|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное автономное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_**ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_\_\_\_\_\_\_**КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

НАПРАВЛЕНИЕ ПОДГОТОВКИ \_\_\_\_\_\_\_\_**09.03.01 Информатика и вычислительная техника**\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОЙ РАБОТЕ***

***НА ТЕМУ:***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***Сайт для размещения***\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***электронных объявлений***\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Студент \_\_\_\_ИУ6-51Б\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Д.Ю. Воронин**\_\_\_**

(Группа) (Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Д.А. Миков**\_\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

*Количество баллов:*

*за программный продукт –*

*за расчетно-пояснительную записку –*

*за доклад и ответы на вопросы –*

*Итого:*

*Оценка:*

*2025 г.*

**РЕФЕРАТ**

Расчетно-пояснительная записка состоит из 23 страниц и включает в себя 14 рисунков, 2 приложения, 5 источников.

Целью данной курсовой работы является проектирование, реализация и документирование системы «Омега» в соответствии с требованиями технического задания и современными практиками разработки программного обеспечения. В процессе выполнения работы применяются принципы объектно-ориентированного программирования и клиент-серверной архитектуры. Результатом работы станет функциональный программный продукт, сопровождаемый расчётно-пояснительной запиской и необходимой программной документацией.

Для реализации используются реляционная база данных PostgreSQL 17, язык программирования Golang 1.23, библиотека React 13.3, среды разработки PgAdmin 4 и Visual Studio Code.

СОДЕРЖАНИЕ

[Введение 4](#_Toc1)

[1 Анализ требований и уточнение спецификаций 5](#_Toc2)

[1.1 Анализ задания и выбор технологии, языка и среды разработки 5](#_Toc3)

[1.2 Разработка диаграммы вариантов использования 6](#_Toc4)

[1.3 Выбор методов решения задачи 7](#_Toc5)

[2 Проектирование структуры и компонентов программного продукта 8](#_Toc6)

[2.1 Разработка интерфейса пользователя 8](#_Toc7)

[2.1.1 Построение графа (диаграммы) состояний интерфейса 8](#_Toc8)

[2.1.2 Разработка форм ввода-вывода информации 9](#_Toc9)

[2.3 Разработка структурной схемы программного продукта 13](#_Toc10)

[2.4 Проектирование даталогической модели базы данных 14](#_Toc11)

[2.5 Проектирование классов для реализации интерфейса и предметной области 16](#_Toc12)

[2.6 Разработка диаграммы последовательности действий 18](#_Toc13)

[3 Выбор стратегии тестирования и разработка тестов 20](#_Toc14)

[Заключение 22](#_Toc15)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23](#_Toc16)

# Введение

В условиях стремительного развития цифровых технологий и роста онлайн-торговли всё большее значение приобретают платформы, позволяющие частным лицам и представителям малого бизнеса легко и быстро размещать и находить объявления о продаже товаров и услуг. Существующие крупные маркетплейсы, такие как Avito или Юла, несмотря на широкую популярность, зачастую создают барьеры для новых участников рынка: высокая конкуренция и значительные комиссии делают их использование не всегда выгодным для продавцов и покупателей.

Актуальность разработки собственной платформы «Омега» обусловлена необходимостью предоставления упрощённого и экономически выгодного решения. Проект ориентирован на пользователей, которые ценят простоту, прозрачность и прямое взаимодействие без посредников. Основные преимущества системы – минимальные комиссии, прямые контакты между покупателями и продавцами, а также лаконичный пользовательский интерфейс, не перегруженный избыточной функциональностью.

Разрабатываемая система представляет собой полноценное веб-приложение с разделением на клиентскую (frontend) и серверную (backend) части, реализующее ключевые функции: регистрацию и авторизацию пользователей, размещение, редактирование и удаление объявлений, поиск по заголовку, просмотр объявлений, загрузку изображений и взаимодействие через контактные данные. Особое внимание уделено надёжности, безопасности вводимых данных и целостности хранения информации в базе данных.

# 1 Анализ требований и уточнение спецификаций

## 1.1 Анализ задания и выбор технологии, языка и среды разработки

На основе анализа технического задания к проекту «Омега» были определены следующие ключевые требования к системе:

* поддержка двух типов пользователей (авторизованных и неавторизованных);
* реализация CRUD-операций для объявлений (создание, чтение, редактирование, удаление);
* возможность поиска объявлений по заголовку;
* загрузка изображений;
* обеспечение надёжности и контроля вводимых данных;
* поддержка основных веб-браузеров;
* простота пользовательского интерфейса.

Для реализации веб-приложения с такими характеристиками была выбрана клиент-серверная архитектура с разделением на frontend и backend. Такой подход обеспечивает гибкость, масштабируемость и удобство сопровождения.

Frontend разработан с использованием React – современной JavaScript-библиотеки для построения пользовательских интерфейсов. React позволяет создавать модульные, переиспользуемые компоненты и обеспечивает минимально необходимое изменение DOM при изменении состояния [1]. Стилизация выполнена с использованием библиотеки Bootstrap, что позволяет разрабатывать интерфейсы быстрее, чем на чистом CSS, сохраняя при этом современный и аккуратный вид приложения.

Backend реализован на языке Golang с использованием библиотеки echo и дополняющих пакетов, таких как x/crypto для хеширования паролей и gorm.io для взаимодействия с PostgreSQL посредством технологии ORM, которая связывает объекты языка с реляционной базой данных[2]. Go был выбран благодаря своей производительности, простоте синтаксиса, встроенной поддержке конкурентности и сильной типизации, что снижает вероятность ошибок на этапе разработки. Архитектура backend построена по принципам чистой архитектуры (Clean Architecture): чёткое разделение на слои handlers, service, repository и models обеспечивает тестируемость и независимость от внешних зависимостей. Для управления схемой базы данных используются миграции с помощью утилиты migrate, что гарантирует воспроизводимость и контроль версий структуры БД.

В качестве СУБД выбрана PostgreSQL – надёжная, производительная и полнофункциональная реляционная база данных с отличной поддержкой в экосистеме Go. Её использование позволяет эффективно работать со сложными запросами и обеспечивать целостность данных благодаря реализации в PostgreSQL концепции ACID [3].

Выбранный технологический стек (Go + React + PostgreSQL) полностью соответствует требованиям курсовой работы: он современен, поддерживает разработку программного продукта средней сложности, обеспечивает развитый пользовательский интерфейс и позволяет применять инструменты автоматизации и контроля версий, такие как Git. Кроме того, совместимость этих технологий между собой и их активные сообщества упрощают поиск решений типовых задач и способствуют быстрому устранению возникающих проблем.

## **1.2 Разработка диаграммы вариантов использования**

Для формализации функциональных требований к веб-приложению «Омега» была построена диаграмма вариантов использования в соответствии с методологией UML – рисунок 1. В системе выделены два типа пользователей, указанных в техническом задании. Незарегистрированный пользователь может только просматривать и искать объявления. Зарегистрированный пользователь же обладает полным набором функций: создание, редактирование и удаление собственных объявлений, а также просмотр и поиск.

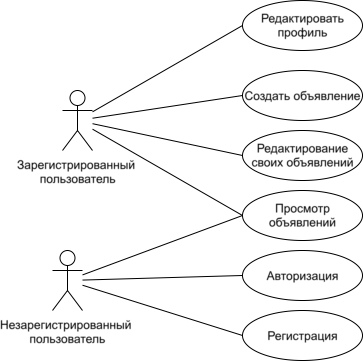


Рисунок 1 – Диаграмма вариантов использования

## **1.3 Выбор методов решения задачи**

Для аутентификация используется JSON Web Token. После проверки логина и хэша пароля (с помощью bcrypt) сервер выдаёт токен с user\_id и сроком действия 24 часа. Все защищённые маршруты проверяют наличие и валидность токена через middleware.

Все пользовательские данные проходят валидацию:

* имя пользователя от 3 до 100 символов;
* логин имеет корректный формат электронной почты;
* номер телефона является корректным;
* пароль от 6 символов;
* заголовок объявления от 3 до 100 символов;
* описание объявления от 20 до 5000 символов;
* изображения размером не более 10 МБ в форматах JPEG, JPG, PNG или WEBP.

Поиск объявлений реализован как регистронезависимый подстроковый поиск с использованием SQL-оператора ILIKE в PostgreSQL.

# 2 Проектирование структуры и компонентов программного продукта

## 2.1 Разработка интерфейса пользователя

### 2.1.1 Построение графа (диаграммы) состояний интерфейса

Интерфейс веб-приложения «Омега» реализован в виде одностраничного приложения (SPA) на основе библиотеки React. В соответствии с требованиями технического задания была разработана диаграмма состояний интерфейса зарегистрированного и незарегистрированного пользователей – рисунок 2.

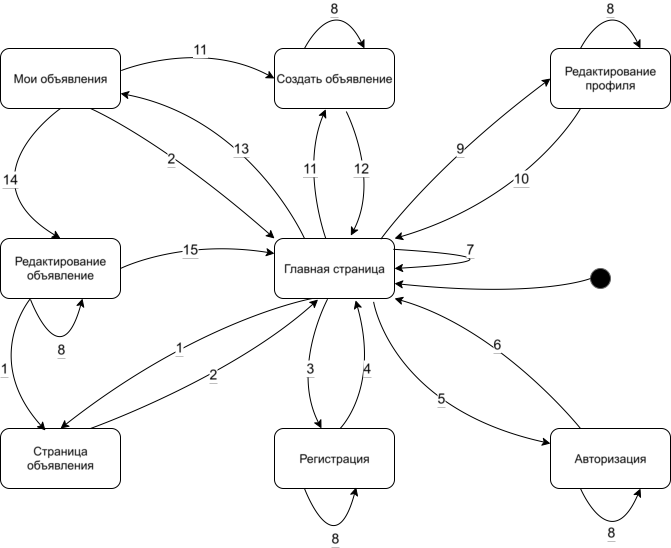


Рисунок 2 – Диаграмма состояний интерфейса

Ниже представлены условия переходов между состояниями:

* 1 – нажатие на кнопку «Просмотр» у объявления;
* 2 – возврат на главную страницу;
* 3 – нажатие на кнопку «Регистрация»;
* 4 – возврат на главную страницу или успешная регистрация;
* 5 – нажатие на кнопку «Войти»;
* 6 – возврат на главную страницу или успешная авторизация;
* 7 – поиск или фильтр по категориям;
* 8 – ввод некорректных данных;
* 9 – нажатие на кнопку «Профиль» у выпадающего меню профиля;
* 10 – успешное редактирование профиля;
* 11 – нажатие на кнопку «Создать объявление»;
* 12 – возврат на главную страницу или успешное создание объявление;
* 13 – нажатие на кнопку «мои объявления»;
* 14 – нажатие на кнопку «Редактировать» у объявления;
* 15 – возврат на главную или успешное редактирование объявления.

### 2.1.2 Разработка форм ввода-вывода информации

Были разработаны следующие формы интерфейса – рисунки 3-10.

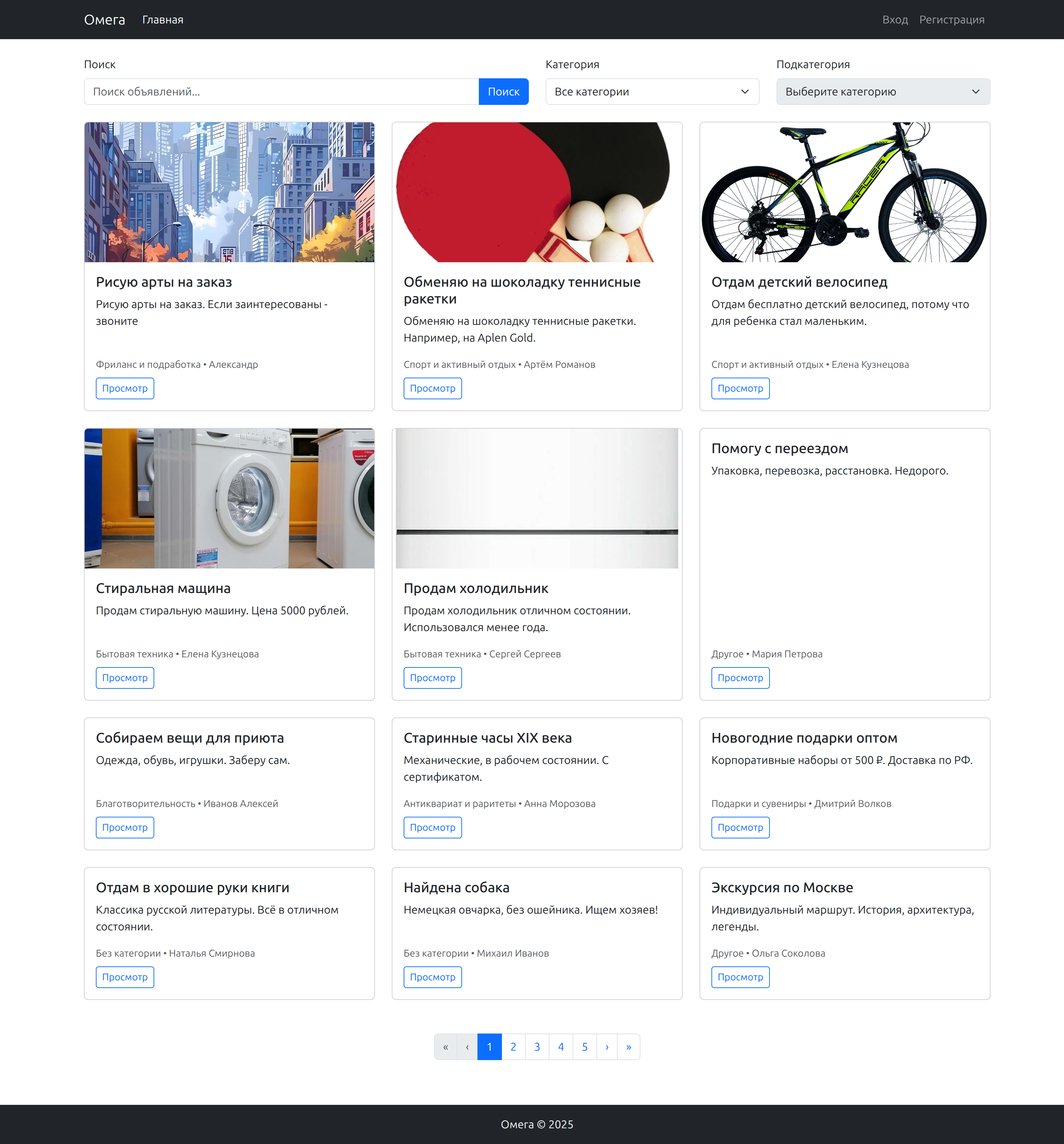


Рисунок 3 – Форма главной страницы

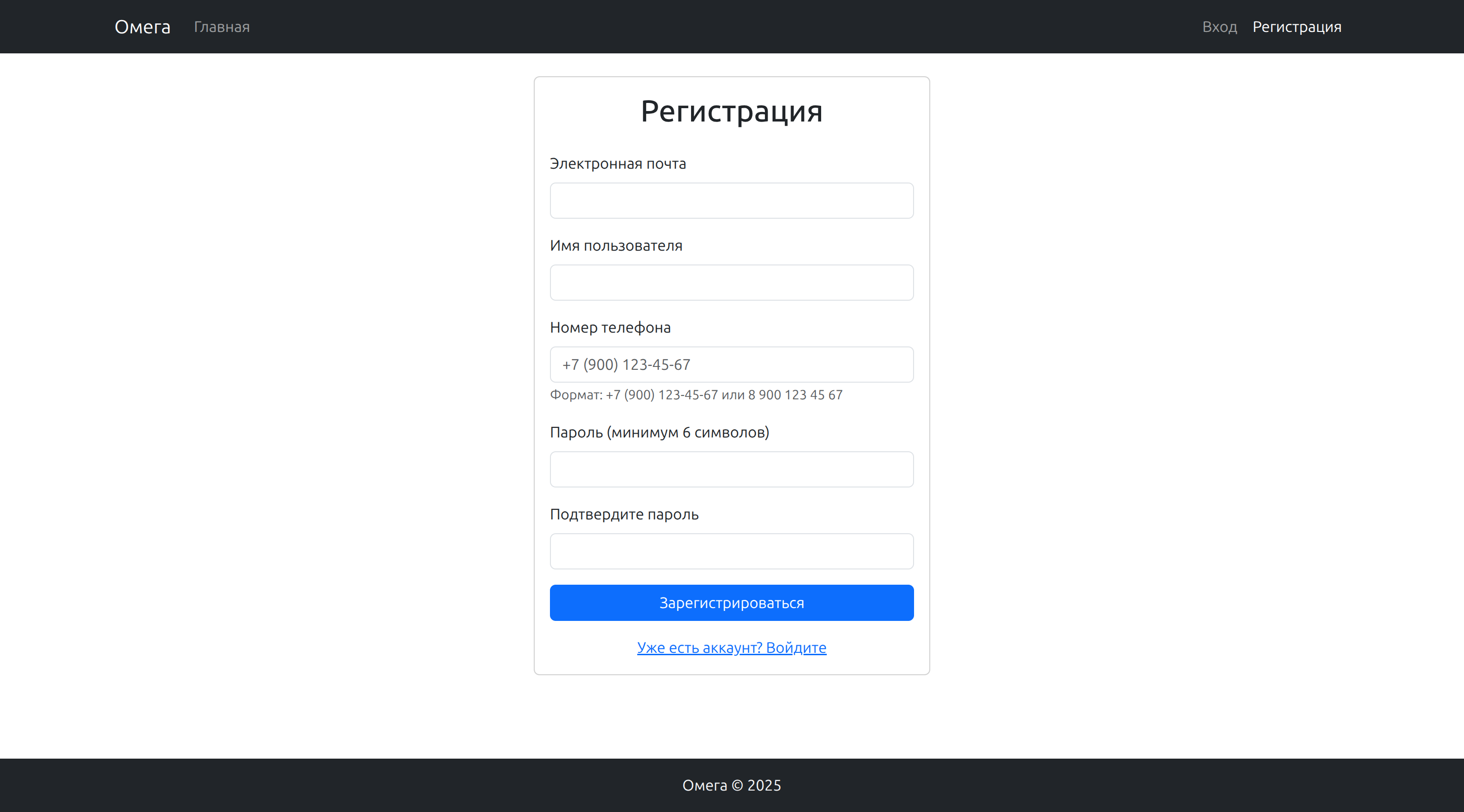


Рисунок 4 – Форма регистрации

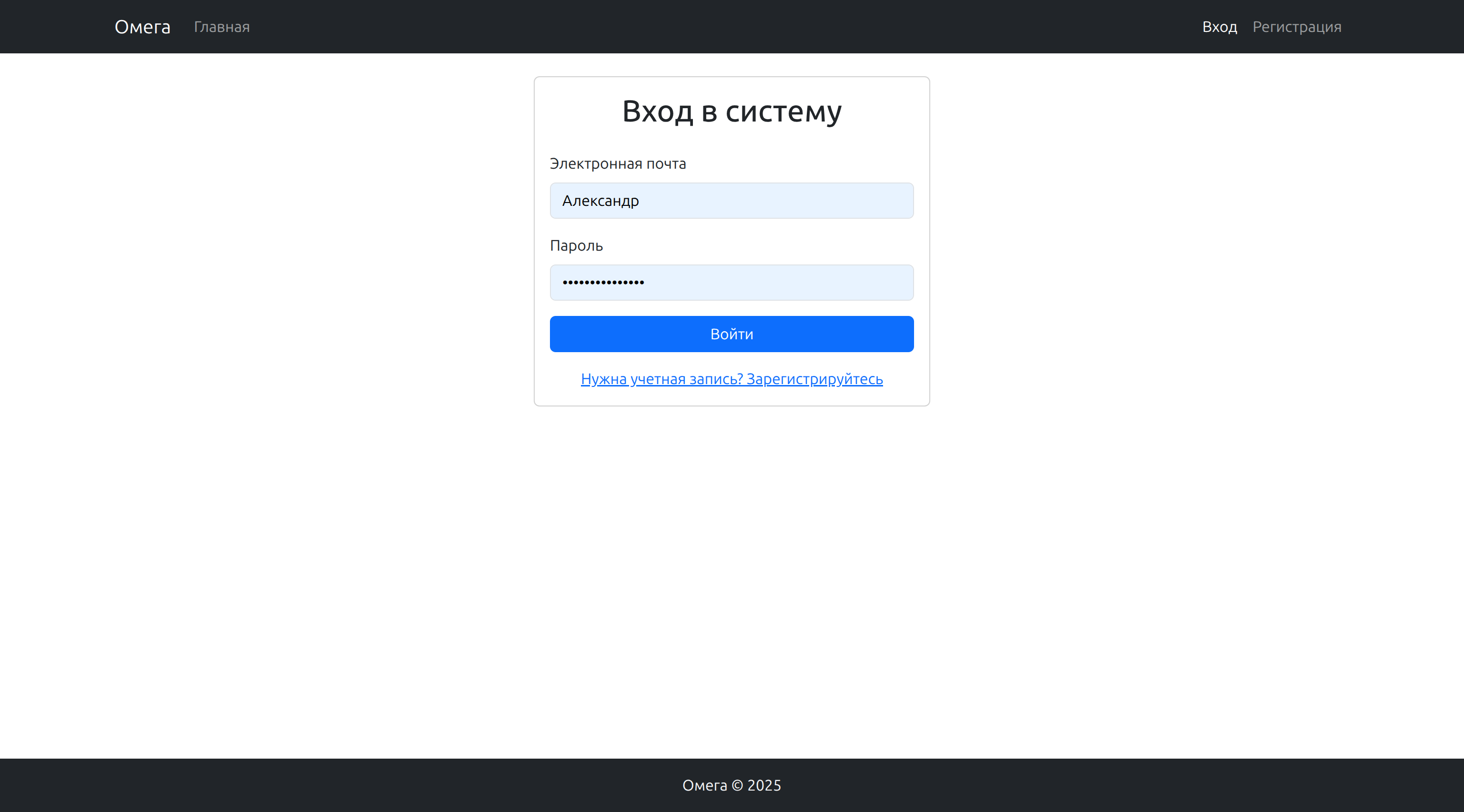


Рисунок 5 – Форма входа в систему

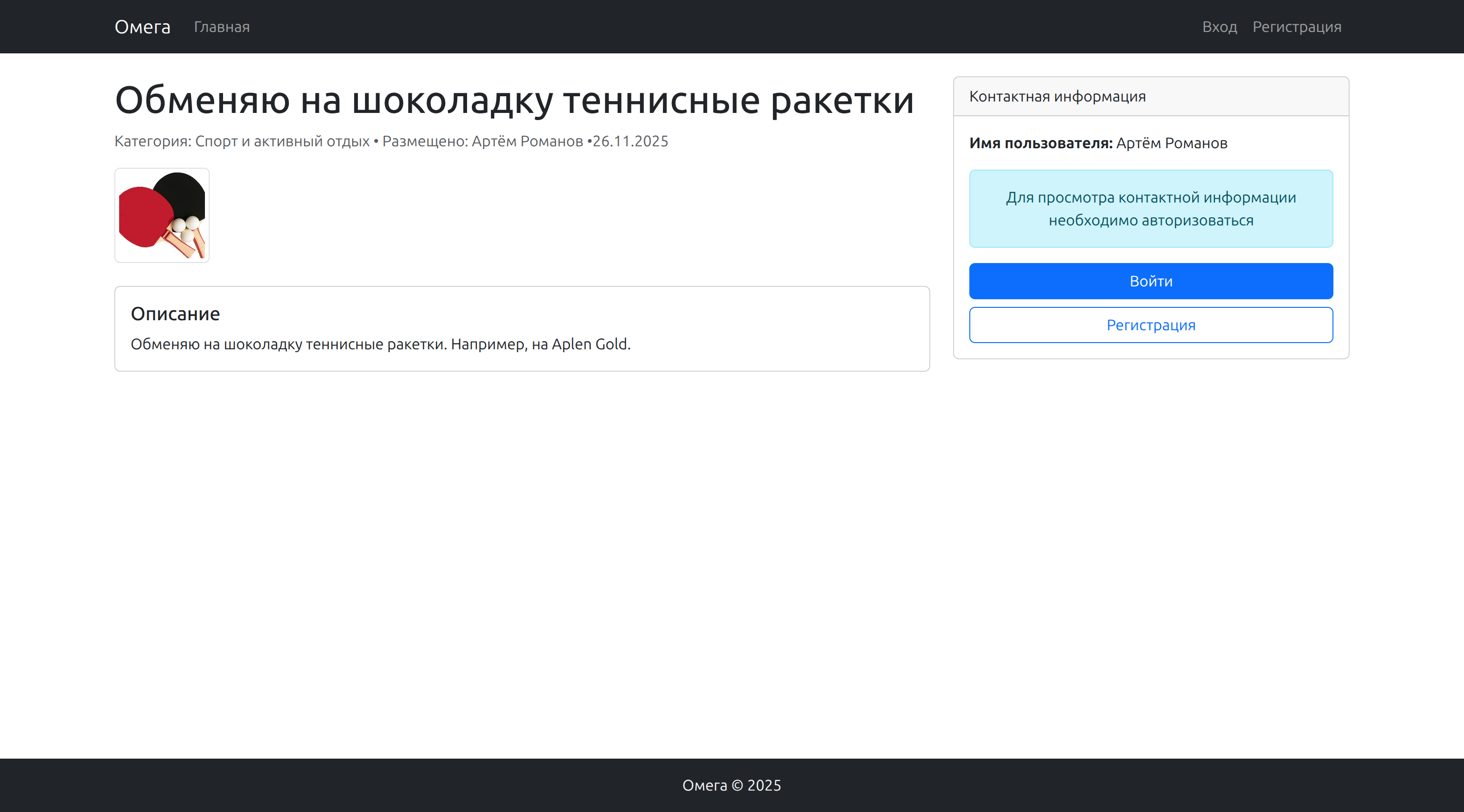


Рисунок 6 – Форма просмотра объявления

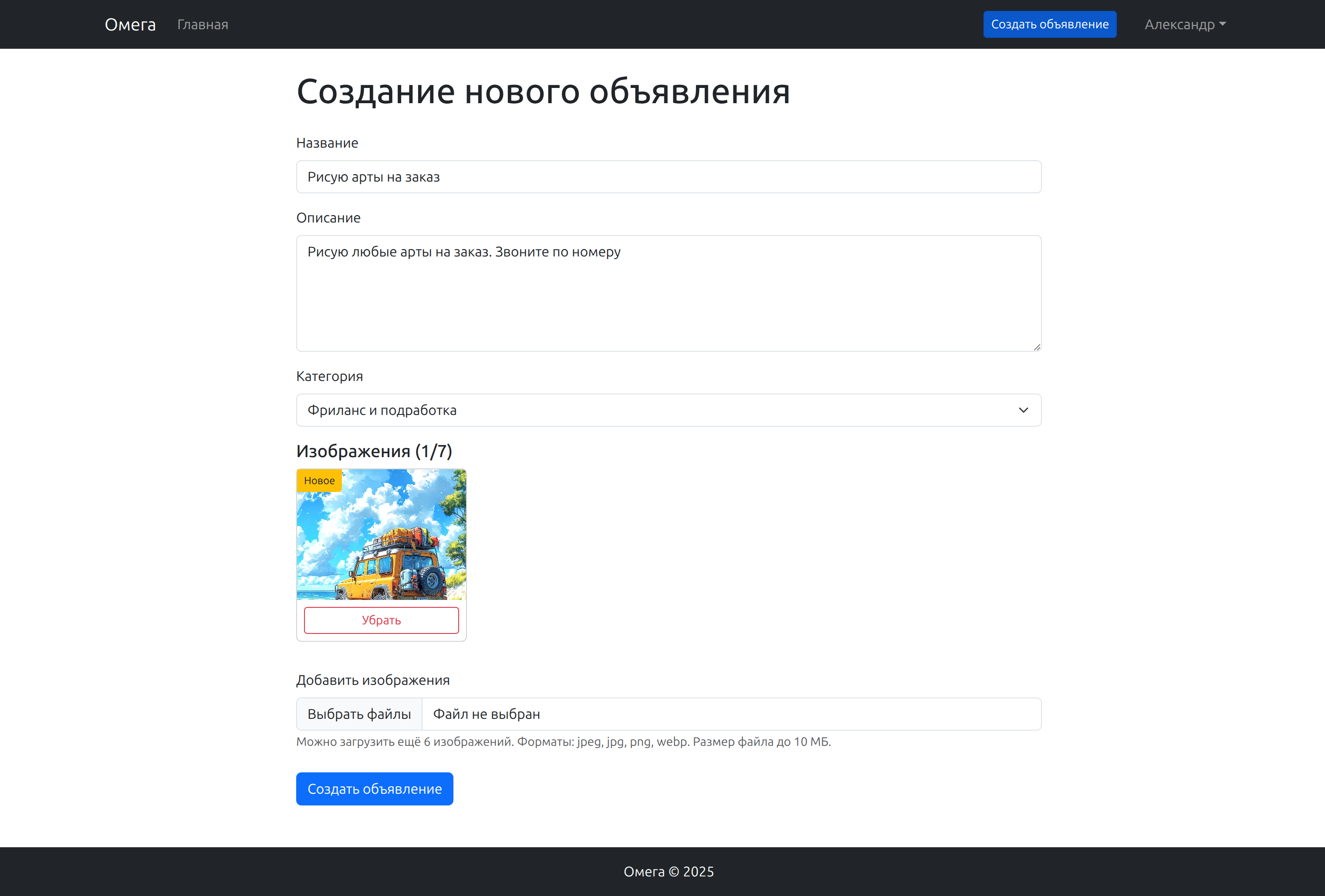


Рисунок 7 – Форма создания объявления



Рисунок 8 – Форма объявлений авторизованного пользователя

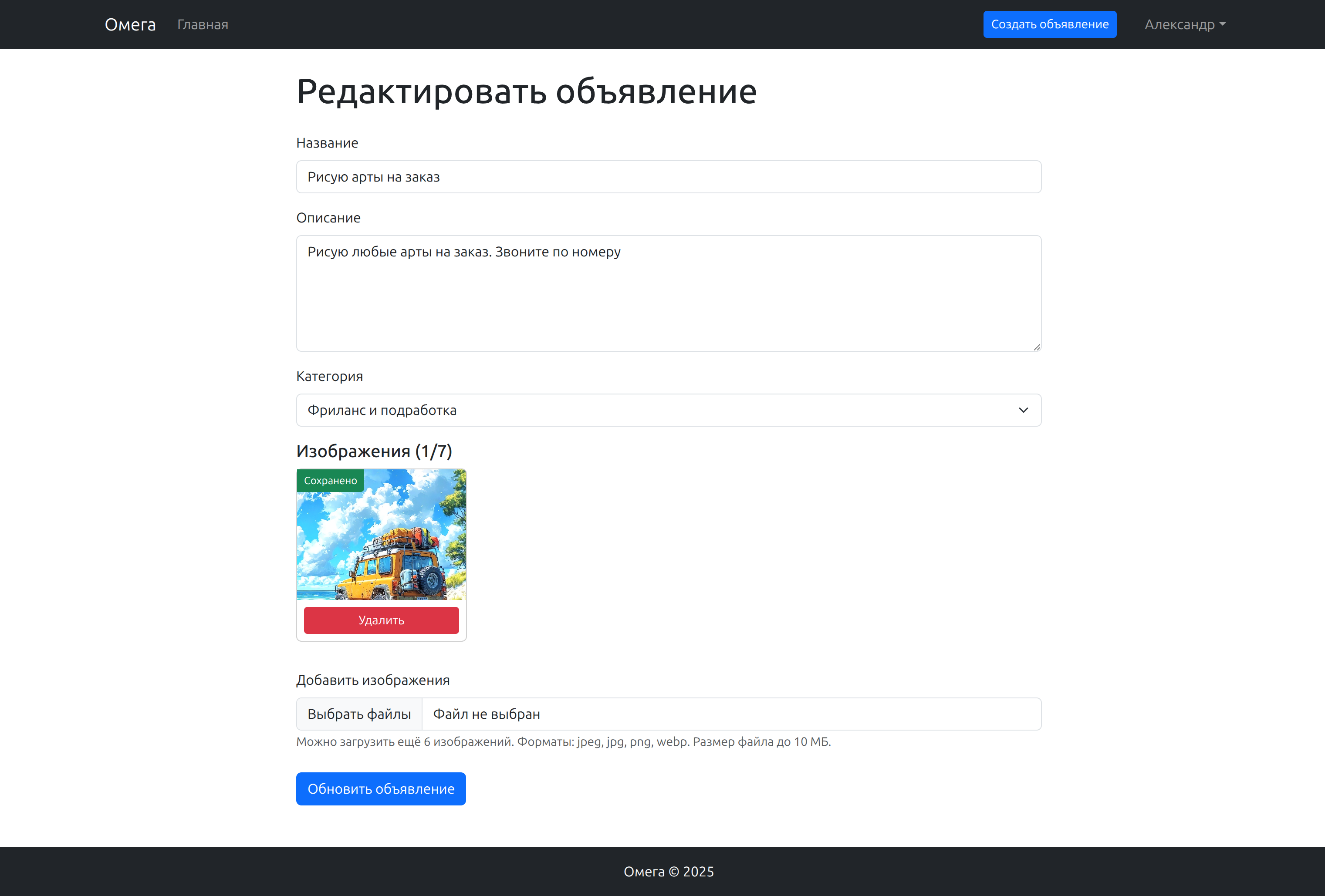


Рисунок 9 – Форма редактирования объявления

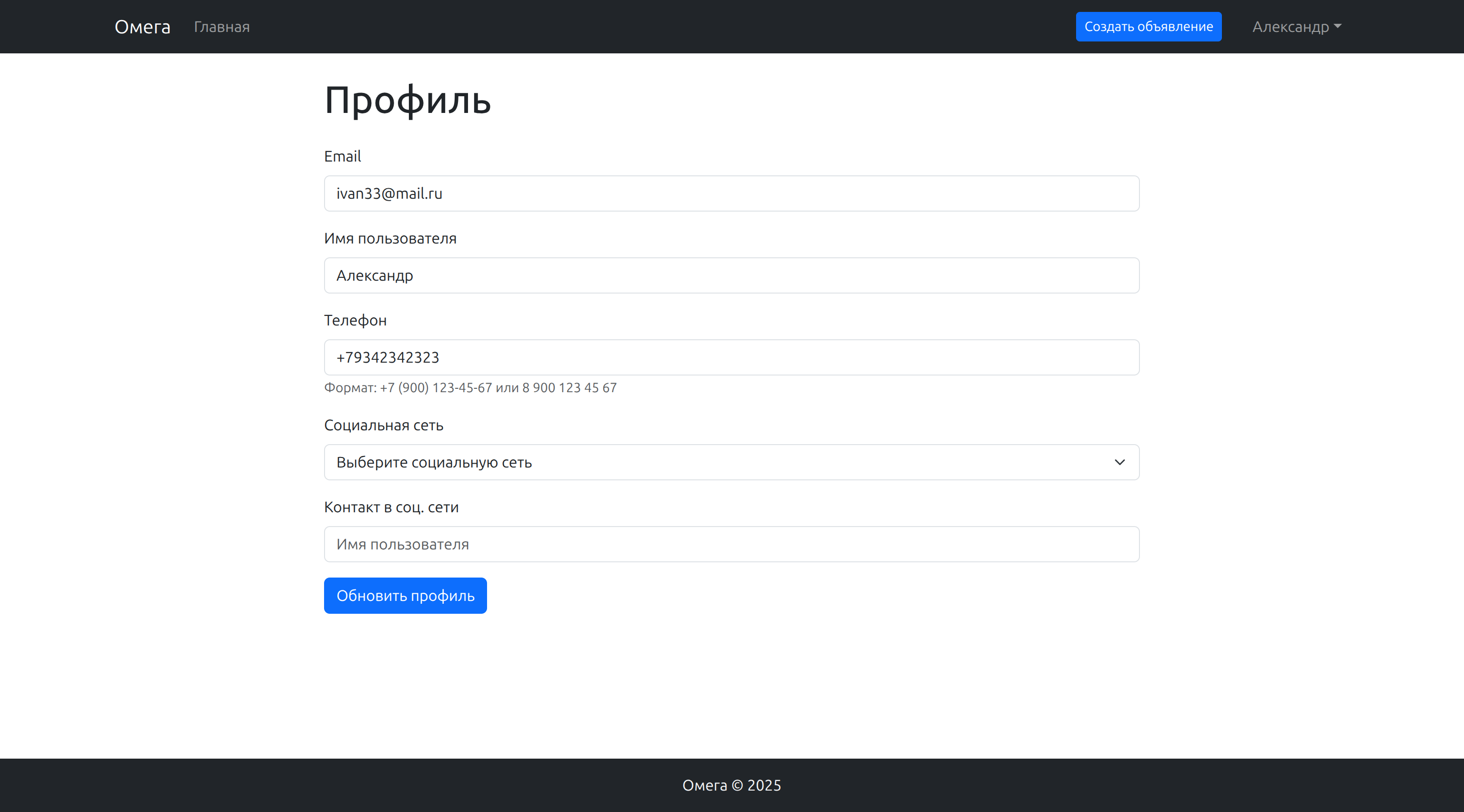


Рисунок 10 – Форма редактирования профиля

## 2.3 Разработка структурной схемы программного продукта

Структурная схема программного продукта отражает состав и взаимодействие компонентов системы. В проекте «Омега» реализована клиент-серверная архитектура, в которой выделены две основные части: клиентская (frontend) и серверная (backend).

Серверная часть реализована на языке Go и построена по принципу чистой архитектуры (Clean Architecture), что позволяет отделить бизнес-логику от деталей реализации (HTTP, база данных). Выделены следующие слои:

* handlers – обрабатывают входящие HTTP-запросы, выполняют первичную валидацию, вызывают сервисы и формируют ответ;
* services – содержат бизнес-логику приложения. Например, логика создания объявления, проверки прав доступа;
* repository – обеспечивают доступ к данным, инкапсулируя взаимодействие с PostgreSQL.

Все зависимости направлены внутрь: Handlers зависят от Services, Services – от Repository.

Клиентская компонента реализована с использованием библиотеки React и состоит из следующих логических блоков:

* pages – компоненты верхнего уровня, представляющие собой отображаемые страницы;
* components – переиспользуемые элементы интерфейса;
* services – модули для взаимодействия с API;
* contexts – управление глобальным состоянием, в частности, AuthContext для хранения токена и данных пользователя.

Фронтенд взаимодействует с бэкендом посредством RESTful API, передавая данные по HTTP в формате JSON и multipart/form-data в случае загрузки файлов.

Для сохранения и получения данных серверная часть обращается к СУБД PostgreSQL.

Полученная схема представлена на рисунке 11.



Рисунок 11 – Структурная схема программного продукта

## 2.4 Проектирование даталогической модели базы данных

Проектирование даталогической моделей базы данных является этапом, на котором структура данных формализуется с учетом требований предметной области. Даталогическая модель конкретизирует сущности, их атрибуты и взаимосвязи в виде таблиц, типов данных и ограничений. Полученная модель представлена на рисунке 12.

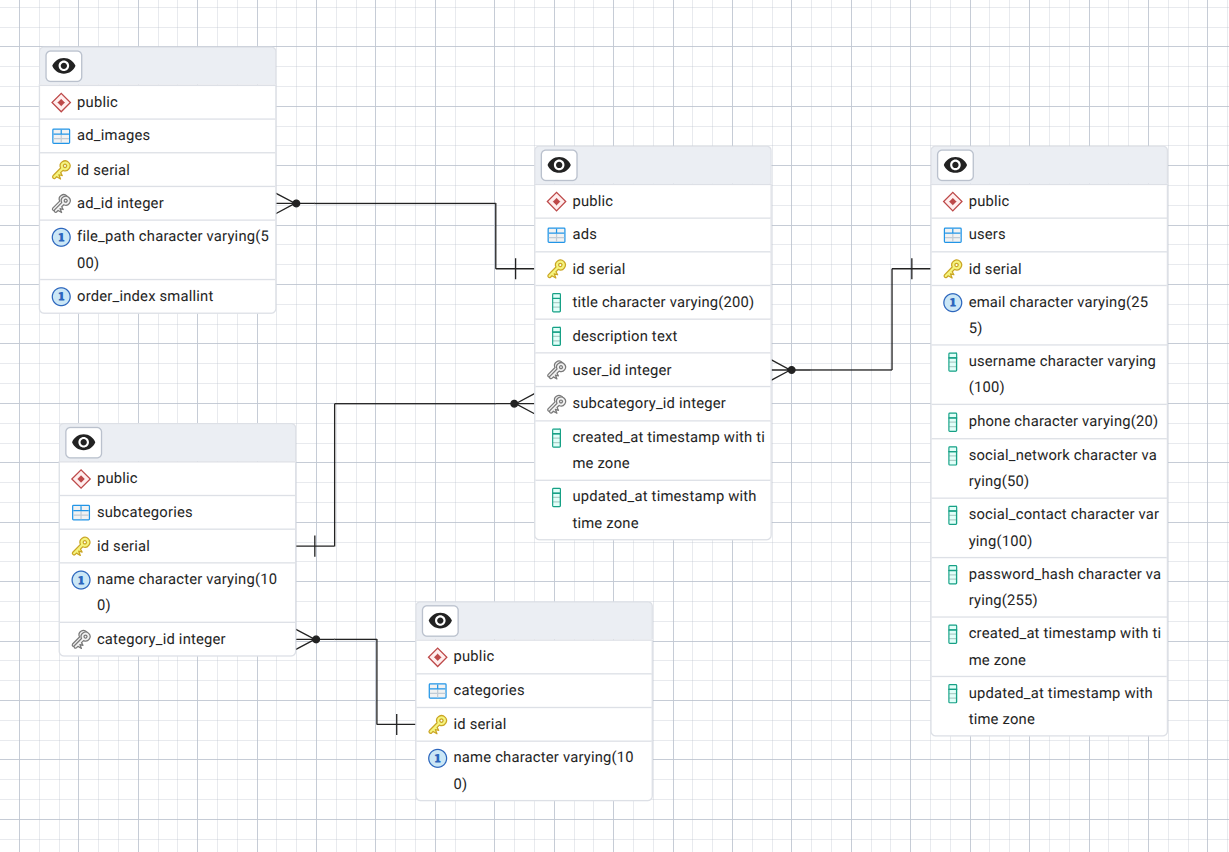


Рисунок 12 – Даталогическая модель базы данных

Главной сущностью системы является объявление (ads). Оно объединяет информацию о пользователе, категории и прикреплённых изображениях, формируя основной контент платформы. Сущности users, categories и subcategories выступают в качестве справочных: они обеспечивают контекст для объявлений (кто разместил, к какой тематике относится). Второстепенная сущность ad\_images детализирует визуальное представление объявления, позволяя прикреплять несколько изображений с учётом порядка отображения.

Нормализация данных достигнута за счёт чёткого разделения ответственности между таблицами и устранения дублирования. Контактные данные (email, телефон, соцсети) хранятся только в users, а не дублируются в каждом объявлении. Категории и подкатегории вынесены отдельно, что позволяет гибко управлять иерархией тематик. Изображения хранятся как отдельные записи в ad\_images, что упрощает их управление и масштабирование.

Ограничения целостности обеспечивают надёжность системы: в каждой таблице определён первичный ключ (PRIMARY KEY), гарантирующий уникальность записей; внешние ключи (FOREIGN KEY) с каскадным удалением автоматически удаляют связанные данные, предотвращая появление «висячих» ссылок; кроме того, наложены уникальные ограничения (UNIQUE) на поля email в таблице users, name в таблицах categories и subcategories, а также на пару полей file\_path и order\_index в таблице ad\_images, что исключает дублирование критически важной информации [4].

Таким образом, спроектированная даталогическая модель обеспечивает эффективное хранение, целостность и масштабируемость данных, полностью соответствующих функциональным и надёжностным требованиям, предъявляемым к веб-приложению «Омега».

## 2.5 Проектирование классов для реализации интерфейса и предметной области

В соответствии с требованиями технического задания, в рамках разработки веб-приложения «Омега» была выполнена детальная декомпозиция системы на компоненты с использованием принципов объектно-ориентированного проектирования. Несмотря на то, что язык Go не поддерживает классы в традиционном смысле, он предоставляет мощные инструменты для реализации ООП-концепций через структуры, методы и интерфейсы [5]. В проекте применена архитектура, основанная на паттерне «Чистая архитектура» (Clean Architecture), где взаимодействие между слоями строго определяется через абстракции, представленные в виде интерфейсов.

На диаграмме (рисунок 13) отражена иерархия компонентов приложения, начиная с главного модуля App, который инициализирует и запускает все основные обработчики.. Каждый обработчик зависит от своего сервиса, например, AuthHandler зависит от AuthService. AuthService в свою очередь, зависит от репозитория, который используется для доступа к данным.

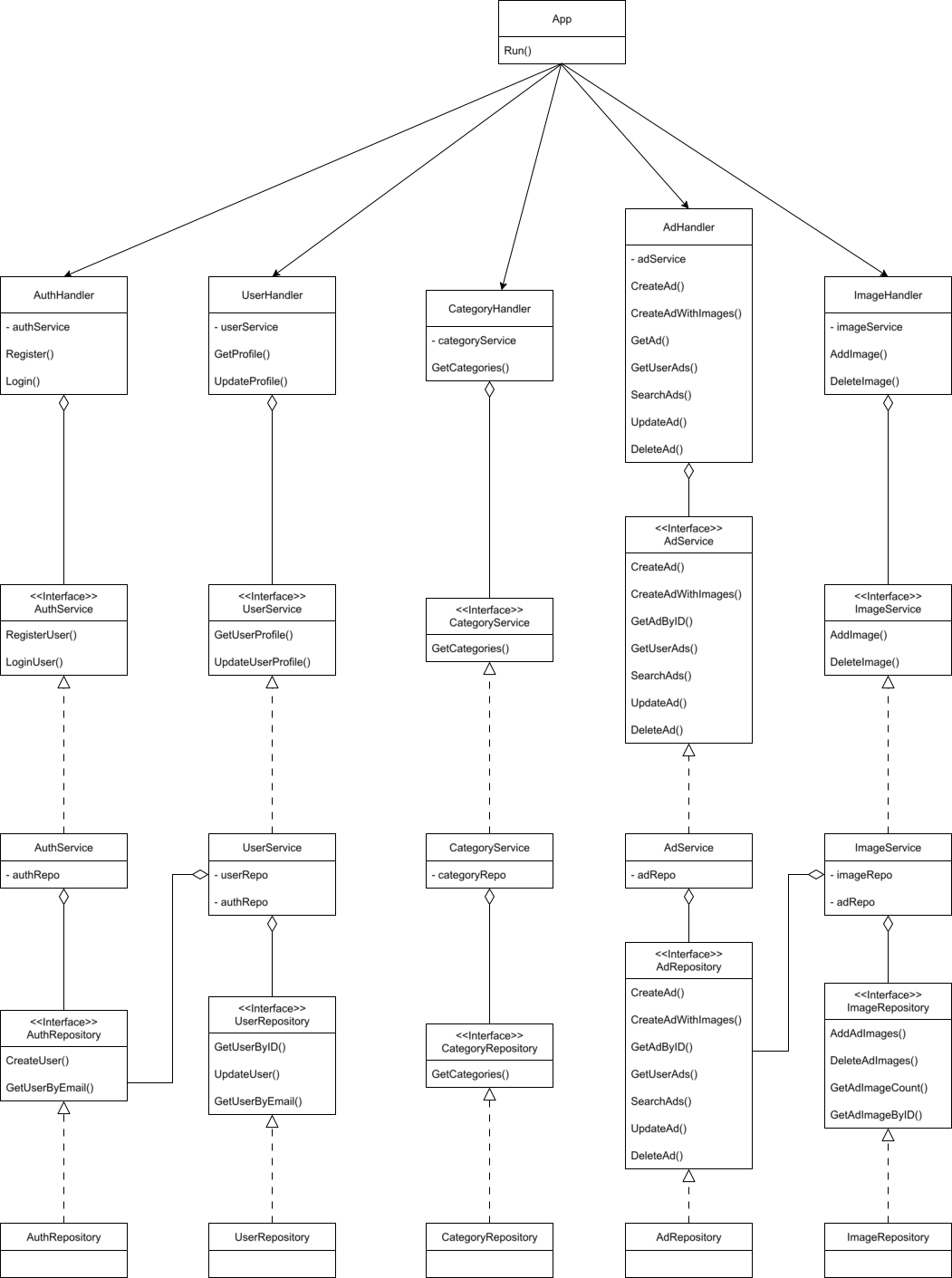


Рисунок 13 – Диаграмма классов

Слой обработчиков отвечает за прием HTTP-запросов, парсинг входных данных, вызов соответствующих методов сервиса и формирование ответа. Например, AdHandler.CreateAd() получает данные из запроса, передаёт их в AdService.CreateAd(), а затем возвращает клиенту JSON-ответ или ошибку.

Слой сервисов реализует бизнес-логику приложения. Здесь происходит валидация данных, проверка прав доступа, логика создания объявления с изображениями и т.д.

Слой репозиториев осуществляет работу с базой данных. Каждый репозиторий реализует свой интерфейс и содержит методы для выполнения CRUD-операций и сложных запросов.

Все зависимости между слоями определены через интерфейсы, что обеспечивает слабую связанность и упрощает тестирование. Например, AdService зависит не от конкретной реализации AdRepository, а от интерфейса AdRepository, что позволяет легко заменить реализацию.

Модуль App служит точкой входа и координатором всех компонентов. Он создаёт экземпляры сервисов и репозиториев, внедряет зависимости и запускает HTTP-сервер.

Таким образом, данная архитектура обеспечивает:

* масштабируемость – добавление новых функций не требует изменения существующего кода;
* тестируемость – возможность изолированного тестирования каждого слоя;
* поддерживаемость – чёткое разделение ответственности и слабая связанность между компонентами.

## 2.6 Разработка диаграммы последовательности действий

Для уточнения взаимодействия компонентов системы при выполнении успешной операции создания объявления с прикреплёнными изображениями была разработана диаграмма последовательности действий.

Выбор именно этого сценария обусловлен его комплексностью: он включает в себя аутентификацию пользователя, валидацию входных данных, загрузку файлов, сохранение метаданных в базу данных и формирование ответа клиенту. Это позволяет продемонстрировать взаимодействие всех основных слоёв архитектуры. Диаграмма представлена на рисунке 14.

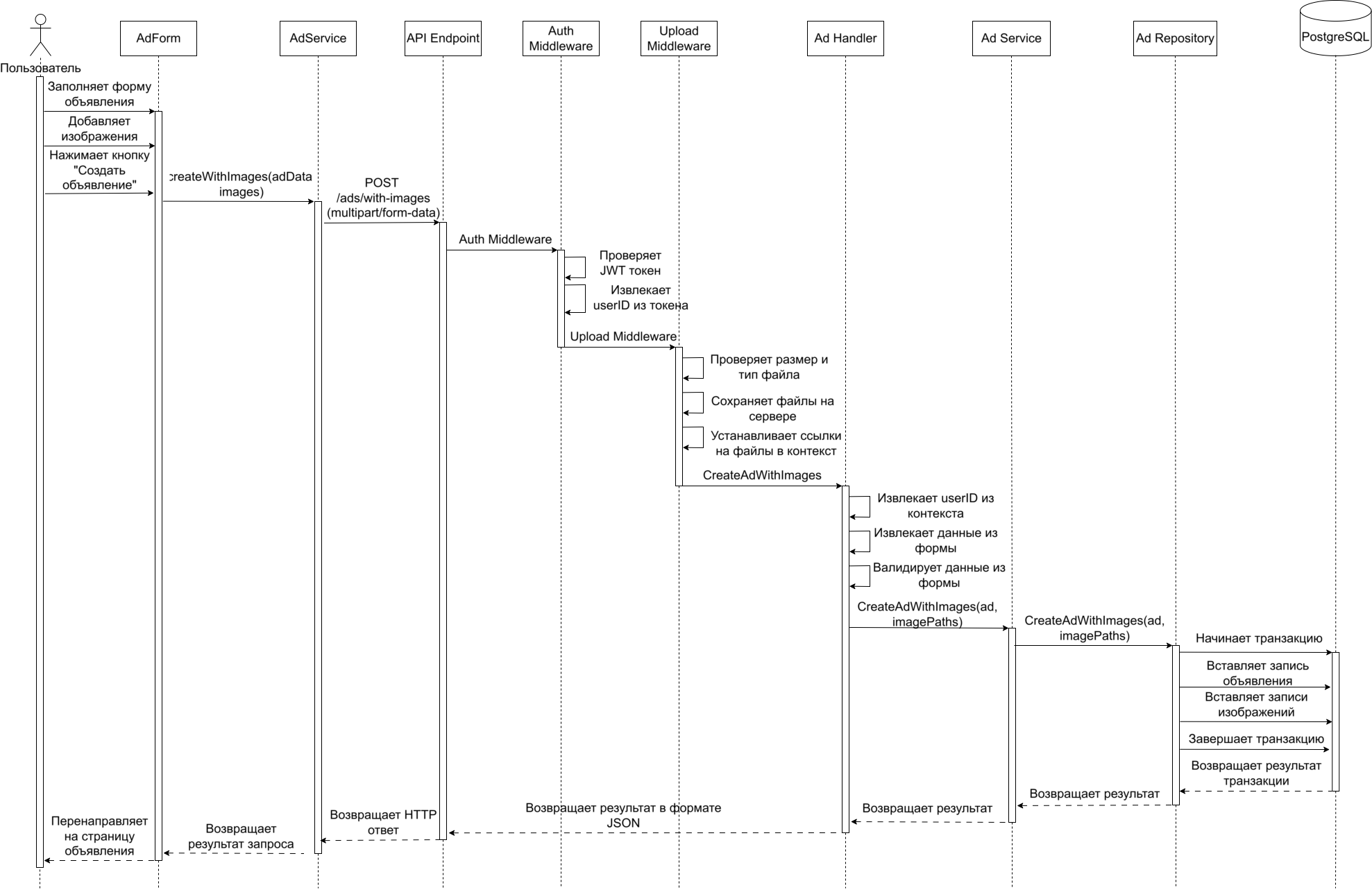


Рисунок 14 – Диаграмма последовательности действий при создании объявления с изображениями

Диаграмма иллюстрирует следующую логику. Пользователь, находясь на странице «Создать объявление», заполняет форму, прикрепляет файлы и нажимает кнопку «Создать объявление». Далее запрос отправляется к сервису, который отправляет HTTP запрос серверу. На сервере запрос проходит через промежуточные обработчики, которые проверяют авторизацию и валидность файлов. Далее в обработчике происходит валидация данных и их передача в слой бизнес логики, которая в свою очередь передаёт запрос слою работы с базой данных. Он формирует транзакцию и вставляет данные. После этого в обратном порядке возвращаются ответы. При этом обработчик напрямую возвращает ответ API Endpoint, минуя промежуточные обработчики (middleware).

# 3 Выбор стратегии тестирования и разработка тестов

Для проверки соответствия реализации функциональным требованиям ТЗ была выбрана стратегия функционального (чёрного ящика) тестирования. Цель – убедиться, что система корректно реагирует на входные данные и выполняет заявленные функции независимо от внутренней реализации.

В таблице 1 приведены тестовые сценарии, охватывающие ключевые пользовательские действия.

Таблица 1 – Таблица функциональных тестов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Условие теста | Значение исходных данных | Ожидаемый результат | Полученный результат | Статус теста |
| 1 | Тестирование регистрации пользователя с валидными данными | Логин: user123@mail.ru,  имя: Иван,  номер телефона: +79523423423  пароль: password123 | Успешная регистрация | Успешная регистрация | Тест пройден |
| 2 | Тестирование регистрации пользователя с ошибкой в данных | Логин: user123@mail.ru,  имя: Иван,  номер телефона: +795234234  пароль: password123 | Ошибка валидации данных и сообщение об этом | Ошибка валидации и отображение ошибки «Неверный формат номера телефона (пример: +7 (900) 123-45-67)» | Тест пройден |
| 3 | Тестирование корректной авторизации пользователя | Логин: user123@mail.ru,  пароль: password123 | Успешная авторизация | Успешная авторизация | Тест пройден |
| 4 | Нажатие на кнопку «Просмотр» у объявления | Объявление с названием «Стиральная машина», заданным описанием и картинкой | Открытие страницы объявления | Открытие страницы объявления и отображением соответствующей ему информации | Тест пройден |

Продолжение таблицы 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | Тестирование создания объявления с картинками | Название: Наручные часы,  описание: Стильные мужские часы на металлическом браслете,  Категория: Аксессуары  Файл: watches.jpg (размер 131,4 КБ) | Успешное создание объявления | Успешное создание объявления и открытие страницы созданного объявления с заданными данными, включая картинку | Тест пройден |
| 6 | Тестирование редактирования объявления | Изменение название объявления на «Продаю наручные часы» и добавление файла watches1.jpg (размер 61 КБ) | Успешное редактирование объявления | Успешное редактирование объявления и открытие страницы отредактированного объявления с заданными данными, включая новую картинку | Тест пройден |
| 7 | Тестирование поиска объявления | Значение поиска: Продаю наручные часы | Найдено ранее созданное объявление | Найдено единственное объявление, которое было создано раннее | Тест пройден |
| 8 | Тестирование удаления объявления | Удаления объявления с названием «Продаю наручные часы» | Объявление удалено | Объявление успешно удалено | Тест пройден |
| 9 | Тестирование редактирования профиля | Социальная сеть: Telegram,  Контакт в соц. сети: @abacaba | Профиль успешно обновлён | Данные профиля успешно обновлены | Тест пройден |

# Заключение

В ходе выполнения курсовой работы был спроектирован и реализован веб-сайт «Омега», предназначенный для размещения и поиска объявлений. Были реализованы все функции, указанные в техническом задании: регистрация, авторизация, создание, редактирование и удаление объявлений, поиск по заголовку, загрузка изображений. Применена клиент-серверная архитектура, обеспечена надёжность ввода, целостность данных и безопасность доступа. Проведено функциональное тестирование, подтвердившее корректность работы системы. Все этапы работы сопровождались диаграммами для наглядного представления архитектуры, логики взаимодействия компонентов и структуры пользовательского интерфейса.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация к React [Электронный ресурс]. URL:<https://react.dev/learn> (дата обращения: 20.11.2025)
2. Документация к библиотеке GORM [Электронный ресурс]. URL:<https://gorm.io> (дата обращения: 20.11.2025)
3. Фомин М.М. Реляционные базы данных. Учебное пособие для бакалавров: МГТУ имени Н.Э. Баумана, 2023. 161 с.
4. Документация PostgreSQL и Postgres Pro [Электронный ресурс]. – URL: <https://postgrespro.ru/docs> (дата обращения 20.11.2025).
5. Effective Go: официальное руководство по написанию идиоматического кода на языке Go [Электронный ресурс]. URL:<https://go.dev/doc/effective_go> (дата обращения: 20.11.2025)

# **ПРИЛОЖЕНИЕ А. Техническое задание**

# **ПРИЛОЖЕНИЕ Б. Руководство пользователя**

**ПРИЛОЖЕНИЕ В. Исходный текст модуля App (2-3 страницы) и ссылка на местоположение полного комплекта исходных модулей программы**