50) |
|  | Password | varchar | (50) |
|  | ISAdmin | bool |  |
| Сategories(категории) | Сategory ID | Number |  |
|  | Name | varchar | (50) |
|  | Description | text |  |
| Goods(товары) | Good ID | Number |  |
|  | Name | varchar | (50) |
|  | Сategory | Number |  |
|  | Price | real |  |
|  | Image | varchar | (50) |
|  | Description | text |  |
| Order(заказ) | Order ID | Number |  |
|  | User | Number |  |
|  | Good | Number |  |
|  | Count | int |  |
| Payment(оплата) | Payment | Number |  |
|  | Order | Number |  |
|  | Payment metod | varchar | (50) |
|  | Paid | bool |  |
|  | Amount | real |  |
|  | Date | date |  |

## ER диаграммы.

Логическая модель данных.

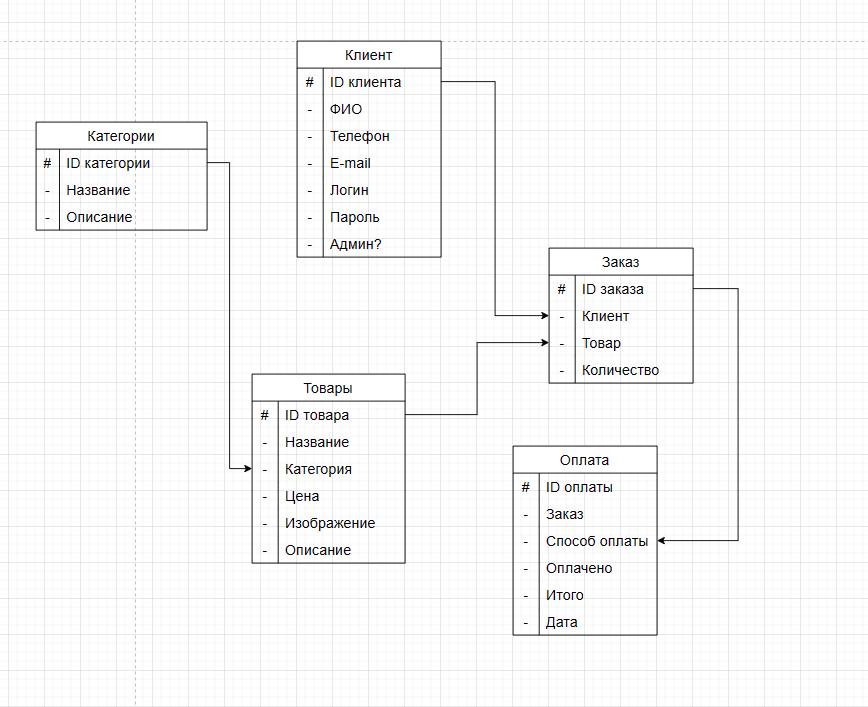


Рисунок 1 – Логическая ER диаграмма.

Физическая модель данных.

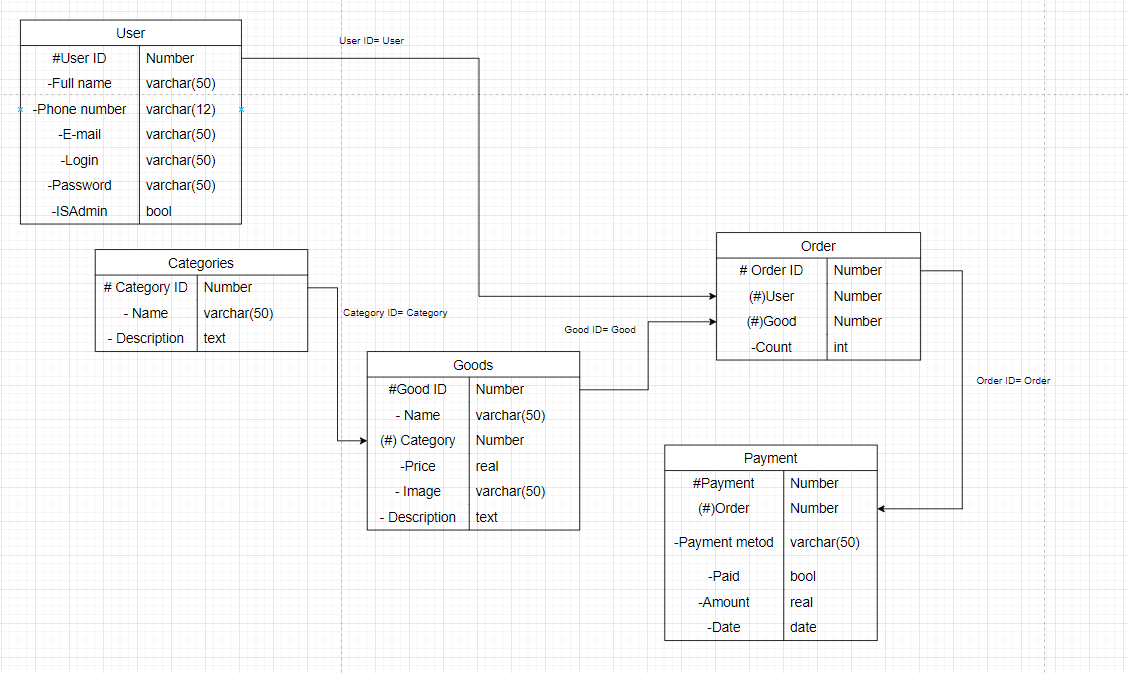


Рисунок – Физическая ER диаграмма.

# Алгоритмы решения задачи

Алгоритм «Алгоритм редактирования товара».

Ниже на рисунке 3 представлена схема работы этого алгоритма.

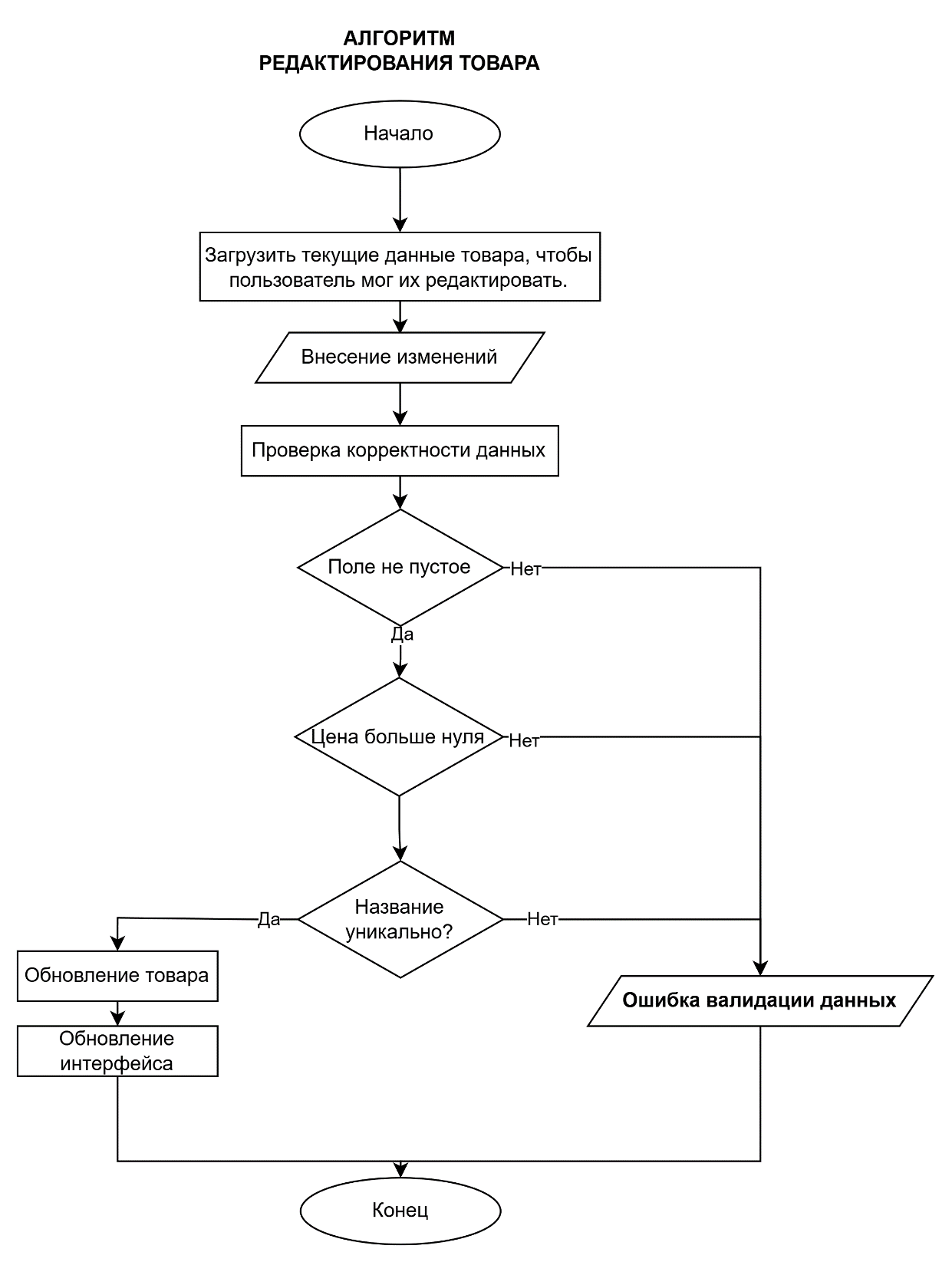


Рисунок 3 – Алгоритм редактирования заказа.

Алгоритм «Алгоритм добавления товара».

Ниже на рисунке 4 представлена схема работы этого алгоритма.

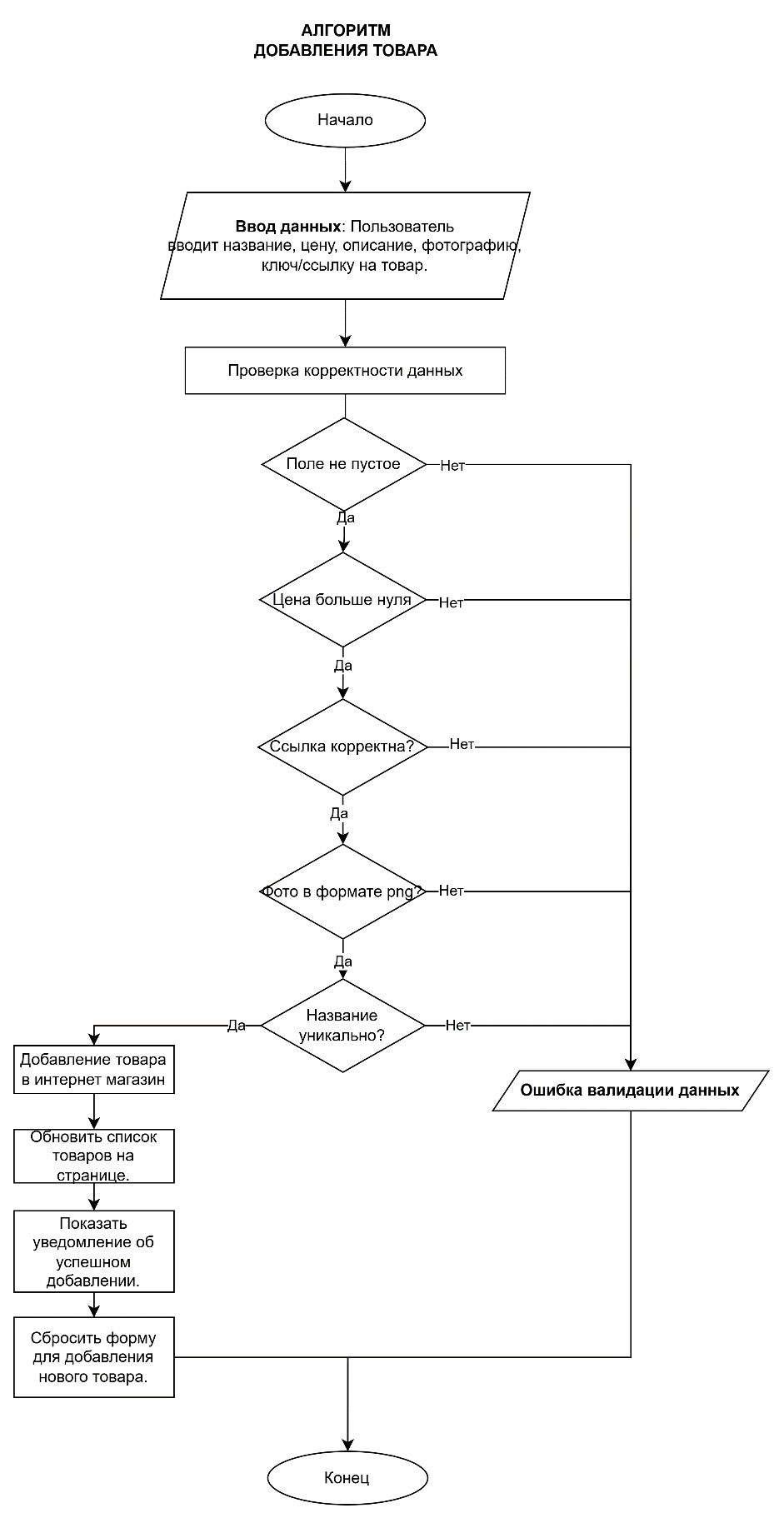


Рисунок 4 - Алгоритм добавления товара.

Алгоритм «Алгоритм поиска товара».

Ниже на рисунке 5 представлена схема работы этого алгоритма.

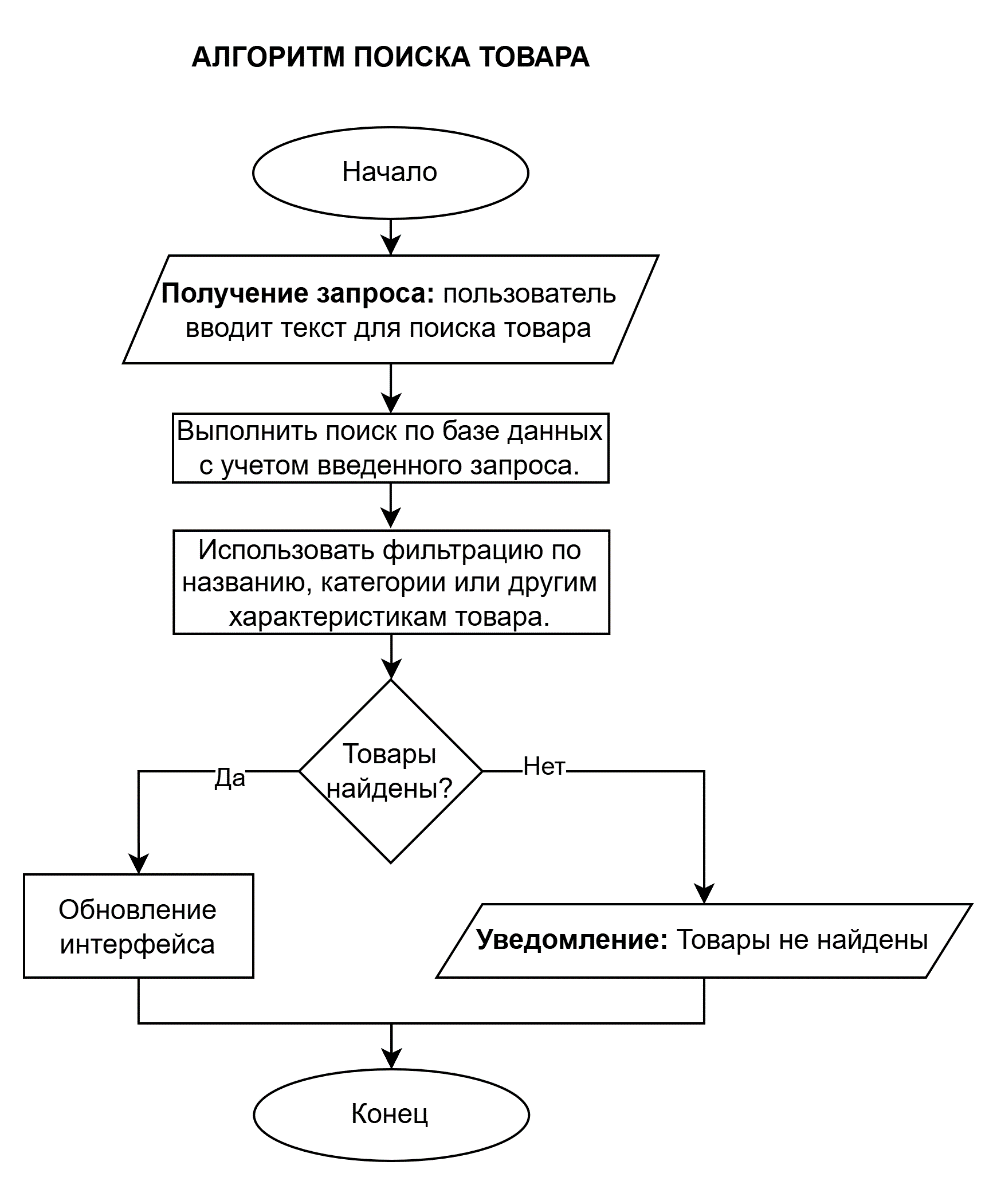


Рисунок 5 - Алгоритм поиска товара.

Алгоритм «Алгоритм управления корзиной».

Ниже на рисунке 6 представлена схема работы этого алгоритма.

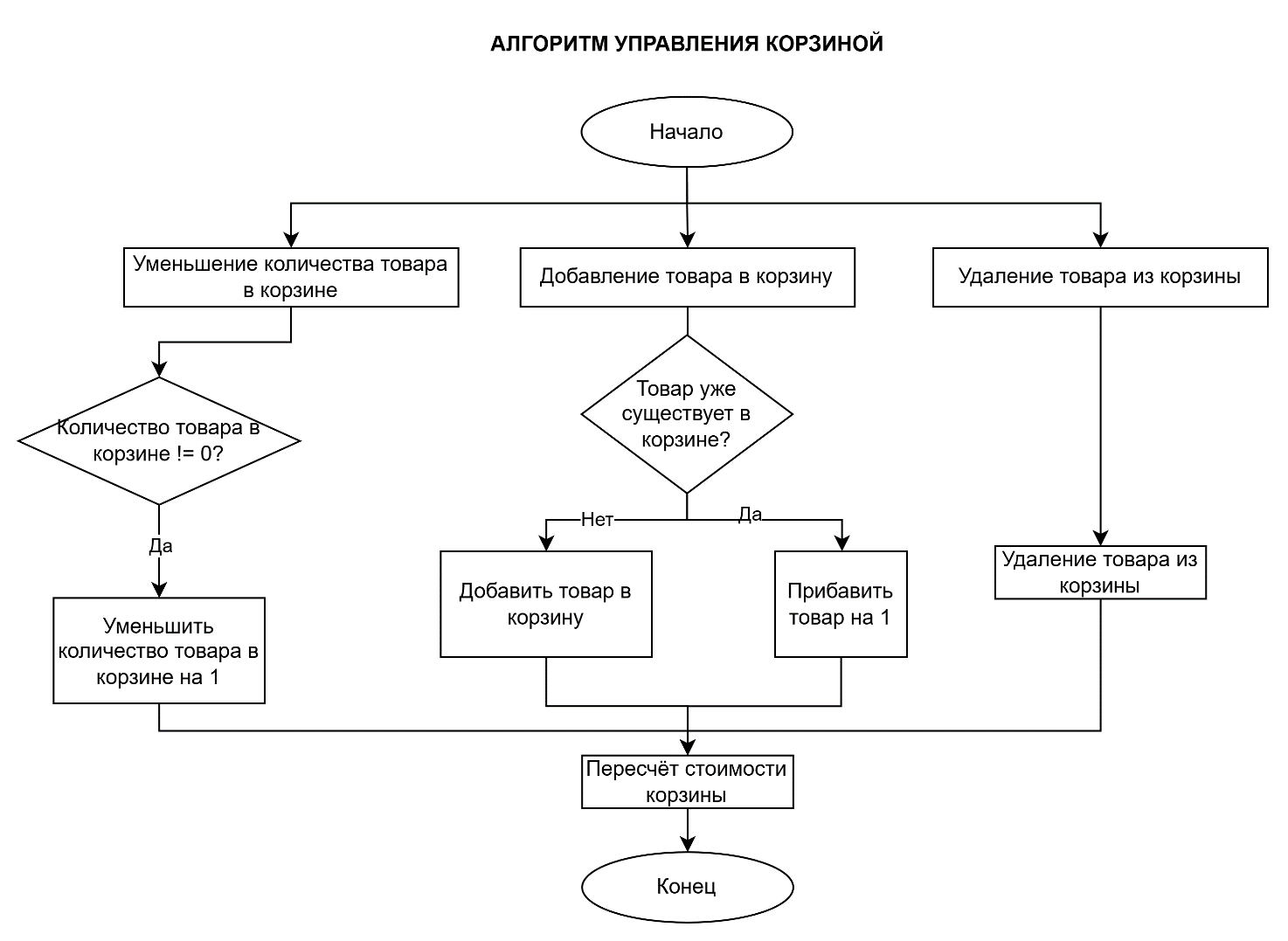


Рисунок 6 - Алгоритм управления корзиной.

Алгоритм «Алгоритм удаления корзиной».

Ниже на рисунке 7 представлена схема работы этого алгоритма.

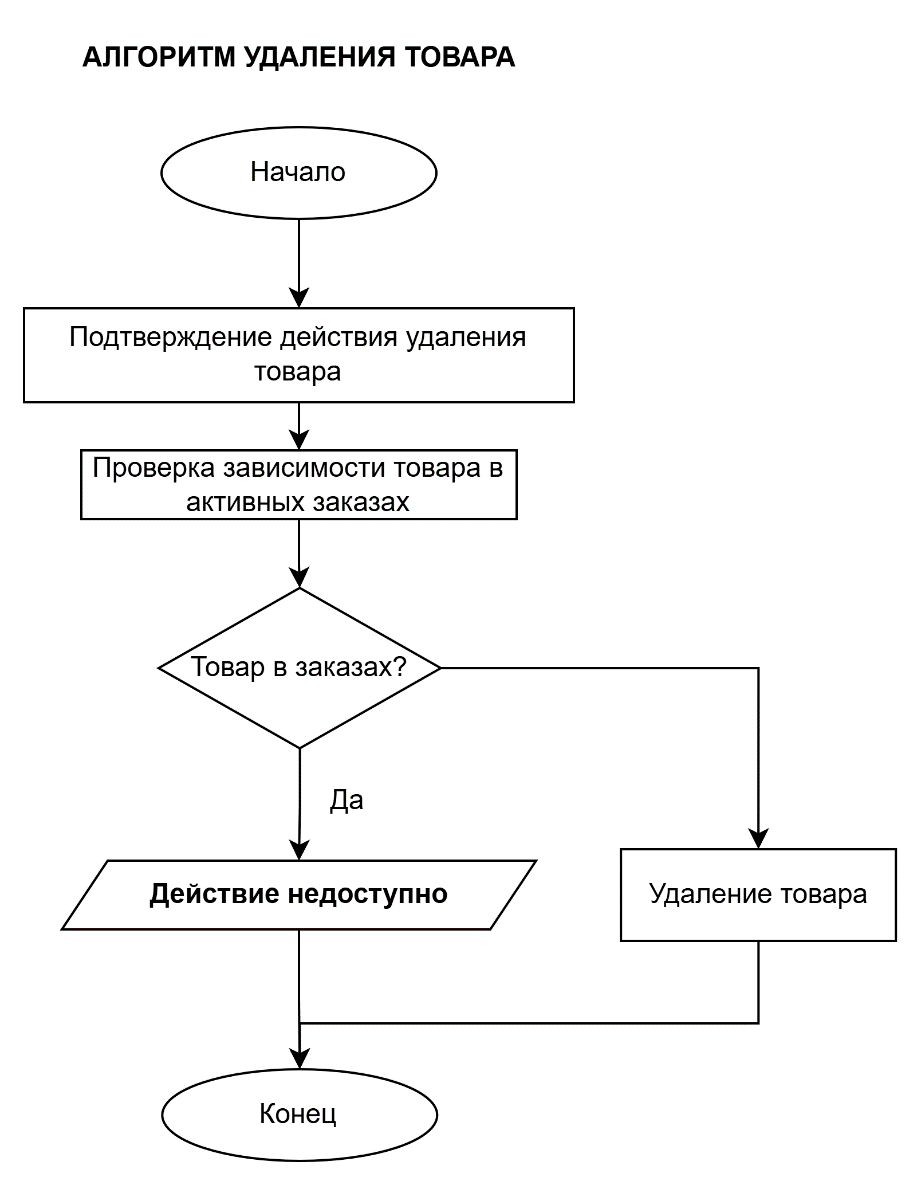


Рисунок 7 – Алгоритм удаления корзиной.

Алгоритм «Алгоритм формирования отчёта».

Ниже на рисунке 8 представлена схема работы этого алгоритма.

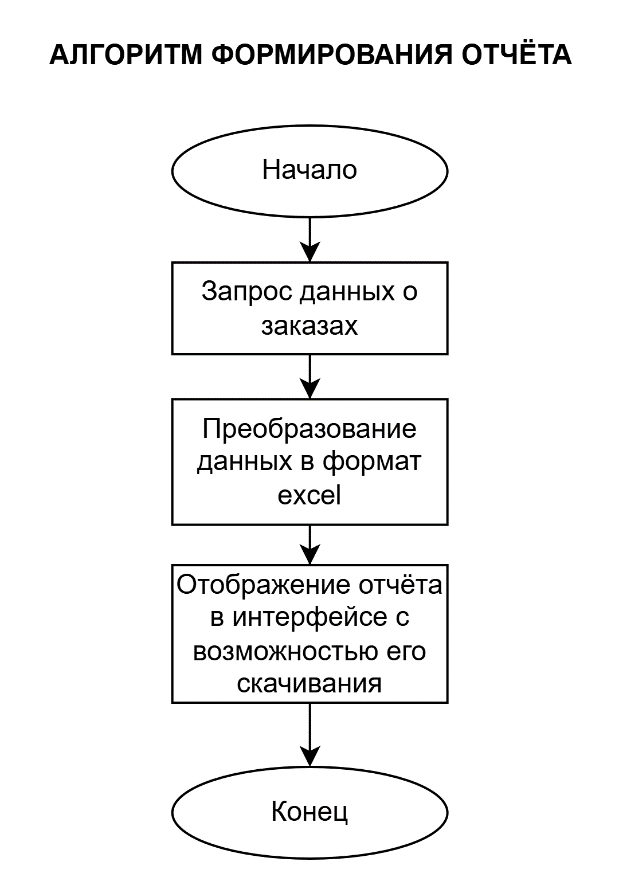


Рисунок 8 – Алгоритм формирования отчёта.

Алгоритм «Алгоритм оформления заказа».

Ниже на рисунке 9 представлена схема работы этого алгоритма.

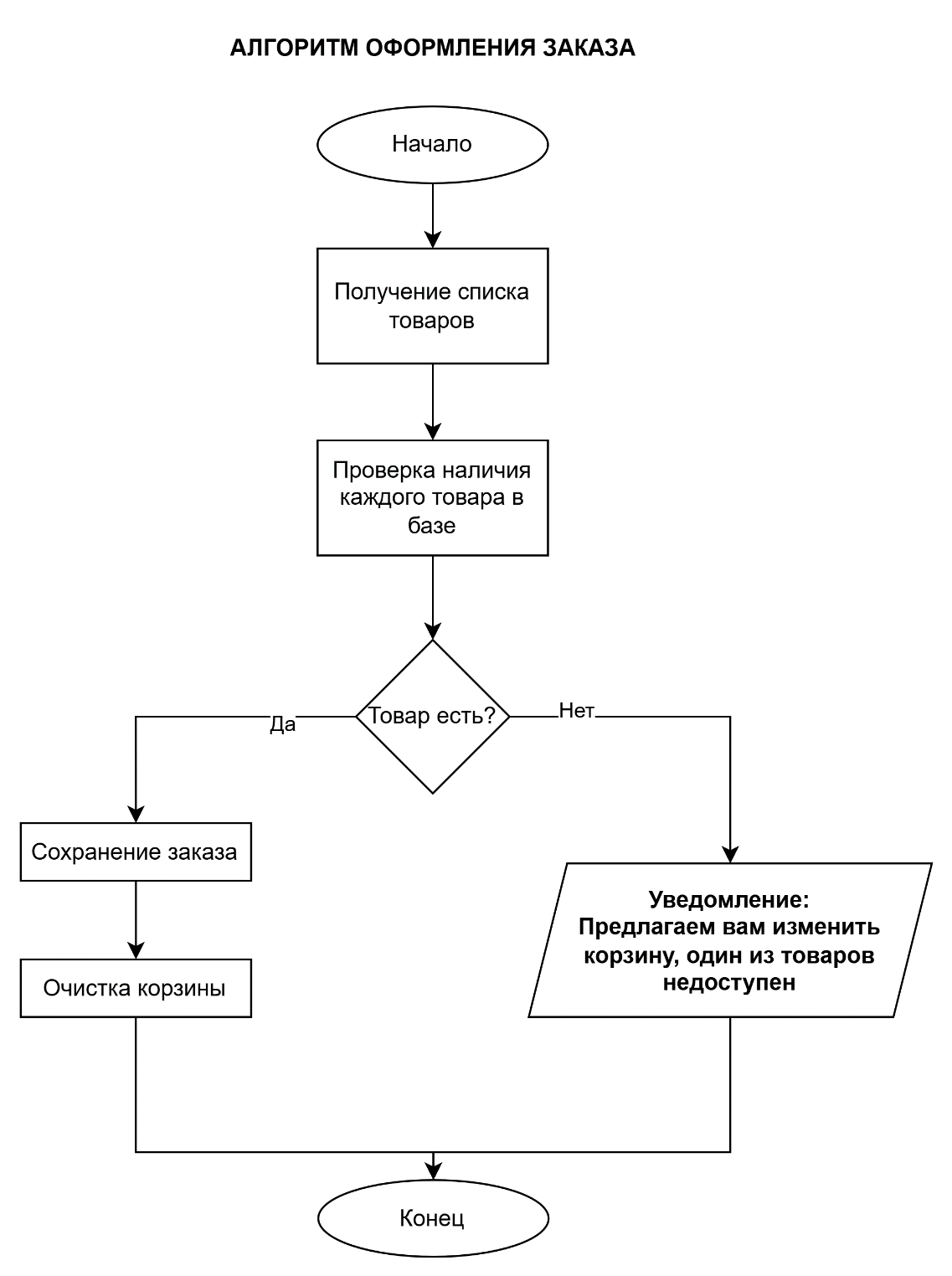


Рисунок 9 - Алгоритм оформления заказа

Алгоритм «Алгоритм авторизации/регистрации».

Ниже на рисунке 10 представлена схема работы этого алгоритма.

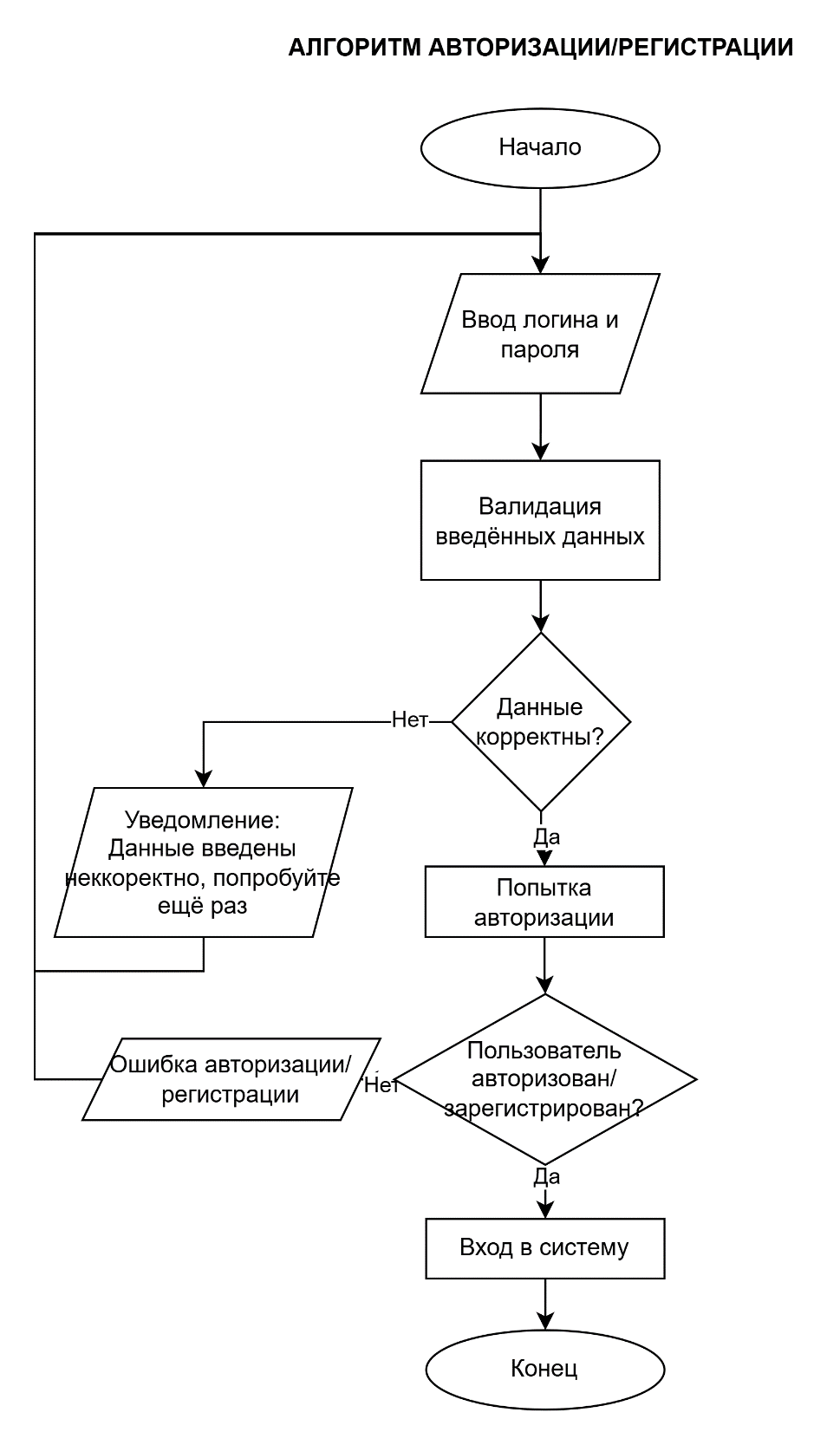


Рисунок 10 – Алгоритм авторизации/регистрации.

Так же для понимания как реагирует система был составлен пользовательский сценарий, который указан в таблице 1 и 2.

Таблица 1 – Пользовательские сценарий (для администратора)

|  |  |  |
| --- | --- | --- |
| № функции | Администратор | Система |
| 1 | Сценарий добавления товара | 1. Проверяет, что все поля заполнены и соответствуют требованиям (не пустые, цена положительное число, фотография имеет допустимый формат, ссылка корректна). 2. Если все проверки пройдены успешно, создается новый товар в базе данных и привязывается изображение, если оно было добавлено. 3. Система обновляет список товаров на странице, показывая новый товар. 4. Пользователь получает уведомление об успешном добавлении товара. 5. Форма очищается для добавления следующего товара. |
| 2 | Сценарий редактирования товара | 1. Проверяет, что все измененные поля соответствуют требованиям (не пустые, цена положительное число, уникальность названия). 2. Если фото было изменено, старое фото удаляется, и загружается новое. 3. Данные товара обновляются в базе. 4. Система обновляет список товаров на странице, отображая измененный товар. 5. Пользователь получает уведомление об успешном редактировании. |
| 3 | Сценарий удаления товара | 1. Проверяет, не используется ли товар в активных заказах. Если используется, предлагает изменить его статус на "Недоступен" или "Снят с продажи". 2. Удаляет товар из базы данных. 3. Удаляет все связанные с товаром изображения и записи. 4. Система обновляет список товаров, исключая удаленный товар. 5. Пользователь получает уведомление об успешном удалении товара. |

|  |  |  |
| --- | --- | --- |
| № функции | Пользователь | Система |
| 1 | Сценарий поиска товара | 1. Выполняет поиск по базе данных товаров с учетом введенного текста (по названию, категории и другим характеристикам). 2. Показивает найденные товары на странице. 3. Если товары не найдены, отображается сообщение "Товары не найдены". 4. При необходимости предоставляется возможность фильтровать результаты по цене, категории и другим параметрам. |
| 2 | Сценарий управления корзиной | 1. При уменьшении количества товара на 1 проверяет, не достигает ли количество товара минимального (например, 1). 2. Если количество товара в корзине достигает 0, товар удаляется из корзины. 3. Система пересчитывает общую стоимость корзины после добавления, удаления или изменения товара. |
| 3 | Сценарий оформления заказа | 1. Проверяет наличие товаров на складе и информирует покупателя, если какого-то товара нет в наличии. 2. Создает запись в базе данных о заказе, включая товары, количество, цену и статус. 3. Генерирует уникальный номер заказа. 4. После успешного оформления заказа очищает корзину покупателя. 5. Покупатель получает уведомление об успешном оформлении заказа. |
| 4 | Сценарий формирования отчета | 1. Получает список заказов за выбранный период. 2. Формирует отчет в нужном формате (например, PDF, CSV, Excel), включая данные о покупателях, товарах, общей стоимости и статусах заказов. 3. Отображает отчет на странице. 4. Администратор получает опцию для скачивания отчета или отправки его на email. |

Таблица 2 – Пользовательские сценарий (для пользователя)

# Формы представления данных.

Форма представления для авторизации/регистрации должна обеспечить выполнение следующий функций:

* Содержать окно взаимодействия для авторизации пользователей;
* Содержать два поля для ввода логина и пароля;
* Содержать кнопку «Войти».
* Содержать кнопку «Зарегистрироваться».

Прототип интерфейса авторизации представлен на рисунке 11.

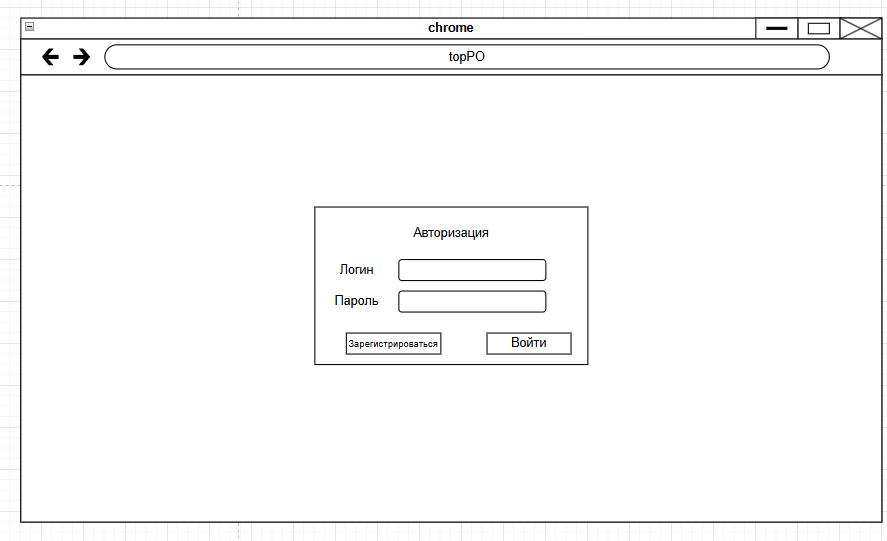


Рисунок 11 – Прототип интерфейса окна авторизации



Рисунок 12 – Прототип вкладки Приложения

Вкладка Приложения должна содержать поле для поиска, кнопку «Найти», каталог приложений.

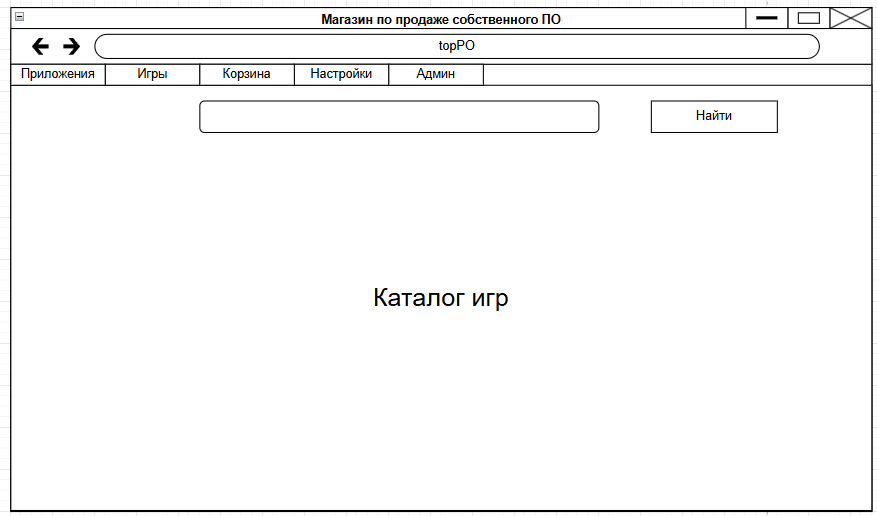


Рисунок 13 – Прототип вкладки Игры

Вкладка Приложения должна содержать поле для поиска, кнопку «Найти», каталог игр.

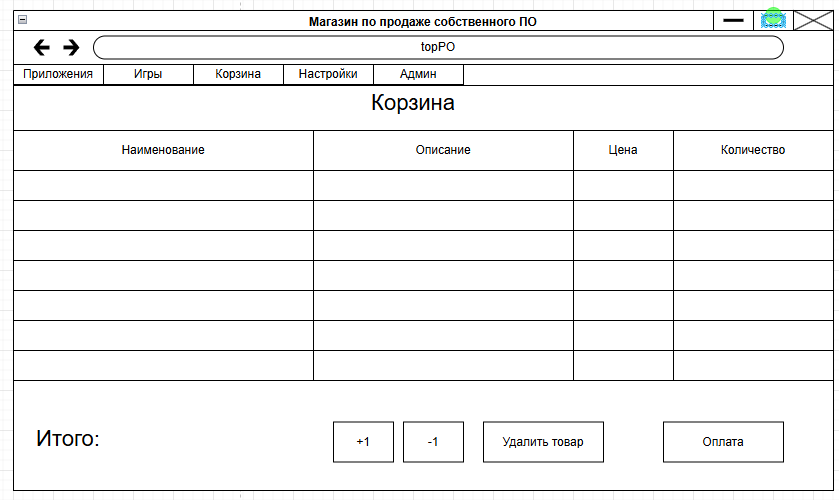


Рисунок 14 – Прототип вкладки Корзина

Вкладка Козина должна содержать таблицу с товарами, выбранными пользователем, кнопки: «+1», «-1», «Удалить товар», «Оплата».

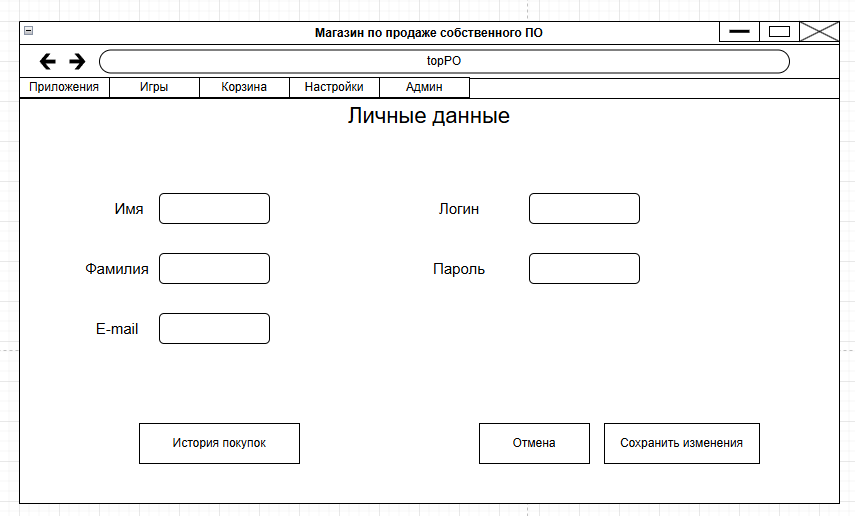


Рисунок 15 – Прототип вкладки Настройки

Вкладка Козина должна содержать поля для ввода имени, фамилии, эл.почты, логина, пароля , кнопки: «История покупок»(при нажатии вызывается окно с историей покупок), «Сохранить изменения», «Оплата».

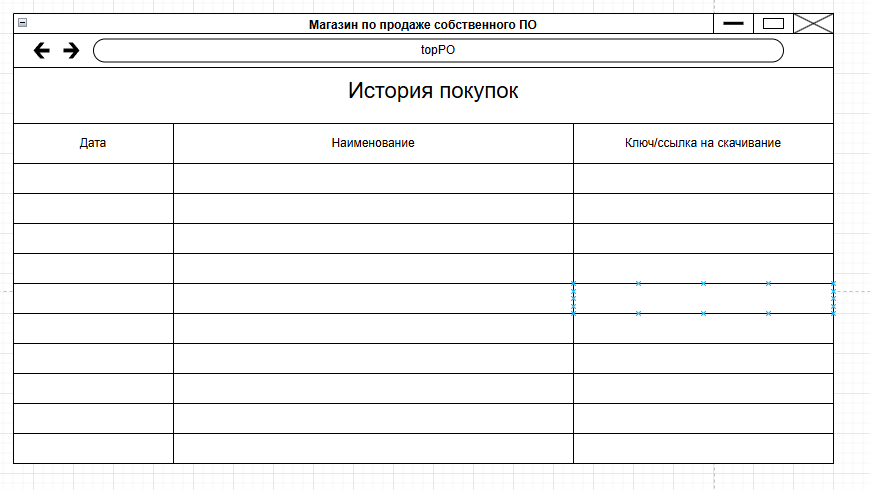


Рисунок 16 – Прототип окна истории покупок

Окно с историей покупок должно содержать таблицу с историей покупок пользователя.

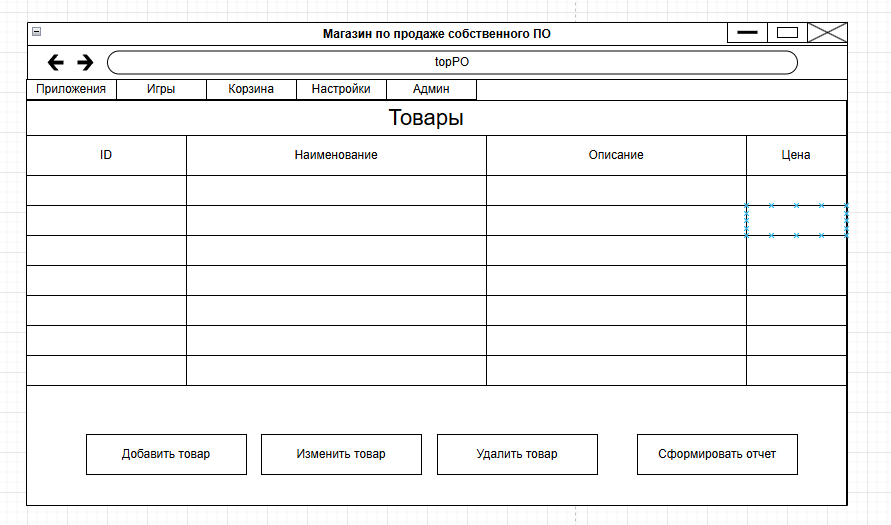


Рисунок 17 – Прототип вкладки Админ

Вкладка Админ должна содержать таблицу с товарами, имеющимися в БД, кнопки: «Добавить товар», «Изменить товар»(при нажатии вызывается окно для редактирования информации в БД), «Удалить товар», «Сформировать отчет» »(при нажатии вызывается окно для просмотра отчетности).

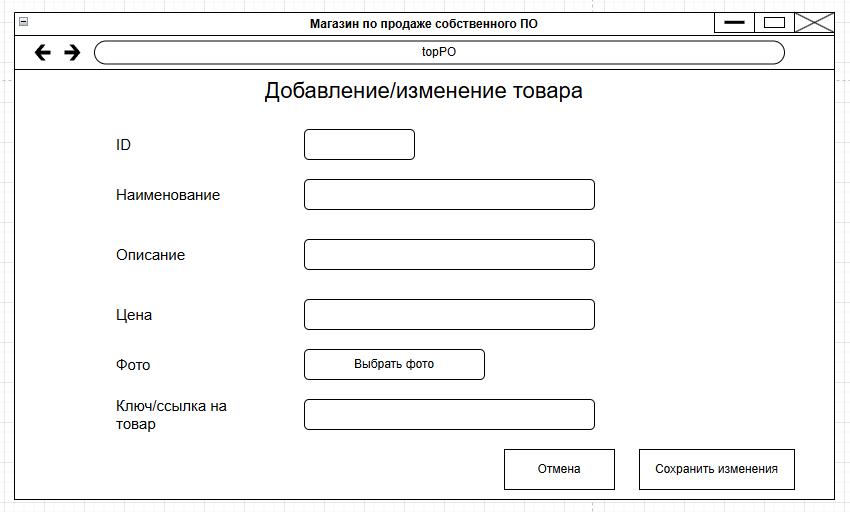


Рисунок 18 – Прототип окна редактирования информации в БД

Окно редактирования информации в БД должно поля для ввода ID, наименования, описания, цены, ключа/ссылки на товар, кнопки: «Сохранить изменения, «Выбрать фото», «Отмена».

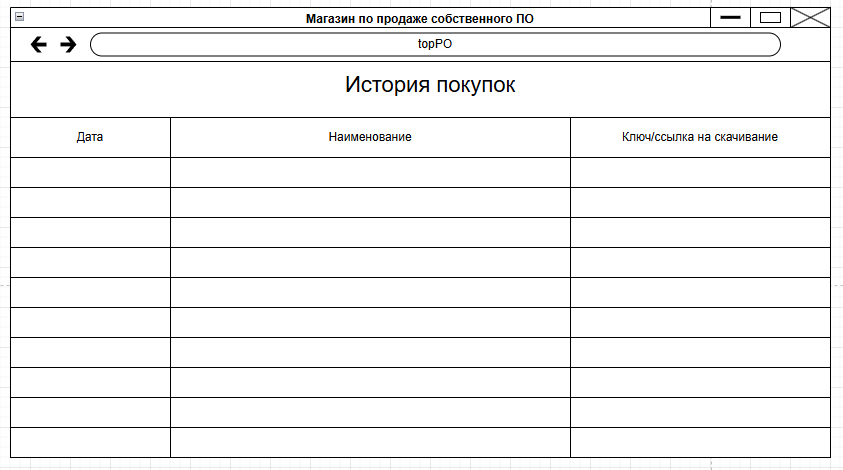


Рисунок 19 – Прототип окна с отчетностью

Окно с отчетностью должно содержать отчетность.

# Структура программы

Проект с FastAPI на бэкенде и Nuxt.js SPA на фронте будет организован так, чтобы реализовать четкое разделение обязанностей между серверной и клиентской частями. Вот как будет устроено их взаимодействие:

* 1. **Бэкенд (FastAPI)**

FastAPI выступает в роли REST API сервера. Его задачи:

* Обрабатывать HTTP-запросы от клиента (Nuxt.js).
* Управлять бизнес-логикой.
* Подключаться к базе данных (MySQL) с помощью SQLAlchemy для работы с данными.
* Обеспечивать безопасность (аутентификация, авторизация, валидация данных).
* Возвращать данные в формате JSON, которые будут использоваться на фронте.

Основной функционал:

FastAPI использует SQLAlchemy для работы с MySQL: подключение, создание таблиц, выполнение CRUD-операций.

Запросы от клиента (GET, POST, PUT, DELETE) обрабатываются через роутеры, которые вызывают функции из сервисов.

* 1. **Фронтенд (Nuxt.js SPA)**

Nuxt.js работает в режиме SPA (Single Page Application). Его задачи:

* Отображать интерфейс для взаимодействия с пользователем.
* Отправлять запросы к API FastAPI для получения и обновления данных.
* Обрабатывать ответы API (например, список пользователей, детали объекта) и показывать их в интерфейсе.
* Управлять состоянием приложения (Pinia).

Основной функционал:

Axios используется для отправки запросов к REST API.

Данные, полученные от FastAPI, динамически отображаются с помощью компонентов Vue.js.

Интерфейс может включать таблицы, формы, графики и другие элементы для работы с данными.

* 1. **Взаимодействие между FastAPI и Nuxt.js.**

Процесс их взаимодействия можно разделить на несколько этапов:

Получение данных

Пользователь открывает страницу (например, /goods) на Nuxt.js SPA. Компонент на этой странице через Axios отправляет запрос GET /api/goods на сервер FastAPI. FastAPI обрабатывает запрос и вызывает соответствующий роутер. Роутер вызывает функцию сервиса, которая обращается к базе данных через SQLAlchemy. SQLAlchemy выполняет запрос к MySQL (например, SELECT \* FROM goods). Результат передается обратно в роутер, а затем сериализуется в JSON. Nuxt.js получает JSON-ответ (например, массив приложений) и рендерит его в виде каталога приложений.

Добавление данных

Пользователь заполняет форму в интерфейсе (например, создание нового товара). После нажатия кнопки "Создать" компонент Nuxt.js отправляет запрос POST /api/goods с телом запроса (JSON) на сервер FastAPI. FastAPI обрабатывает запрос и создает новую запись в базе данных с помощью SQLAlchemy. Возвращает подтверждение или созданный объект в JSON.Nuxt.js получает подтверждение и обновляет интерфейс.

Редактирование данных

Пользователь выбирает товар(приложение) для редактирования. Nuxt.js отправляет запрос PUT /api/goods/{id} с обновленными данными. FastAPI обновляет запись в MySQL через SQLAlchemy. Возвращает обновленную запись в JSON. Интерфейс обновляется с учетом новых данных.

Удаление данных

Пользователь нажимает кнопку "Удалить" рядом с записью. Nuxt.js отправляет запрос DELETE /api/goods/{id}. FastAPI удаляет запись из базы данных и возвращает подтверждение. Nuxt.js удаляет элемент из интерфейса.

* 1. **Технические особенности**

FastAPI (Back-end):

Работа с MySQL. FastAPI подключается к MySQL через SQLAlchemy. Используется драйвер, aiomysql для асинхронных запросов.

Уровни приложения:

Models. SQLAlchemy описывает структуры таблиц базы данных.

Routers. Роутеры разделяют эндпоинты на модули (например, /users, /posts).

Services. Сервисы содержат бизнес-логику (например, фильтрация, обработка данных).

Nuxt.js (Front-end):

Одностраничное приложение работает на клиенте, загружает данные по мере необходимости через API. Axios используется для отправки запросов к FastAPI. Для хранилища состояний используется библиотека Pinia. Динамическая маршрутизация, страницы Nuxt.js подгружают данные с сервера на основе параметров маршрута.

Пример взаимодействия (Flow)

Пользователь открывает приложение Nuxt.js. Nuxt.js отправляет запрос GET /api/posts на FastAPI. FastAPI запрашивает записи из базы данных MySQL. Данные возвращаются на фронт, и Nuxt.js отображает их в интерфейсе. Пользователь добавляет новую запись, и Nuxt.js отправляет POST /api/posts с JSON-данными. FastAPI создает запись, и интерфейс обновляется.

FastAPI отвечает за Model и Controller: обработка данных, работа с базой, логика API. Nuxt.js SPA отвечает за View: отображение данных и взаимодействие с пользователем. Взаимодействие между ними осуществляется через REST API. Эта структура обеспечивает хорошую модульность, читаемость и разделение ответственности между клиентом и сервером.

Схема взаимодействия представлена ниже.

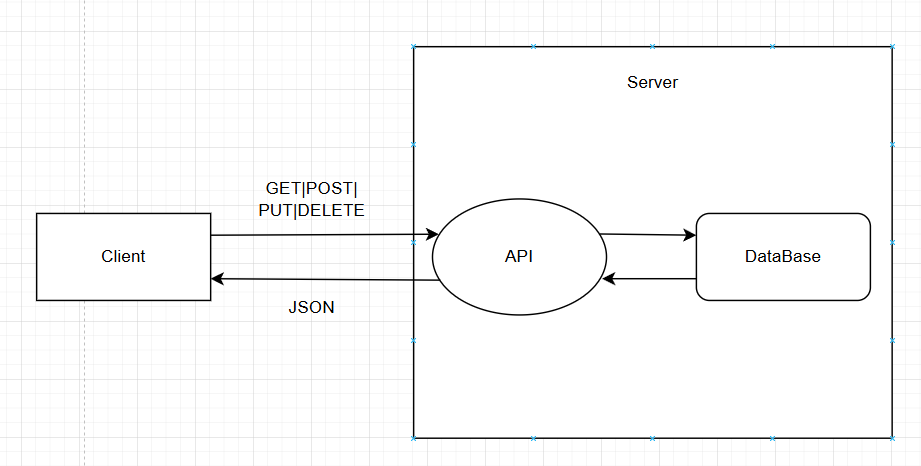


Рисунок 20 – Схема взаимодействия.

# Требования к техническим средствам

Минимальные требования для информационной системы:

Операционная система: Windows 10;

ОЗУ: 2 гб;

Жесткий диск: 32 гб;

Видеоадаптер с поддержкой DirectX 9 или более поздних выпусков с драйвером WDDM;

Монитор с разрешением не ниже 800x600px.