|  |  |
| --- | --- |
|  | **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 - ИУ6**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

***К КУРСОВОЙ РАБОТЕ***

***по дисциплине «Технология разработки программных систем»***

***НА ТЕМУ:***

Сайт для размещения электронных объявлений

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-51Б |  |  | Д.Ю. Воронин |
|  | Группа |  | Подпись, дата | И.О. Фамилия |
|  |  |  |  |  |
| Преподаватель |  |  |  | Д.А. Миков |
|  |  |  | Подпись, дата | И.О. Фамилия |

Москва, 2025

**РЕФЕРАТ**

Расчетно-пояснительная записка состоит из Х страниц и включает в себя X рисунков, X листингов, X источника.

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 проектирования, клиент-серверной архитектуры, а также инструменты автоматизации, такие как Docker и миграции базы данных. 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](#_Toc196573099)

[1. Анализ предметной области, конкретизация задач исследования 5](#_Toc196573100)

[2. Разработка бизнес-процессов предметной области 8](#_Toc196573101)

[3. Проектирование инфологической и даталогической модели БД 11](#_Toc196573102)

[4. Заполнение таблиц 15](#_Toc196573103)

[5. Разработка запросов 31](#_Toc196573104)

[Заключение 39](#_Toc196573105)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 40](#_Toc196573106)

# 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 эффективно 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 локальных продавцов и покупателей.

Актуальность разработки собственной платформы «Омега» обусловлена необходимостью предоставления упрощённого и экономически выгодного решения для локального рынка. 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.

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[1]. 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.

Frontend разработан с использованием React – современной JavaScript-библиотеки для построения пользовательских интерфейсов. React позволяет создавать модульные, переиспользуемые компоненты и обеспечивает эффективное обновление интерфейса. Стилизация выполнена с использованием библиотеки Bootstrap, что позволяет разрабатывать интерфейсы быстрее, чем на чистом CSS, сохраняя при этом современный и аккуратный вид приложения.

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.

Для развёртывания и изоляции окружения используется Docker и docker-compose, что обеспечивает идемпотентность среды разработки и упрощает дальнейшее размещение приложения на сервере.

Выбранный технологический стек (Go + React + PostgreSQL + Docker) 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.

## **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.

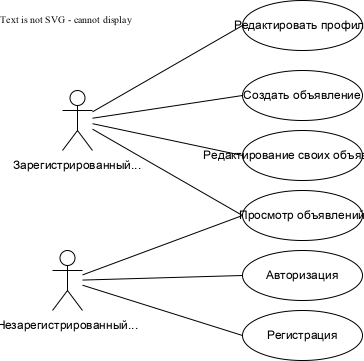


Рисунок 1 – Диаграмма вариантов использования

## **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, 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.

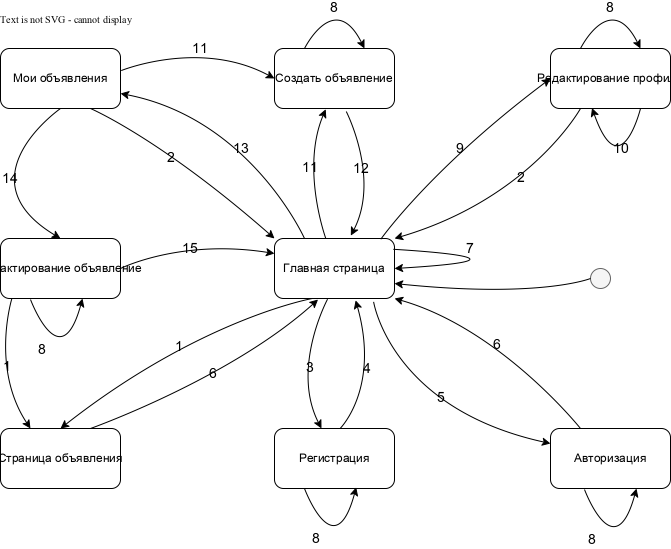


Рисунок 2 – Диаграмма состояний интерфейса

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.

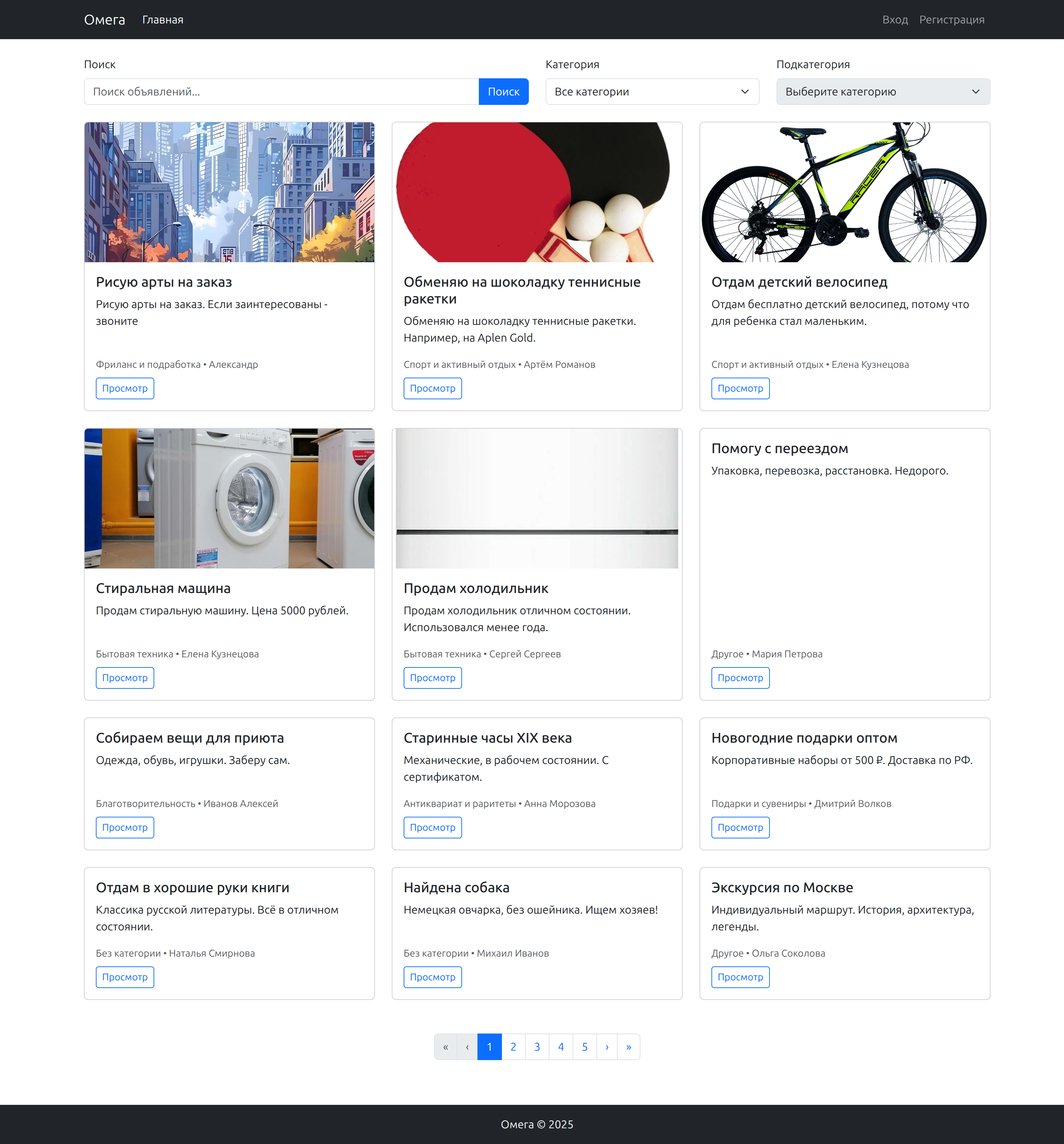


Рисунок 3 – Форма главной страницы

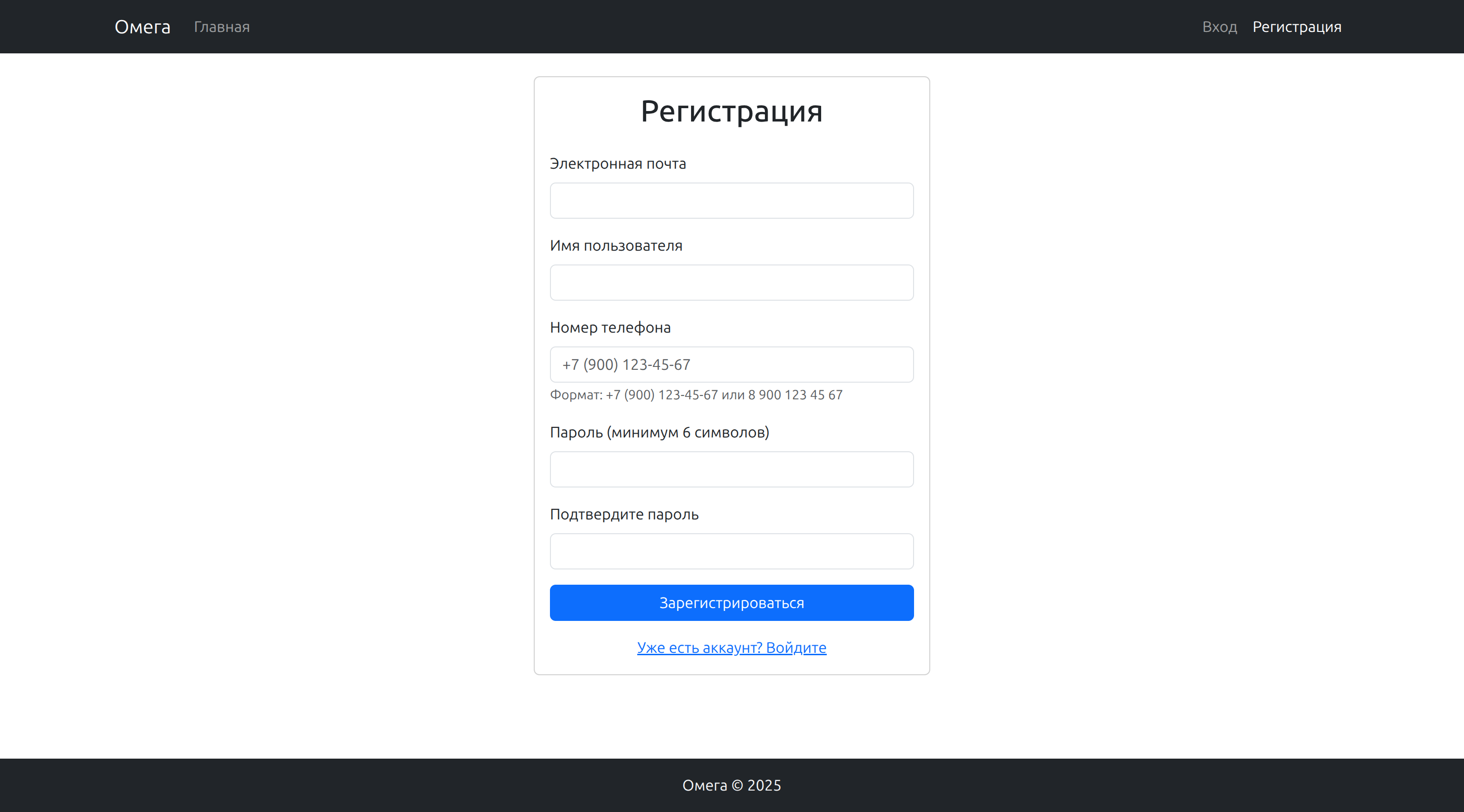


Рисунок 4 – Форма регистрации

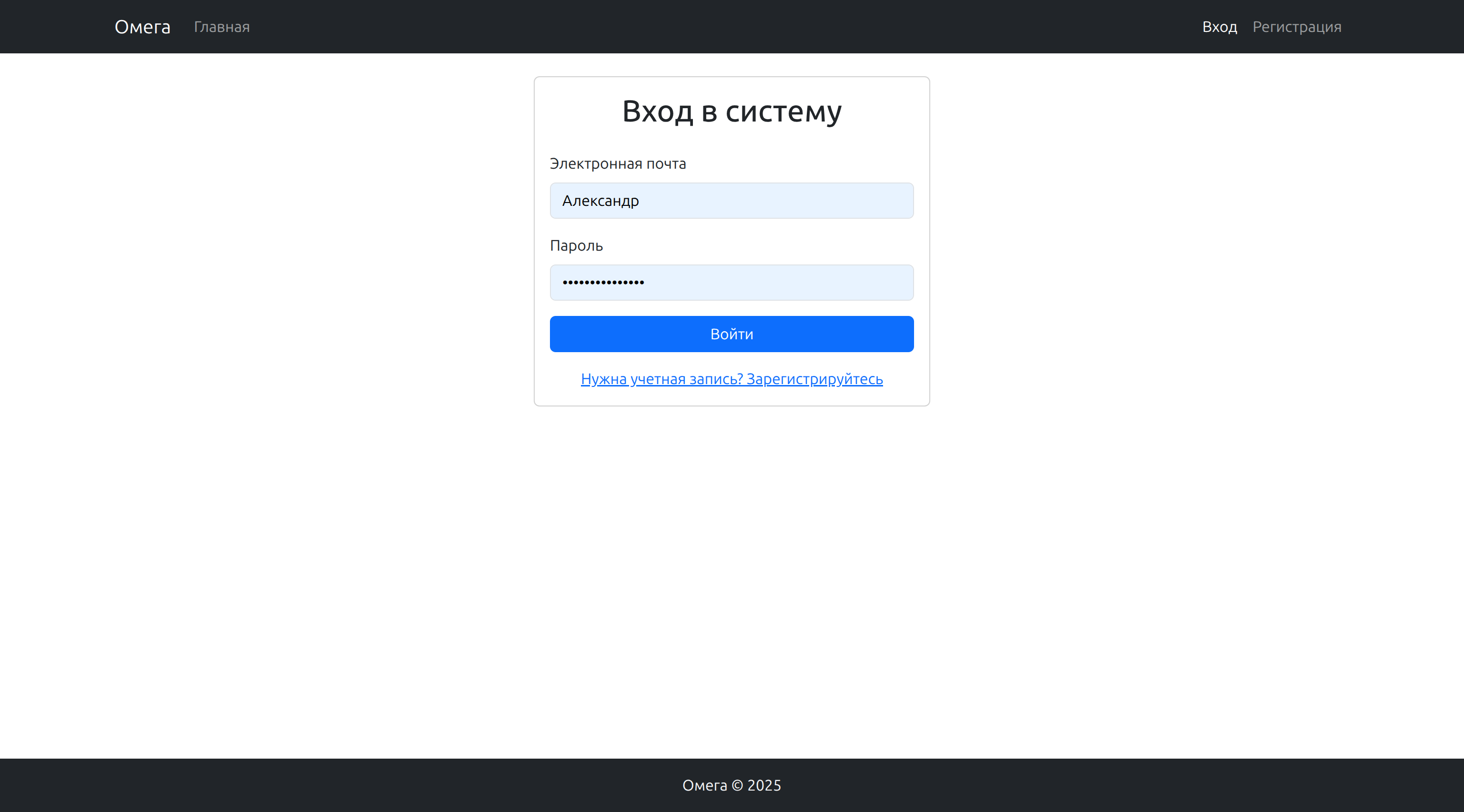


Рисунок 5 – Форма входа в систему

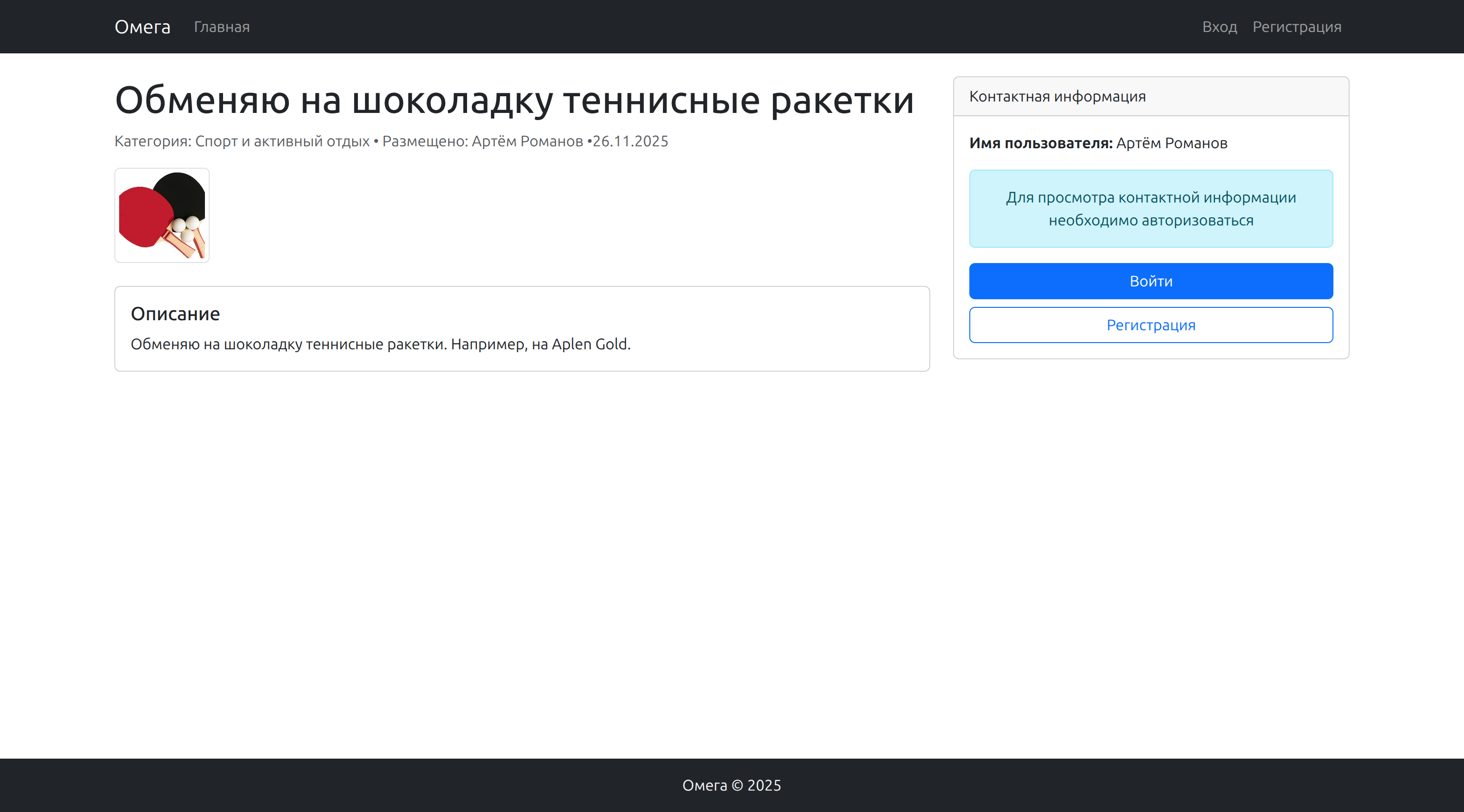


Рисунок 6 – Форма просмотра объявления

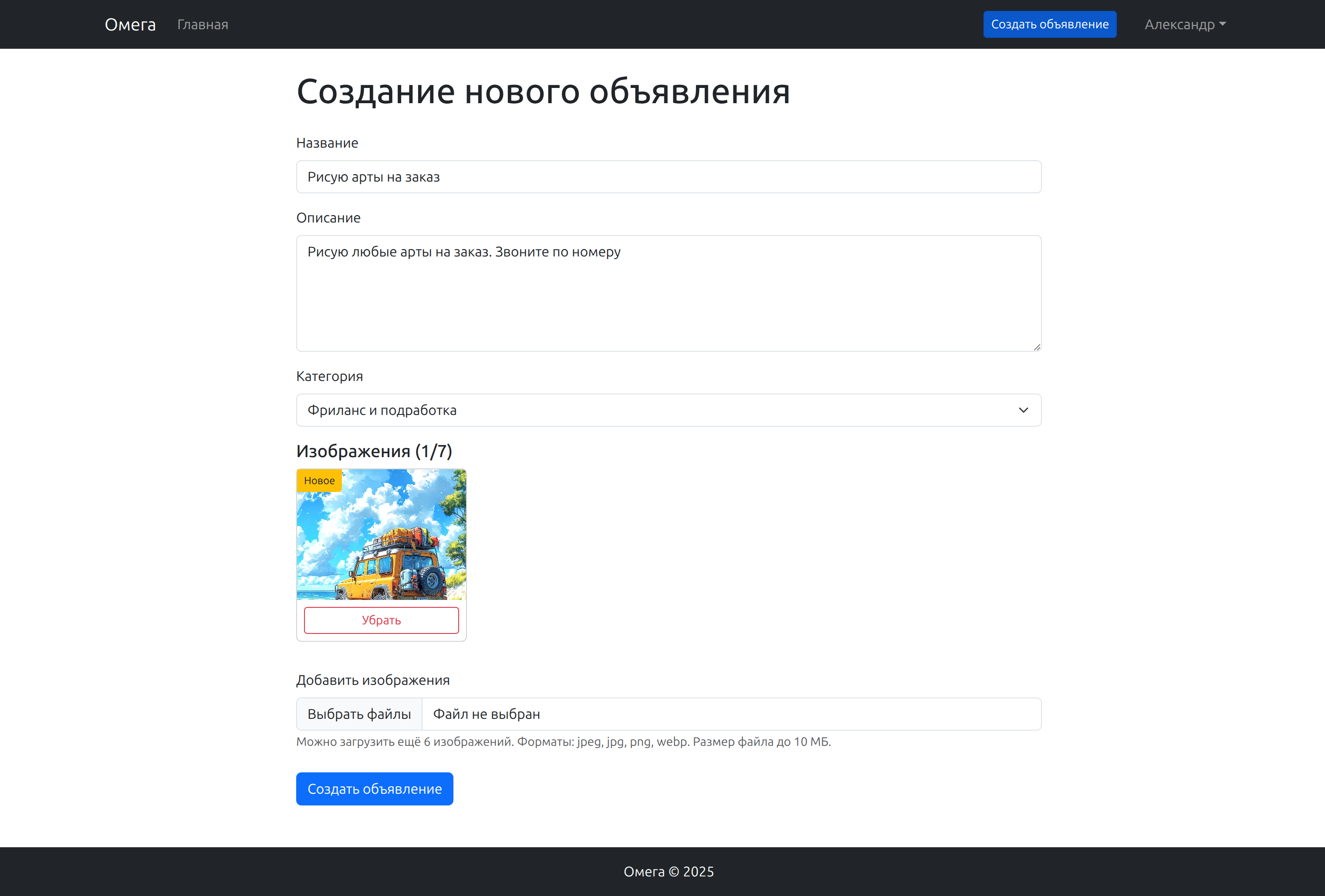


Рисунок 7 – Форма создания объявления



Рисунок 8 – Форма объявлений авторизованного пользователя

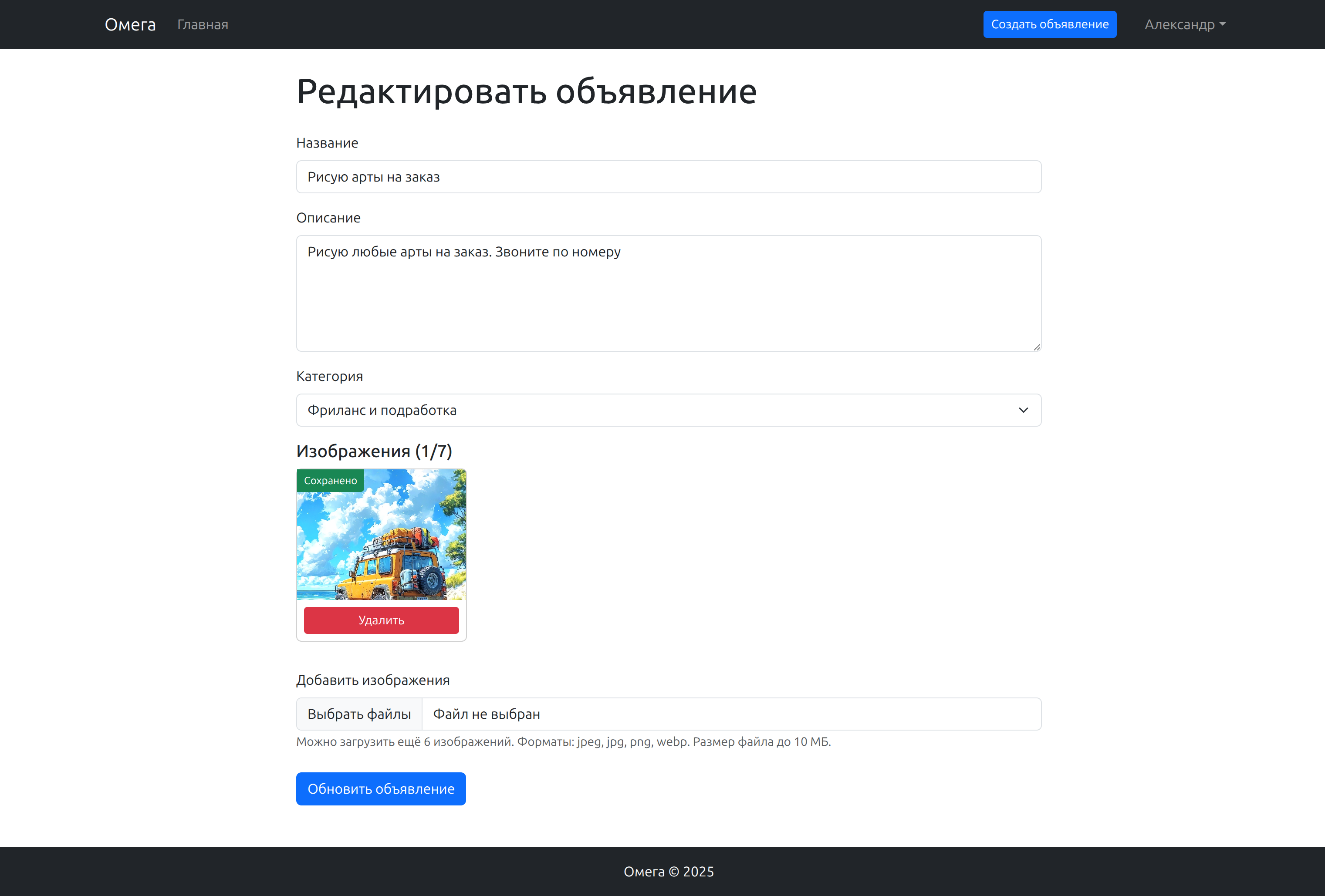


Рисунок 9 – Форма редактирования объявления

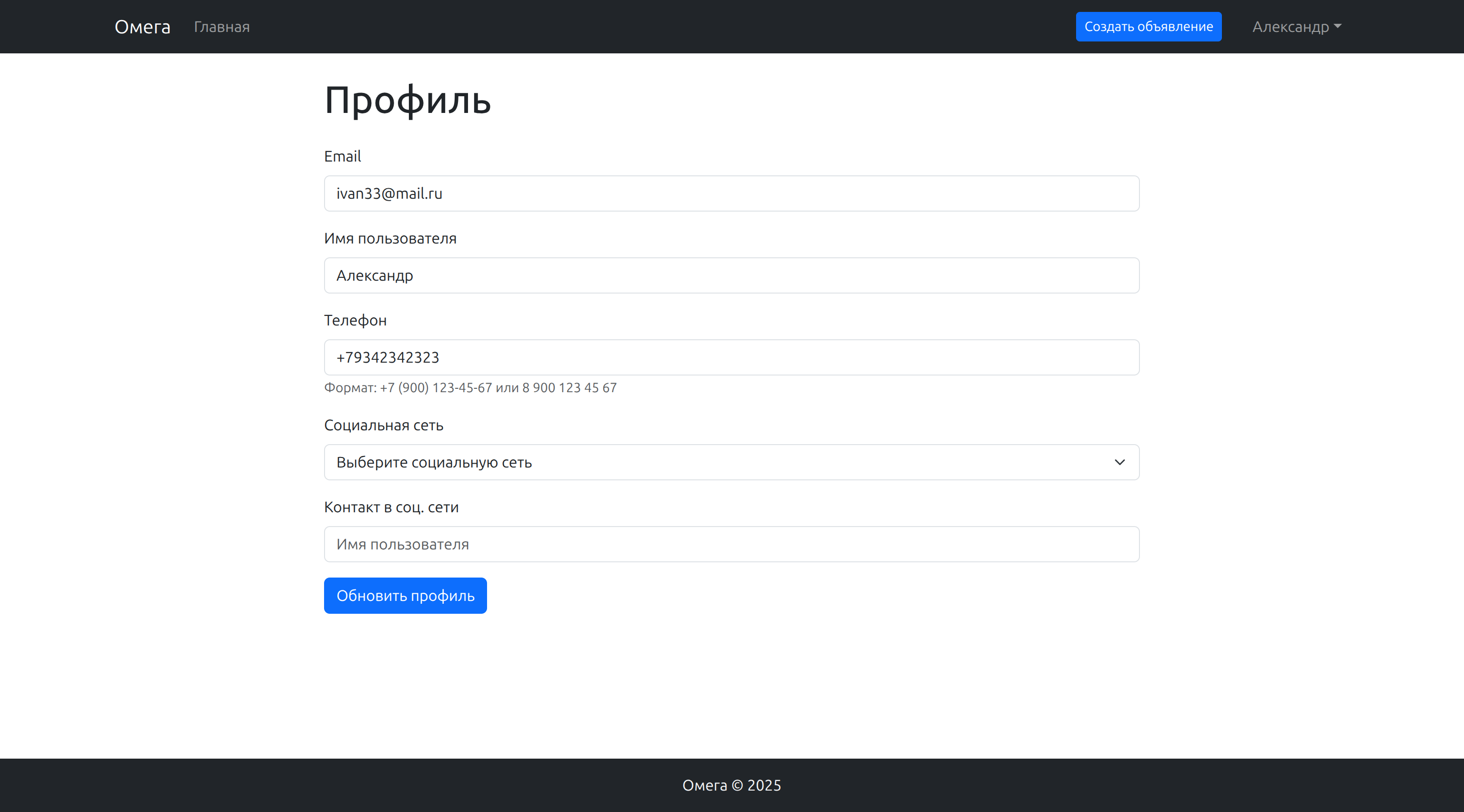


Рисунок 10 – Форма редактирования профиля

## 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 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.

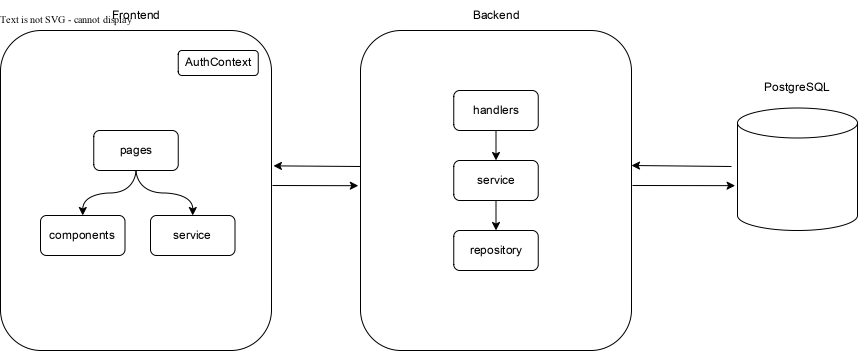


Рисунок 11 – Структурная схема программного продукта

## 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.

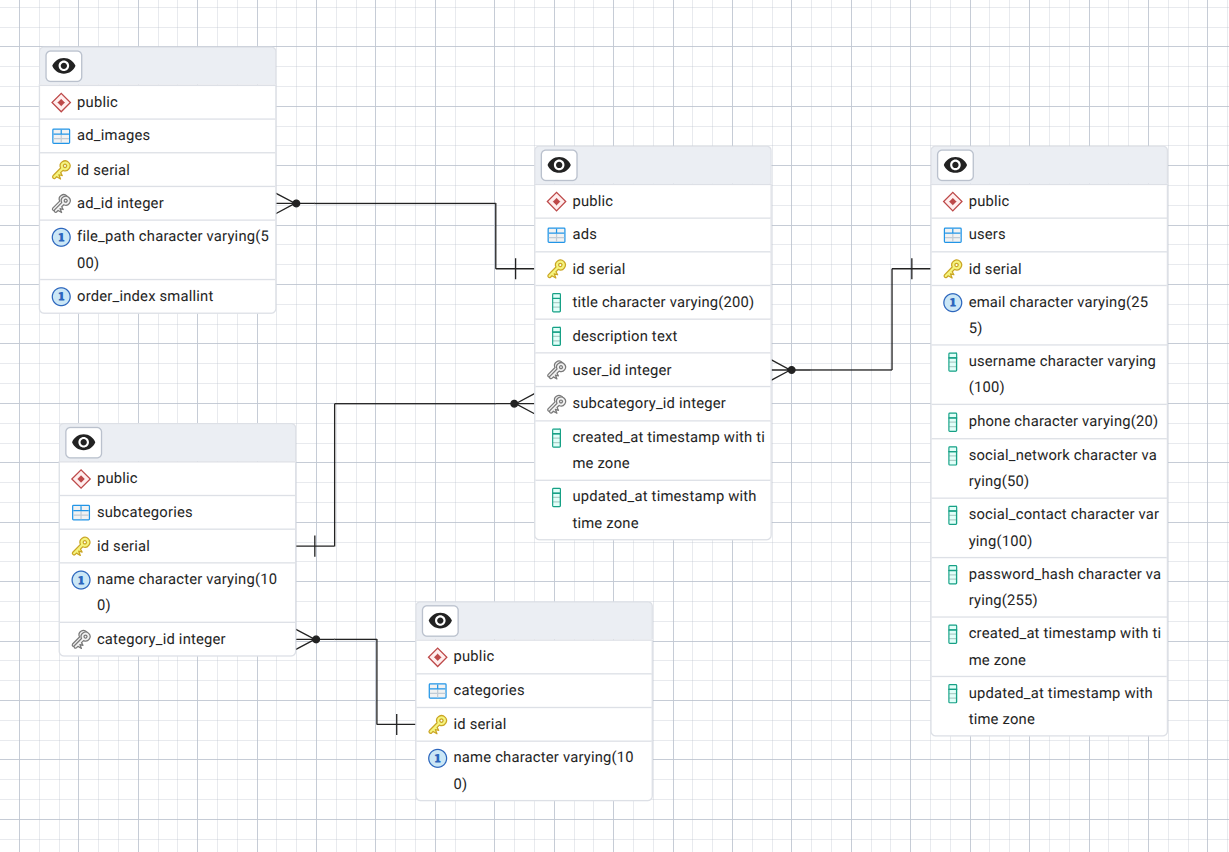


Рисунок 12 – Даталогическая модель базы данных

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:

* PRIMARY KEY в каждой таблице гарантирует уникальность записей;
* FOREIGN KEY с каскадным удалением обеспечивает автоматическую очистку зависимых данных;
* UNIQUE-ограничения на email в users, name в categories и в subcategories и file\_path и order\_index в ad\_images.

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. 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.

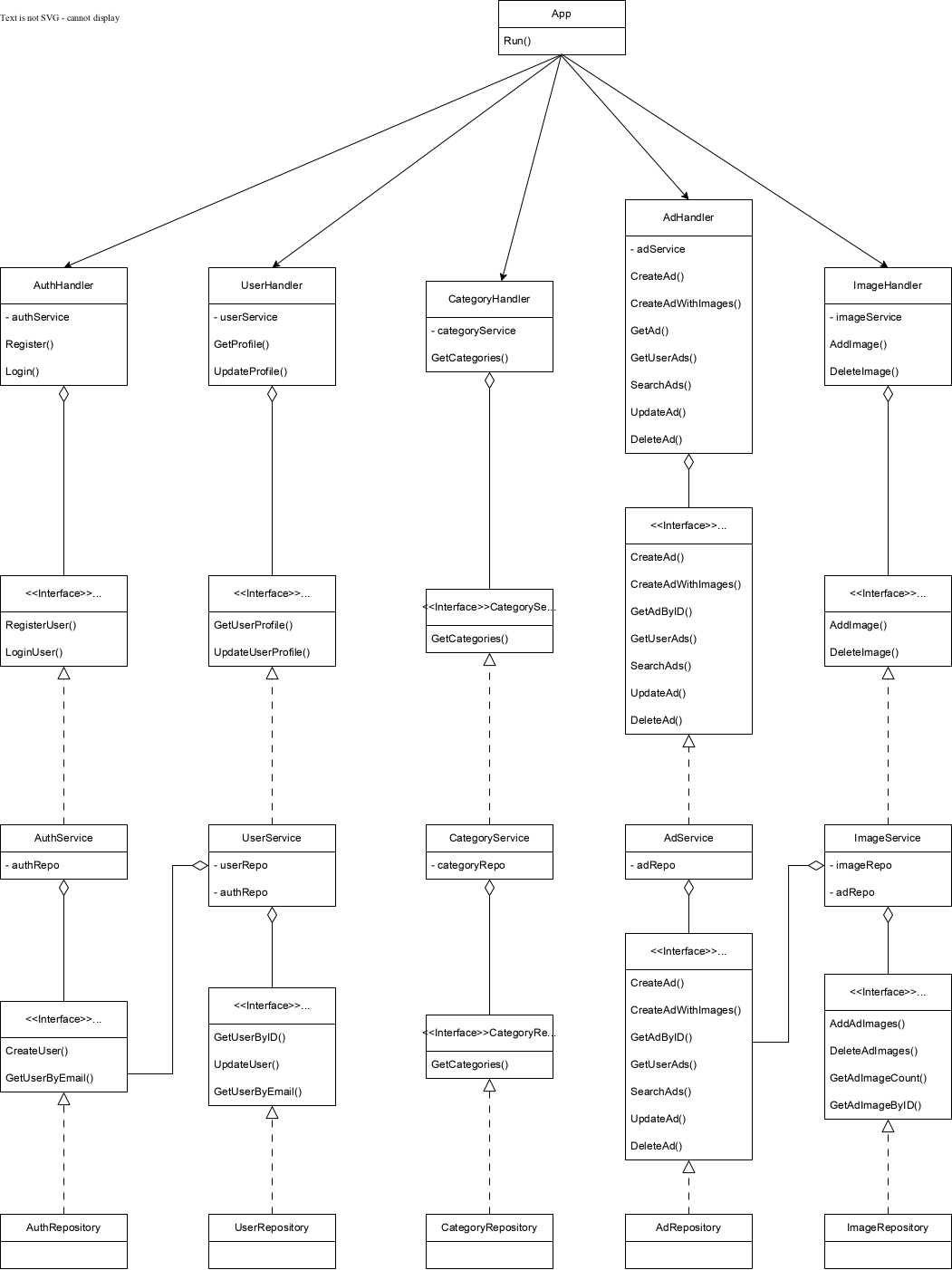


Рисунок 13 – Диаграмма классов

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.

# 3 Выбор стратегии тестирования и разработка тестов

# Заключение

В ходе выполнения курсовой работы проведен анализ предметной области управления медицинскими данными, спроектирована реляционная база данных, включая инфологическую и даталогическую модели. Разработаны сложные запросы с использованием многотабличных соединений и агрегатных функций, обеспечивающие аналитическую поддержку медицинского учреждения. Реализован скрипт генерации осмысленных тестовых данных на русском языке с учетом нормативных требований. Созданная система позволяет автоматизировать медицинскую систему и оптимизировать ресурсы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Скворцова М.А., Замула М.И. Методические рекомендации по написанию курсовой работы по курсу «Базы данных» [Электронный ресурс]. – URL: <https://e-learning.bmstu.ru/iu6/pluginfile.php/21072/mod_resource/content/2/МР_КР_БД_2%20курс_2025_фин1.pdf> (дата обращения 26.04.2025).

2. Приказ Минздрава РФ № 834н «Об утверждении унифицированных форм медицинской документации» [Электронный ресурс]. – URL: <https://normativ.kontur.ru/document?moduleId=1&documentId=471591> (дата обращения 26.04.2025).

3. Федеральный закон "О персональных данных" от 27.07.2006 N 152-ФЗ [Электронный ресурс]. – URL: <https://www.consultant.ru/document/cons_doc_LAW_61801/> (дата обращения 26.04.2025).

4. Фомин М.М. Реляционные базы данных. Учебное пособие для бакалавров: МГТУ имени Н.Э. Баумана, 2023. 161 с.

5. Документация PostgreSQL и Postgres Pro [Электронный ресурс]. – URL: <https://postgrespro.ru/docs> (дата обращения 26.04.2025).