МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка веб-приложения для обмена вещами между пользователями

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-284.ВКР

Научный руководитель,
профессор кафедры СП, д.ф.-м.н.,
доцент
_____ Т.А. Макаровских

Автор работы,
студент группы КЭ-403
_____ Д.А. Липин

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Липину Даниилу Александровичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка веб-приложения для обмена вещами между пользователями.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**
3.1. Tsoukalos M. Mastering Go: Harness the power of Go to build professional utilities and concurrent servers and services. – USA: Packt Publishing; 3rd Edition, 2021. – 682 p.
3.2. Документация Go. [Электронный ресурс] URL: https://go.dev/doc/ (дата обращения: 22.02.2024 г.).
3.3. Документация фреймворка React.js. [Электронный ресурс] URL: https://react.dev/learn/ (дата обращения: 22.02.2024 г.).
4. **Перечень подлежащих разработке вопросов**
4.1. Выполнить обзор научной литературы, провести анализ предметной области.
4.2. Произвести разбор существующих аналогов и подготовить необходимые требования к системе.
4.3. Выполнить проектирование веб-приложения.

4.4. Разработать веб-приложение для обмена вещами между пользователями.

4.5. Осуществить тестирование веб-приложения.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
профессор кафедры СП, д.ф.-м.н., доцент Т.А. Макаровских

Задание принял к исполнению Д.А. Липин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

С развитием цифровых технологий и расширением интернет-сообщества возникают новые потребности и возможности для обмена вещами и услугами. Исследования показывают, что модели обмена и совместного использования вещей могут способствовать более эффективному расходованию имеющихся ресурсов и уменьшению негативного воздействия на окружающую среду [1].

Традиционные модели потребления, основанные на владении и использовании личных ресурсов, часто приводят к избыточному потреблению и ненужному расходованию. Однако, исследования показывают, что принятие концепции обмена и совместного использования ресурсов может способствовать формированию более ответственного и устойчивого образа жизни.

Согласно исследованию «thredUp» [2], к 2027 году глобальный рынок поддержанных вещей достигнет 350 миллиардов долларов. В 2022 году 52 % потребителей в США приобретали подержанную одежду. Исследование показывает, что 83 % представителей поколения Z совершали или готовы совершать покупки в секонд-хенд магазинах. По подсчетам «thredUp» рынок онлайн-перепродаж достигнет 38 миллиардов долларов и будет расти в 2 раза быстрее, чем офлайн сектор.

Исследование «Data Insight» [3] показывает, что наиболее продаваемые товарные категории на C2C-рынке – «Электроника и бытовая техника» (24% сделок), «Одежда и обувь» (18%), «Детские товары» (17%), а также, что рынок онлайн C2C-торговли вырос на 87% за 1,5 года и превысил 1 триллион рублей, инфографика представлена на рисунке 1.

В этом контексте возникает потребность в разработке новых инструментов, которые облегчат процесс обмена и совместного использования ресурсов. Веб-приложения, способные обеспечить удобство, безопасность и

эффективность обмена вещами между людьми, играют ключевую роль в стимулировании развития и поддержки роста таких практик.

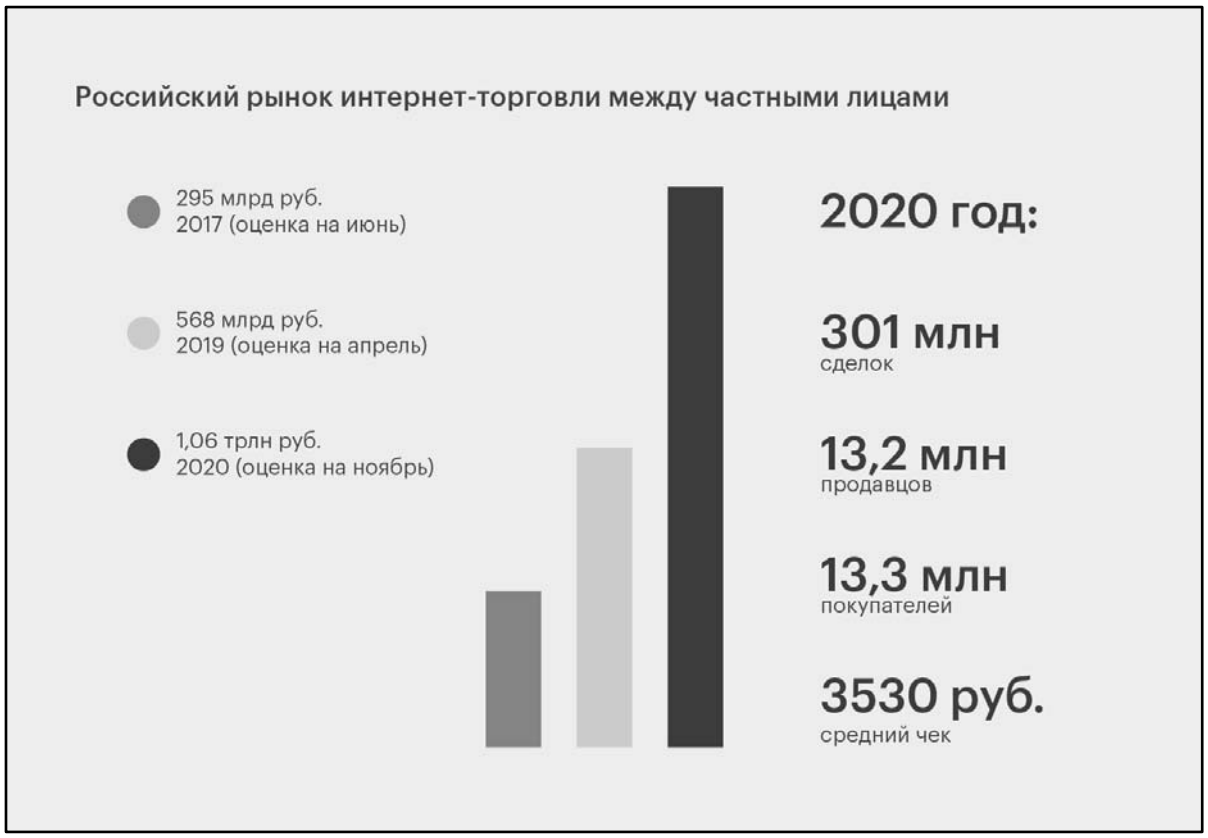

Рисунок 1 – Рынок онлайн C2C-торговли в России по годам

Данная работа описывает проектирование и разработку веб-приложения, которое предназначено для облегчения процесса обмена вещами между пользователями. Приложение объединяет в себе удобный интерфейс, механизмы безопасности и функциональные возможности, способствующие созданию динамичного сообщества, выступающего за разумное потребление. Приложение должно стать не просто техническим решением, а социальной платформой, направленной на улучшение качества жизни и сохранение окружающей среды за счет сокращения потребления и увеличения срока службы вещей.

Постановка задачи

Целью выпускной квалификационной работы является разработка веб-приложения для обмена вещами между пользователями. Для достижения поставленной цели необходимо решить следующие задачи:

1) осуществить анализ предметной области;
2) выбрать средства реализации проекта;
3) провести обзор аналогичных систем;
4) осуществить проектирование веб-приложения;
5) провести разработку приложения;
6) осуществить тестирование веб-приложения.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения, списка литературы и приложений. Объем работы составляет 67 страниц. Объем списка литературы – 15 источников.

В первой главе описывается предметная область разработки веб-приложений и выбираются инструменты разработки. Также были проанализированы сервисы для обмена вещами между пользователями.

Вторая глава посвящена проектированию веб-приложения. В ходе работы над данной главой были определены варианты использования, функциональные и нефункциональные требования, спроектированы база данных и архитектура веб-приложения.

В третьей главе описывается реализация веб-приложения.

В четвертой главе приводятся результаты функционального тестирования, а также тестирования верстки.

В приложении А содержатся скриншоты форм веб-приложения, доступных для заполнения пользователем.

В приложении Б содержатся скриншоты состояний панели навигации.

В приложении В содержаться скриншоты страниц веб-приложения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Технологии разработки веб-приложений

Веб-разработка является одной из самых динамичных и востребованных областей в современной IT-сфере. Это процесс создания приложений, доступных через интернет, которые могут быть использованы на различных устройствах и в разных браузерах.

Разработка веб-приложений требует учета различных аспектов, включая производительность, масштабируемость, безопасность и удобство использования. Выбор языка программирования для реализации серверной части приложения играет ключевую роль в достижении этих целей. Также важно обеспечить интерактивность и динамичность пользовательского интерфейса, что обуславливает выбор соответствующих фреймворков и инструментов для разработки клиентской части приложения.

Рассмотрим основные технологии, используемые в разработке веб-приложений, в контексте нашего веб-приложения для обмена вещами.

Go [4] – этот язык программирования, известный своей производительностью и эффективностью, будет использоваться для разработки серверной части веб-приложения. Он предоставляет простоту и надежность, что позволит создать быстрый и масштабируемый backend. Go разрабатывался как open-source проект и, с момента выхода в 2012 году, смог попасть на 8 место по популярности среди языков программирования. По индексу TIOBE за февраль 2024 года, Go занимает 8 позицию.

На Go написана серверная часть приложений: Twitch, Netflix, Uber, Авито и др.

Fiber [5] – это фреймворк для языка программирования Go, позволяющий писать REST API. Он предназначен для создания быстрых и эффективных серверных приложений. В своей основе, Fiber построен на базе библиотеки fasthttp, обеспечивающей одну из самых быстрых обработок HTTP-запросов.

GORM [6] – это ORM (Object-Relational Mapping) библиотека для Go. Она предоставляет удобный способ работы с базами данных, позволяя разработчикам взаимодействовать с базой данных через объекты Go, а не напрямую через SQL-запросы. Библиотека поддерживает автоматическое сопоставление таблиц и автоматические миграции схемы базы данных.

HTML (HyperText Markup Language) – это основной язык разметки для создания веб-страниц. Он используется для определения структуры и содержимого веб-страниц.

Tailwind CSS [7] – это CSS-фреймворк, который предоставляет классы для быстрого создания пользовательских дизайнов. В отличие от традиционных CSS-фреймворков, Tailwind не предоставляет предустановленных компонентов. Вместо этого он предлагает набор утилитарных классов, которые можно комбинировать для создания любого дизайна.

JavaScript [8] – является одним из основных языков программирования для веб-разработки. Он поддерживает объектно-ориентированное, функциональное и императивное программирование, что делает его гибким и мощным инструментом для разработки. В рамках данной работы, JavaScript будет использоваться для разработки клиентской части приложения, которая обеспечивает интерактивность и динамичность пользовательского интерфейса.

React.js [9] – это один из самых популярных JavaScript-библиотек. Она предназначена для создания пользовательских интерфейсов веб-приложений. Она позволяет строить мощные и эффективные веб-приложения, обеспечивая простой и понятный способ организации кода.

React.js основан на концепции компонентов. Каждый компонент представляет собой некоторую часть пользовательского интерфейса и может содержать в себе HTML, CSS и JavaScript код.

Также React.js использует виртуальный DOM для оптимизации производительности при обновлении пользовательского интерфейса. Вместо того, чтобы непосредственно изменять реальный DOM при каждом обновлении

данных, React создает виртуальное представление DOM, которое затем сравнивается с реальным. Это позволяет уменьшить количество манипуляций с DOM и повысить скорость отрисовки интерфейса.

PostgreSQL [10] – это мощная объектно-реляционная система управления базами данных (СУБД), которая предоставляет расширенные возможности для хранения и обработки данных. PostgreSQL является одной из наиболее популярных и широко используемых реляционных баз данных в мире.

PostgreSQL обеспечивает полную поддержку транзакций, что позволяет выполнить серию операций как единое целое, гарантируя целостность данных. Это делает данную СУБД идеальным выбором для приложений, где надежность и безопасность данных играют ключевую роль.

PostgreSQL обладает высокой производительностью и масштабируемостью, что позволяет эффективно обрабатывать как небольшие объемы данных, так и крупные базы данных с миллионами записей. Благодаря своей архитектуре и оптимизациям, PostgreSQL обеспечивает высокую скорость выполнения запросов и операций с данными.

Эти технологии и инструменты обеспечат надежную и масштабируемую основу для разработки веб-приложения. Они позволят создать удобный, быстрый и безопасный интерфейс для пользователей и обеспечат эффективную работу приложения на серверной стороне.

1.2. Сравнительный обзор аналогов

Для разработки веб-приложения, направленного на обмен вещами между пользователями, необходимо провести обзор уже существующих аналогов для того, чтобы выявить их главные преимущества и недостатки.

Данный подход позволит более точно определить необходимый функционал приложения, который обеспечит комфортную работу пользователя.

Karmitt [11]

Приложение для поиска и передачи вещей бесплатно. Пользователь размещает вещь, которую хочет отдать, а взамен бесплатно может найти в приложении то, что необходимо ему. За каждую отданную вещь пользователи получают баллы (кармитты), в разделе «партнеры» их можно обменять на подарки от различных сервисов. На рисунке 2 представлена главная страница сайта.

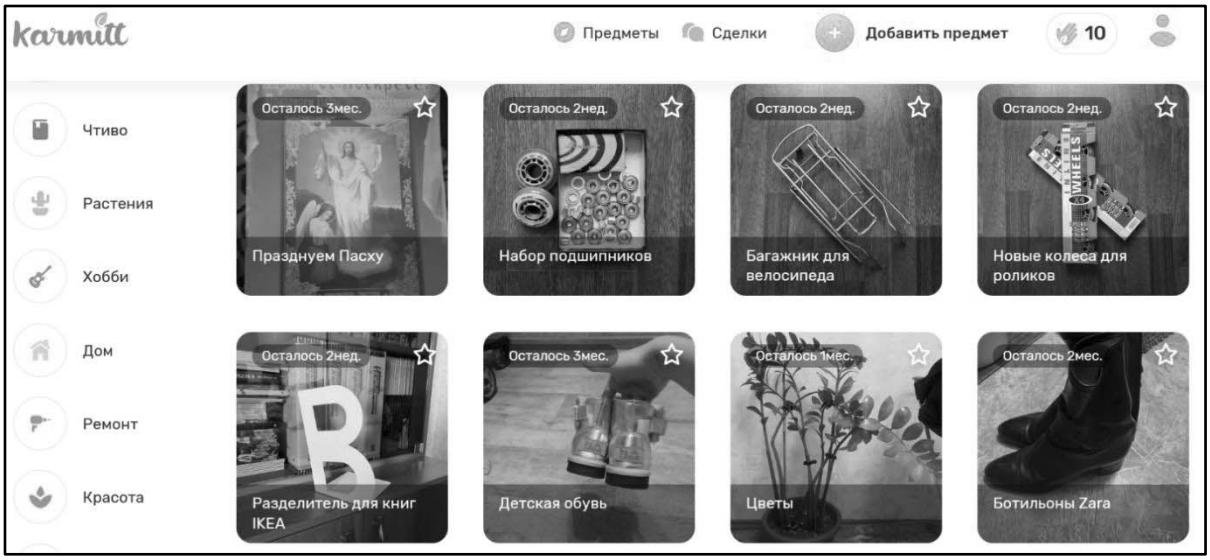

Рисунок 2 – Главная страница сервиса «Karmitt»

Ниже приведены достоинства и недостатки сервиса, выявленные в ходе использования и основанные на отзывах пользователей.

Достоинства:

1) простой и интуитивно понятный интерфейс;
2) простое размещение объявлений;
3) бонусы от партнеров.

Недостатки:

1) ограниченный выбор товаров в некоторых категориях;
2) не всегда эффективная система фильтрации объявлений;
3) возможные задержки в обработке запросов поддержки;
4) не реализована логика прямого обмена между пользователями.

Меняю 24 [12]

Данный сервис также предоставляет возможность обмена вещами между пользователями. На рисунке 3 представлена страница выбора категорий товаров.

Рисунок 3 – Выбор категорий товаров в сервисе «Меняю24»

К достоинствам данного сервиса можно отнести удобный выбор категорий, недостатки представлены ниже:

1) очень малое количество объявлений;
2) устаревший дизайн;
3) не реализована логика прямого обмена между пользователями.

SwopShop [13]

Эта онлайн-платформа позволяет пользователям размещать объявления с вещами для обмена и отдавать их бесплатно. По завершению сделки, пользователь получает внутреннюю валюту (СВОПы), которую он в дальнейшем может потратить на вещи других пользователей. Система «СВОПов» реализована для корректной оценки стоимости лотов и упрощения обмена между пользователями.

На рисунке 4 представлена страница с товарами из категории «Личные вещи» сервиса «SwopShop».

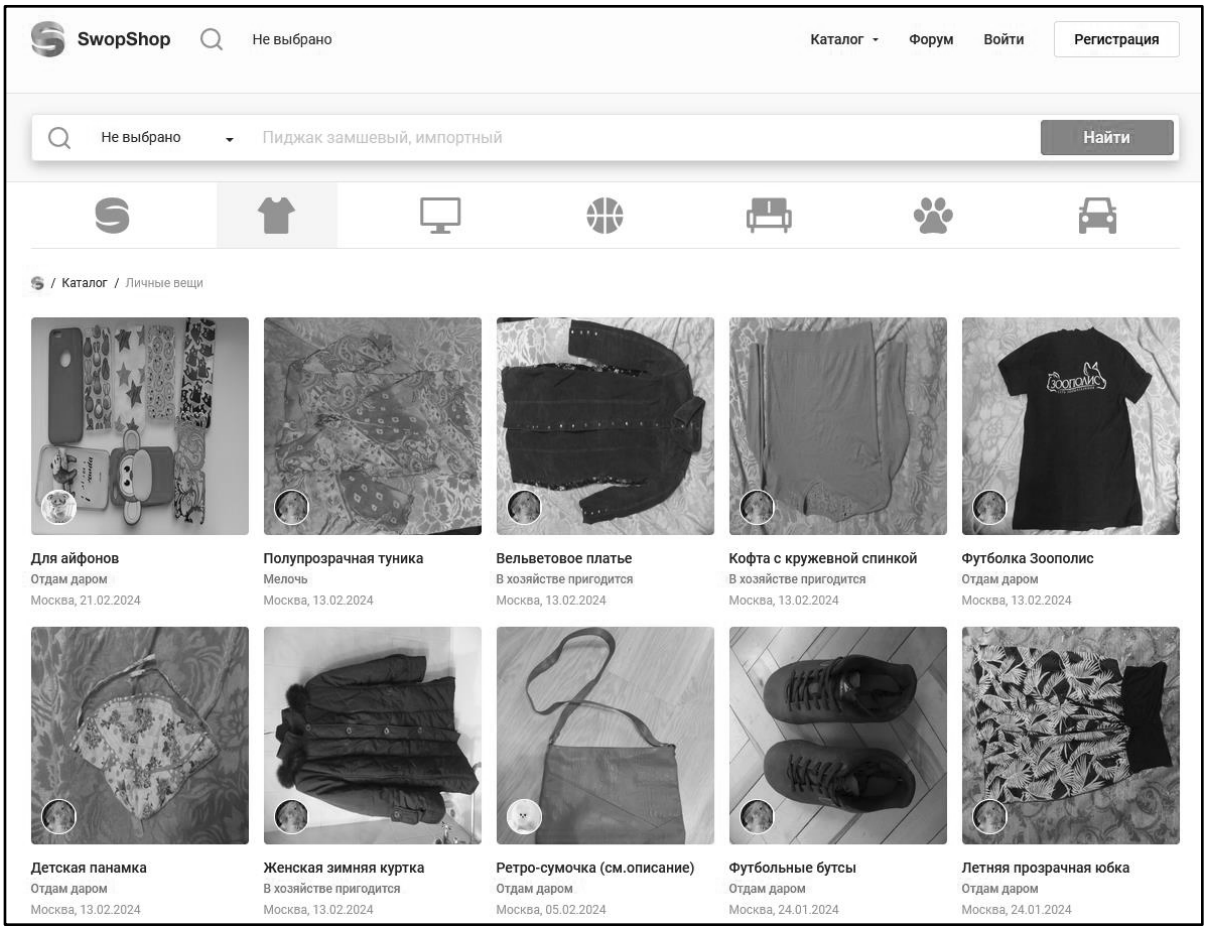

Рисунок 4 – Страница с товарами категории «Личные вещи» сервиса «SwopShop»

В ходе пользования данной онлайн-платформой были выявлены следующие достоинства и недостатки.

Достоинства:

1) широкий выбор товаров;
2) удобный, современный интерфейс;
3) наличие рейтинга у пользователей.

Недостатки:

1) невозможность получить вещь, при нехватке «СВОПов» на балансе;
2) некорректное отображение фото в объявлениях;
3) не реализована логика прямого обмена между пользователями.

Сравнительная таблица аналогов

В таблице 1 представлено сравнение аналогичных сервисов по пунктам. Сравнение аналогов в дальнейшем позволит более точно определить требования к разрабатываемой системе.

Таблица 1 – Сравнение аналогичных сервисов

Характеристики	Karmitt	Меняю24	SwopShop
Удобство интерфейса	+	+	+
Современный дизайн	+	–	+
Возможность просмотра объявлений для незарегистрированных пользователей	+	+	+
Распределение товаров по категориям	+	+	+
Возможность написать пользователю во внутреннем чате	+	+	+
Корректное отображение веб-страниц на мобильных устройствах	+	+	+
Корректное отображение фотографий в объявлениях	+	–	–
Реализация логики прямого обмена между пользователями	–	–	–

Вывод по первой главе

В рамках данной главы был проведен анализ предметной области проекта, и были подобраны инструменты для дальнейшей разработки. Также был произведен обзор аналогичных решений, это позволит определить требования, выдвигающиеся к будущей системе.

2. ПРОЕКТИРОВАНИЕ

2.1. Выявление функциональных и нефункциональных требований

Анализ предметной области и аналогичных сервисов позволил выделить требования, предъявляемые к разрабатываемой системе. Были выявлены следующие функциональные требования.

1. Незарегистрированные пользователи могут просматривать каталог объявлений, а также просматривать подробную информацию об объявлении.

2. Пользователи должны иметь возможность создать учетную запись и войти в систему с помощью email-адреса и пароля.

3. Пользователи могут размещать объявления о предметах, которые они готовы обменять, указывая название, описание, категорию, оценочную стоимость, состояние и фотографии товара.

4. Пользователи могут искать интересующие их товары по различным критериям, таким как категория или ключевые слова.

5. Пользователи могут общаться между собой через внутреннюю систему обмена сообщениями для обсуждения деталей сделок.

6. Пользователи имеют возможность управлять своими объявлениями, включая редактирование и удаление.

7. По завершению обмена, пользователи могут подтвердить в системе состояние обмена.

Нефункциональные требования, предъявляемые к системе.

1. Веб-приложение должно быть быстрым и отзывчивым, обеспечивая минимальное время загрузки страниц и выполнения операций.

2. Интерфейс приложения должен быть интуитивно понятным и удобным для пользователя, обеспечивая комфортный опыт использования.

3. Система должна иметь потенциал, для дальнейшего масштабирования, чтобы обеспечить поддержку роста числа пользователей и объема данных.

4. Система должна быть создана с использованием выбранных средств разработки.

5. Веб-приложение должно корректно отображаться в современных браузерах и иметь адаптивный интерфейс.

2.2. Диаграмма вариантов использования

В соответствии с выявленными требованиями к системе, была составлена диаграмма вариантов использования, представленная на рисунке 5. Диаграмма отображает модель взаимодействия актеров «Незарегистрированный пользователь», «Зарегистрированный пользователь» и «Администратор» с разрабатываемым веб-приложением.

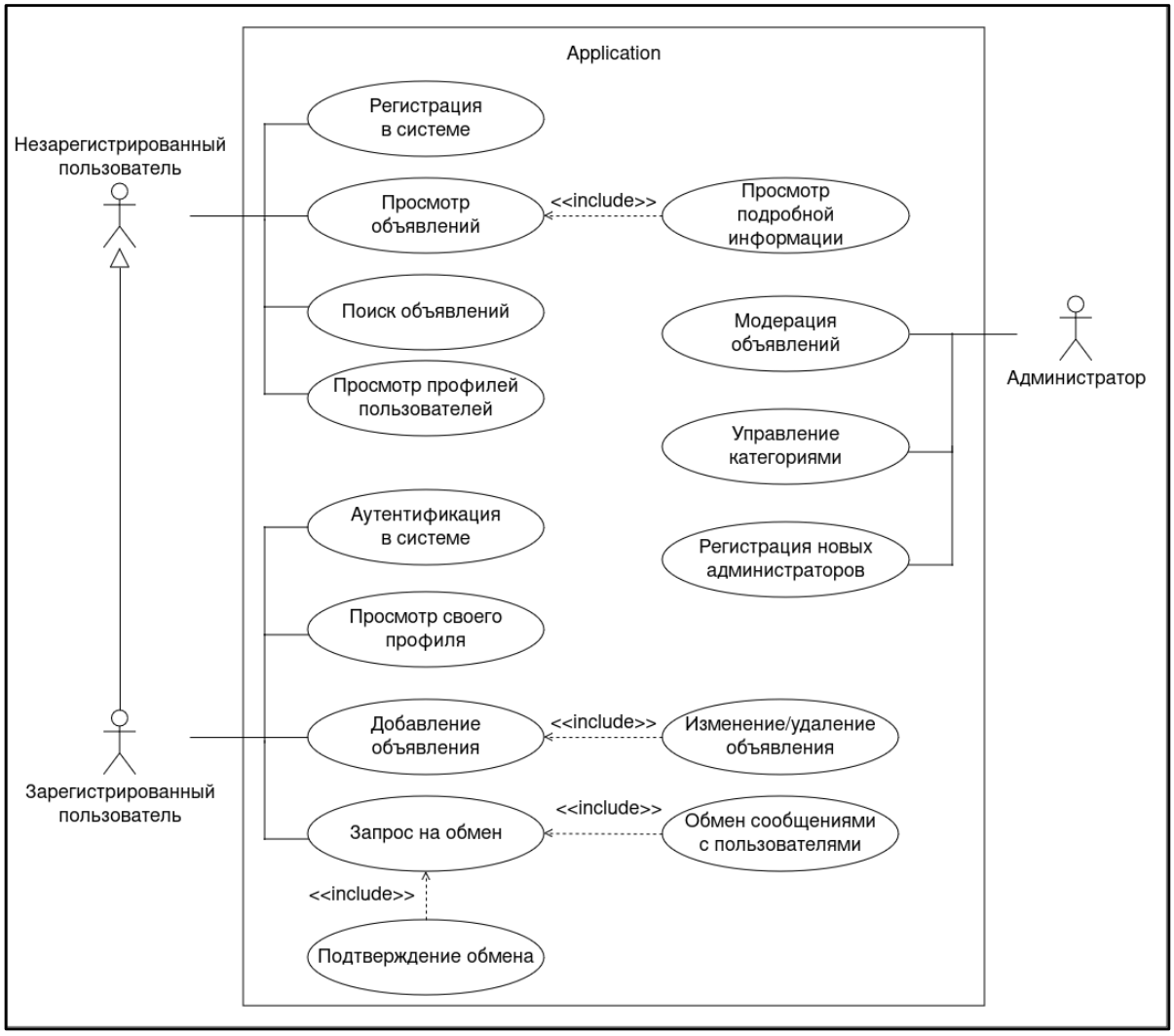

Рисунок 5 – Диаграмма вариантов использования

Актер «Незарегистрированный пользователь» в рамках разрабатываемой системы может:

- зарегистрироваться в системе;
- просматривать выложенные на площадку объявления;
- просматривать подробную информацию в объявлении;
- осуществлять поиск объявлений;
- просматривать профили других пользователей.

Актер «Зарегистрированный пользователь» наследует все действия «Незарегистрированного пользователя», а также обладает собственными действиями:

- аутентификация в системе;
- просмотр своего профиля;
- добавление объявления;
- изменение или удаление своего объявления;
- отправка запроса на обмен другим пользователям;
- обмен сообщениями с другими пользователями;
- подтверждение обмена.

Актер «Администратор» имеет доступ к таким вариантам использования системы, как:

- модерация объявлений;
- управление категориями;
- добавление новых администраторов в систему.

2.3. Диаграмма деятельности

Одной из главных функций, разрабатываемого веб-приложения, является возможность предложить обмен другому пользователю.

Для визуализации шагов выполнения прецедента «Запрос на обмен», была составлена диаграмма деятельности, представленная на рисунке 6.

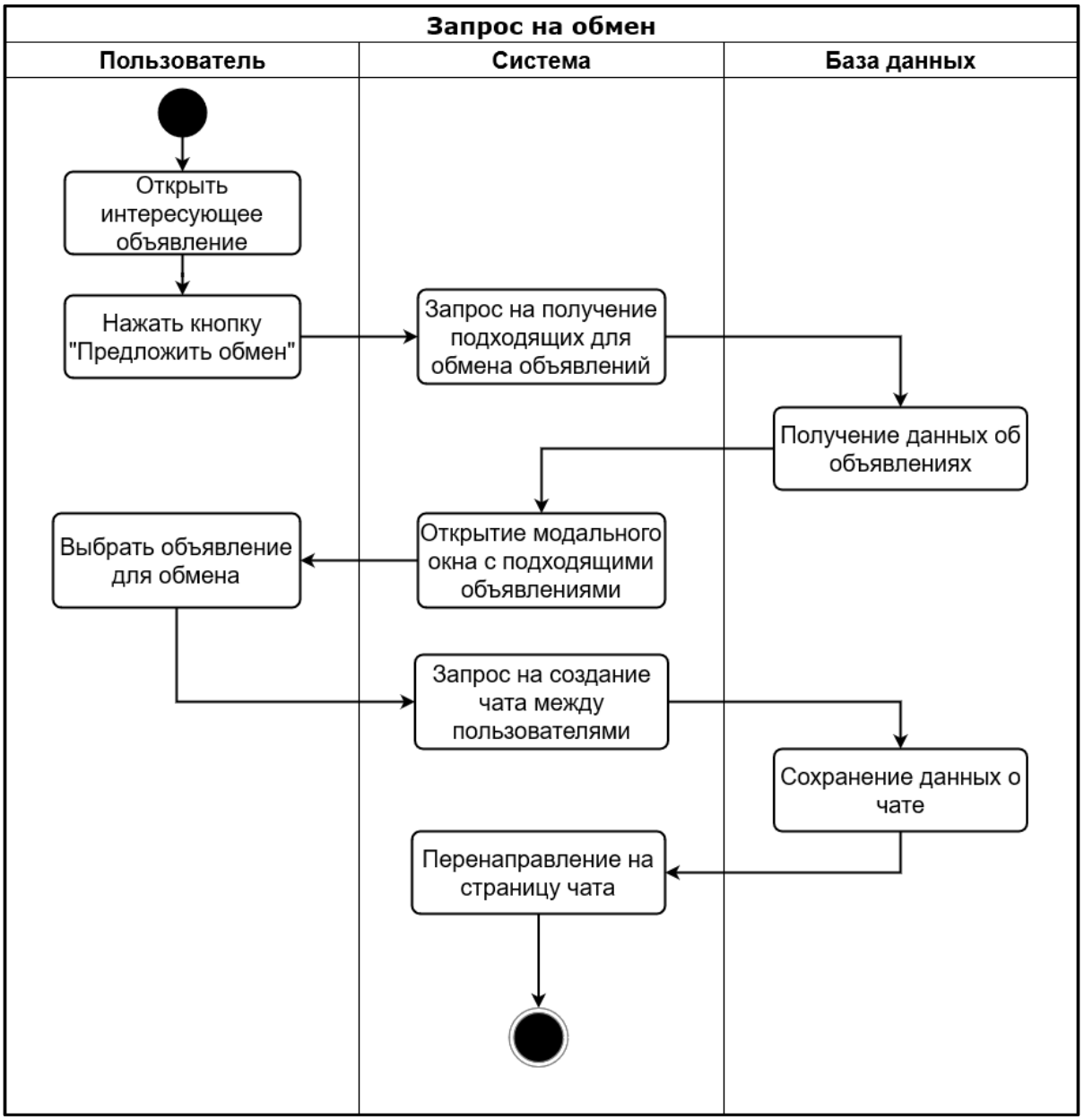

Рисунок 6 – Диаграмма деятельности для прецедента «Запрос на обмен»

2.4. Проектирование базы данных

Для проектирования базы данных разрабатываемого приложения, была выбрана модель семантического моделирования «сущность – связь», разработанная П. Ченом [14]. Для отображения связи между сущностями, была выбрана нотация Дж. Мартина («вороньи лапки») [15]. Спроектированная модель представлена на рисунке 7.

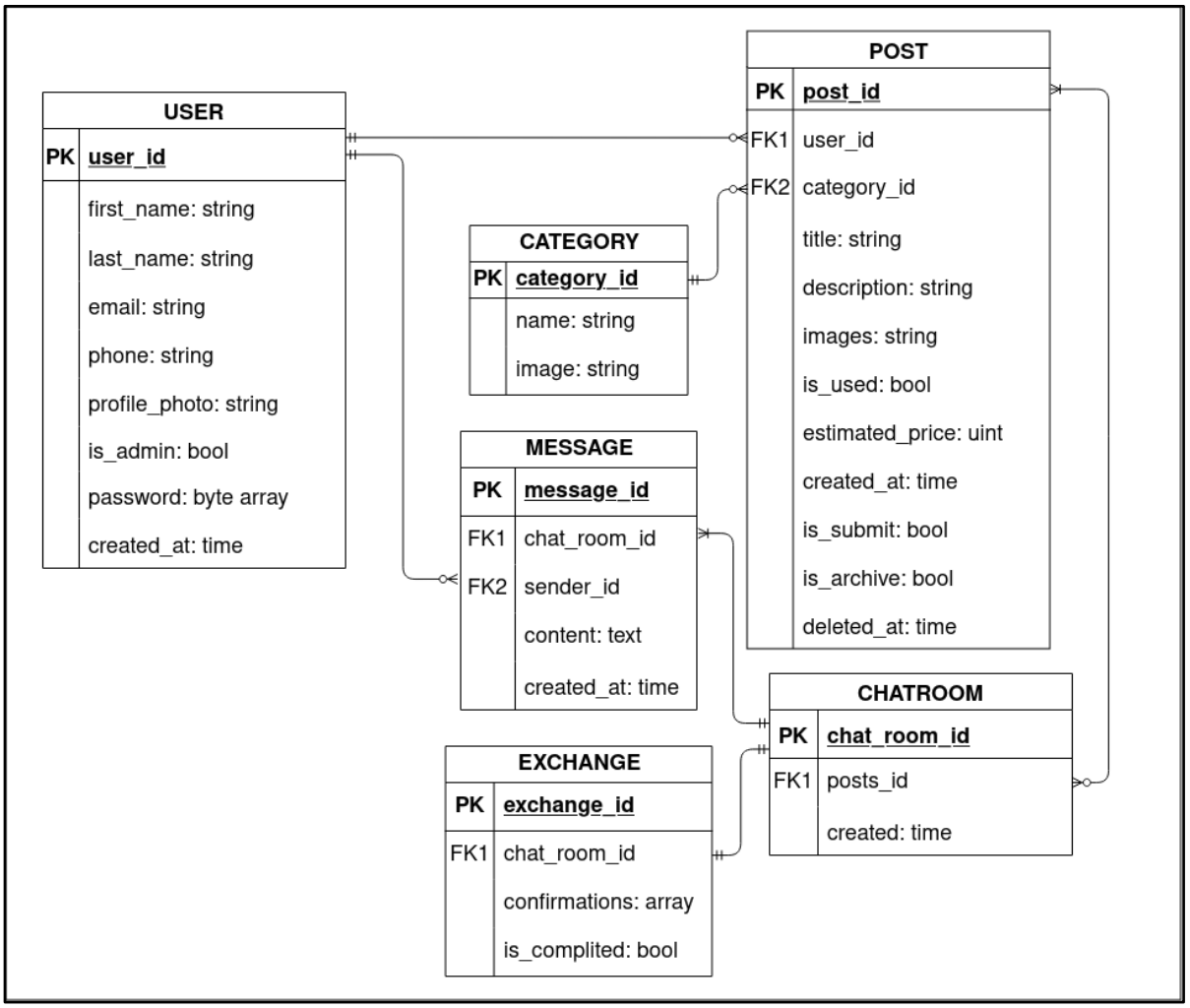

Рисунок 7 – Модель базы данных

Сущность «USER» содержит информацию о зарегистрированном пользователе и обладает такими атрибутами, как:

– user_id: уникальный идентификатор пользователя (PRIMARY KEY);

– first_name: текстовое поле с именем пользователя;

– last_name: текстовое поле с фамилией пользователя;

– email: текстовое поле с email адресом пользователя;

– phone: текстовое поле с телефонным номером пользователя;

– profile_photo: текстовое поле, содержащее URL–адрес фотографии профиля;

– is_admin: булево значение, хранящее информацию о том, является ли пользователь администратором;

– password: массив байтов, содержащий пароль в зашифрованном виде;

– created_at: данные типа time, хранящие информацию о моменте создания профиля.

Сущность «POST» определена для хранения данных объявления, она обладает следующими атрибутами:

– post_id: уникальный идентификатор объявления (PRIMARY KEY);

– user_id: идентификатор пользователя, разместившего объявление (FOREIGN KEY);

– category_id: идентификатор категории, к которой относится объявление (FOREIGN KEY);

– title: текстовое поле с заголовком объявления;

– description: текстовое поле, в котором содержится подробная информация о объявлении;

– photo: поле с URL-адресами фотографий;

– is_used: булево значение, для хранения данных о состоянии предмета;

– estimated_price: беззнаковое число, хранящее информацию об оценочной стоимости объявления;

– is_submit: булево значение, для определения, подтверждено ли объявление администратором;

– is_archive: булево значение, для обозначения, находится ли объявление в архиве;

– created_at: данные типа time, хранящие информацию о моменте создания объявления;

– deleted_at: поле с типом time, будет необходима для реализации функции «мягкого удаления» объявления.

Сущность «CATEGORY» обладает уникальным идентификатором категории (PRIMARY KEY), текстовым полем с названием категории и адресом иконки.

Сущность «CHATROOM» необходима для хранения данных о созданных чатах между пользователями. Она обладает уникальным идентификатором чата (PRIMARY KEY), идентификаторами объявлений, к которым относится чат (FOREIGN KEY) и временем создания чата.

Сущность «MESSAGE» определена для хранения данных сообщения в чате. Она обладает такими атрибутами, как:

– message_id: уникальный идентификатор сообщения (PRIMARY KEY);

– chat_room_id: идентификатор чата, к которому относится сообщение (FOREIGN KEY);

– sender_id: идентификатор отправителя сообщения (FOREIGN KEY);

– content: поле с текстом сообщения;

– time: время отправки сообщения.

Сущность «EXCHANGE» необходима для хранения данных о состоянии обмена между пользователями. Она обладает уникальным идентификатором обмена (PRIMARY KEY), идентификатором чата, к которому относится обмен (FOREIGN KEY), полем, хранящим данные о пользователях, подтвердивших обмен и булево значение, для хранения данных о состоянии обмена.

Связи между сущностями представлены ниже.

1. Связь «один-ко-многим» между сущностями «USER» и «POST». Один пользователь может разместить много объявлений, но каждое объявление принадлежит только одному пользователю.

2. Связь «многие-ко-многим» между сущностями «POST» и «CHATROOM». К одному объявлению может относится много чатов, и чат также содержит информацию о двух объявлениях, участвующих в обмене.

3. Связь «один-ко-многим» между сущностями «CHATROOM» и «MESSAGE». Одному чату может принадлежать много сообщений, но сообщение всегда принадлежит только одному чату.

4. Связь «один-к-одному» между сущностями «CHATROOM» и «EXCHANGE». Одному чату может принадлежать только один обмен.

5. Связи «один-ко-многим» между сущностями «USER» и «MESSAGE». Отправителем сообщения может быть только один пользователь, но пользователь может отправлять множество сообщений.

6. Связь «один-ко-многим» между сущностями «CATEGORY» и «POST». Одной категории может принадлежать много объявлений, но объявление всегда принадлежит одной категории.

2.5. Проектирование архитектуры приложения

Существует четыре класса архитектурных моделей «клиент-сервер»: одноуровневые, двухуровневые, трехуровневые и многоуровневые. В архитектуре разрабатываемого веб-приложения будет использоваться трехуровневая модель. В этой модели присутствуют три слоя: слой представления (клиент, который взаимодействует с пользователем), слой бизнес-логики (сервер) и слой данных (база данных).

Клиент – это компонент, обычно представленный в графическом виде, который предоставляется конечному пользователю. Этот уровень не должен иметь прямых связей с базой данных, основная бизнес-логика также вынесена за пределы этого компонента и клиент не хранит данные о состоянии приложения. Это сделано для поддержания важнейших характеристик качества при эксплуатации, также называемого «external quality»: масштабируемости, надежности и информационной безопасности. В уровне клиента содержится минимальная логика, необходимая для взаимодействия пользователя с приложением. Этот уровень представлен веб-интерфейсом, который пользователь видит и использует в своем веб-браузере. Веб-браузер отправ-

ляет запросы к серверу, чтобы получить информацию или выполнить определенные действия, а затем отображает полученные результаты пользователю.

Серверная часть приложения реализована в виде RESTful сервиса. REST является аббревиатурой от Representational State Transfer и в первую очередь представляет собой архитектуру для проектирования веб-сервисов. REST не привязан ни к какой операционной системе или системной архитектуре и не является протоколом. Тем не менее, чтобы реализовать RESTful-сервис необходимо использовать протокол HTTP. Обычно, RESTful-сервисы используют формат JSON для обмена информацией. Сервер будет отвечать за выполнение бизнес-логики приложения, и предоставлять API для клиента, что позволяет ему взаимодействовать с приложением, отправлять запросы и получать соответствующие ответы.

Одним из основных преимуществ REST является его гибкость и масштабируемость, что позволяет легко добавлять новые функции и расширять существующие. Также, благодаря использованию стандартных HTTP методов (GET, POST, PUT, DELETE), RESTful сервисы обеспечивают простоту в интеграции с различными клиентами, будь то веб-браузеры, мобильные приложения или другие серверы.

В трехуровневой архитектуре база данных представляет собой отдельный слой, где хранится и управляется информация. Этот слой обеспечивает доступ к данным для сервера приложения, который занимается обработкой запросов и логикой приложения. В разрабатываемом веб-приложении, для работы с сервером баз данных будет использоваться система управления базами данных PostgreSQL. Взаимодействие с сервером баз данных осуществляется только с сервера приложения, что обеспечивает централизацию доступа к данным и повышает безопасность.

Архитектура разрабатываемого приложения представлена на рисунке 8.

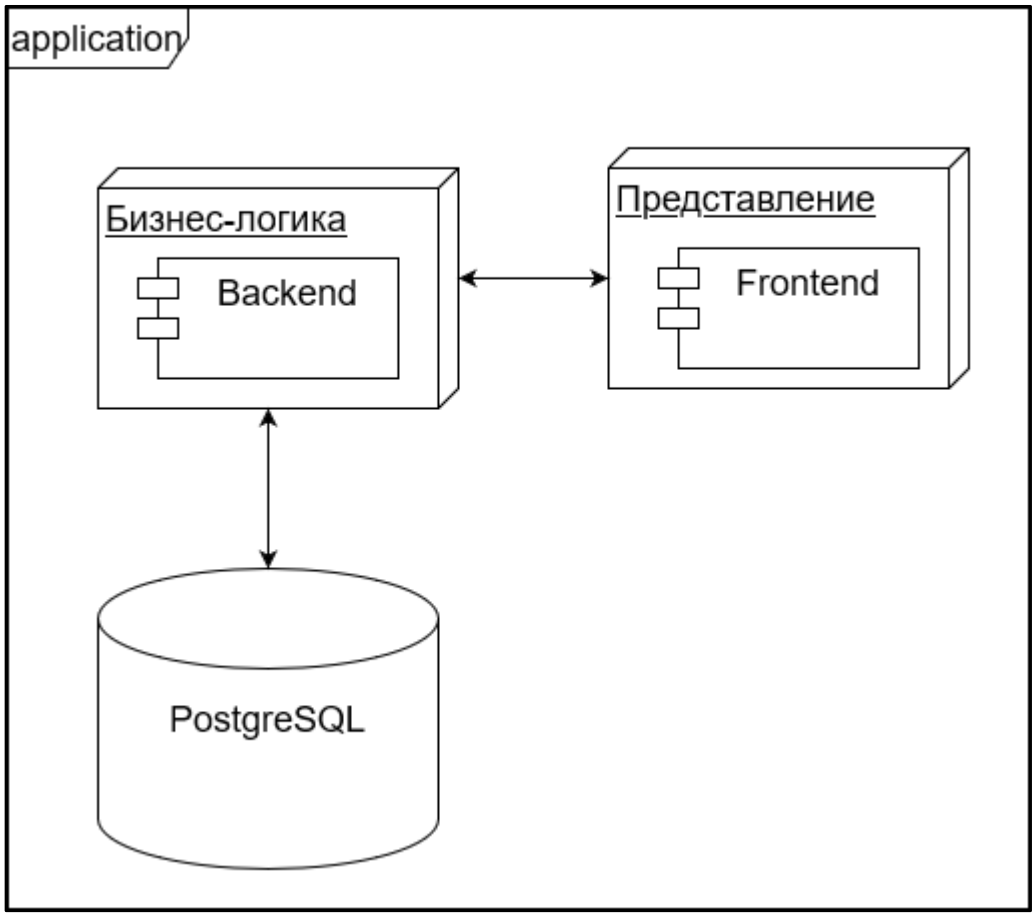

Рисунок 8 – Архитектура веб-приложения

Вывод по второй главе

В рамках данной главы было выполнено проектирование веб-приложения. Были составлены требования к системе, диаграммы вариантов использования и деятельности. Также была разработана схема базы данных и выбрана архитектурная модель веб-приложения.

3. РЕАЛИЗАЦИЯ

3.1. Подключение базы данных и настройка маршрутов обработчиков

Создание проекта

Все программы на языке Go поставляются в виде пакетов, поэтому для создания исполняемого файла необходимо явно указать его принадлежность к пакету `main`. Также, для установления зависимостей для импортируемых пакетов необходимо прописать в консоль команду «`go mod init`». Она создаст файл модуля `go.mod` в котором будут хранится все зависимости проекта с загружаемыми пакетами. Для включения библиотек и фреймворков в проект необходимо прописывать в консоль команду «`go get`».

Для создания клиентского React-приложения, необходимо прописать в консоли команду «`create-react-app`». Она создает проект со всеми базовыми модулями. Для установки внешних библиотек используется утилита командной строки «`npm install`».

Подключение базы данных

Для подключения сервера к серверу базы данных PostgreSQL используется библиотека GORM. Код для подключения к базе данных представлен на листинге 1.

Листинг 1 – Подключение базы данных

```
func Connect() {
	if err := initConfig(); err != nil {
		log.Fatal(«Ошибка инциализации конфигурации»)
	}
	err := godotenv.Load()
	if err != nil {
		log.Fatal(«Ошибка загрузки файла окружения»)
	}
	host := viper.GetString(“db.host”)
	port := viper.GetString(“db.port”)
	username := viper.GetString(“db.username”)
	dbname := viper.GetString(“db.dbname”)
	sslmode := viper.GetString(“db.sslmode”)
	password := os.Getenv(“DB_PASSWORD”)
	fmt.Print(host)
	dsn := fmt.Sprintf(“host=%s user=%s password=%s dbname=%s port=%s
sslmode=%s”, host, username, password, dbname, port, sslmode)
	database, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
	if err != nil {
		panic(«Невозможно подключиться к базе данных»)
	} else {
		log.Println(«Подключение к БД успешно»)
	}
```

```
	DB = database
	database.AutoMigrate(
		&models.User{},
		&models.Post{},
		&models.Category{},
		&models.ChatRoom{},
		&models.Message{},
		&models.Exchange{},
	)
}
```

Переменные для подключения к базе данных, такие как `host`, `port` и другие, хранятся в файле конфигурации `config.yaml` и передаются в функцию с помощью библиотеки `viper`. Пароль базы данных хранится в файле окружения `.env`, который, в свою очередь указан в файле `.gitignore`, для игнорирования загрузки на GitHub.

В функции подключения используется автоматическая миграция моделей из пакета `models`, это позволяет создать сущности и связи в базе данных без SQL-запросов.

Описание моделей

В качестве примера описания моделей, в листинге 2 приводится код модели `Post`, которая описывает сущность объявления в базе данных.

Листинг 2 – Модель `Post`

```
type Post struct {
	Id             uint         `json:”id” gorm:”primaryKey”`
	Title          string       `json:”title” gorm:”not null”`
	Description    string       `json:”description” gorm:”not null”`
	Images         string       `json:”images” gorm:”type:text”`
	IsUsed         bool         `json:”is_used”`
	EstimatedPrice uint         `json:”estimated_price”`
	CreatedAt      *time.Time   `json:”created_at” gorm:”not null;de-
fault:now()”`
	IsSubmit       bool         `json:”is_submit”`
	IsArchive      bool         `json:”is_archive”`
	UserId         string       `json:”user_id” gorm:”not null”`
	User           User         `json:”user” gorm:”foreignKey:UserId”`
	CategoryId     string       `json:”category_id” gorm:”not null”`
	Category       Category     `json:”category” gorm:”foreignKey:Catego-
ryId”`
	Chats          []*ChatRoom `gorm:”many2many:chats_posts;”`
	DeletedAt      *time.Time   `json:”deleted_at” gorm:”index”`
}
```

У каждого поля указано название, тип данных, который оно представляет, тег `json`, который указывает как поле должно именоваться при сериализации и десериализации данных, а также специальный тег `gorm`.

Первичный ключ имеет тег `primaryKey`, внешний же имеет тег `foreignKey`. В данной модели также определена связь «многие-ко-многим», она обозначена тегом `many2many` с последующим названием связующей таблицы, которая также создается автоматически при миграции базы данных.

Маршруты обработчиков

Для установления маршрутов, по которым клиент будет получать доступ к обработчикам, создадим пакет `routes` и определим в нем функцию `Setup`, в которую передается созданный Fiber-сервер, и которая будет вызываться при запуске сервера. В листинге 3 приводятся несколько примеров маршрутов, определенных в функции `Setup`.

Листинг 3 – Маршрутизация контроллеров

```
app.Use(cors.New(cors.Config{
      AllowOrigins: "http://localhost:3000/",
      AllowMethods: "GET,POST,HEAD,PUT,DELETE,PATCH,OPTIONS",
      AllowCredentials: true,
}))
app.Get("/api/posts", controllers.AllPost)
app.Post("/api/registration", controllers.Registration)
app.Static("/api/uploads", "./uploads")
app.Use(middleware.IsAuthenticate)
app.Put("/api/soft-delete/:id", controllers.SoftDeletePost)
app.Delete("/api/delete-post/:id", controllers.DeletePost)
app.Use(middleware.IsAdmin)
app.Get("api/admin/non-submit-posts", controllers.GetNonSubmitPosts)
```

Метод `Get`, фреймворка Fiber, определен для GET HTTP-запроса, в него передается маршрут, по которому клиент будет обращаться к обработчику и указывается сам обработчик из пакета `controllers`. Методы `Post`, `Put` и `Delete` определены сходным образом.

Метод `Static` будет использоваться клиентом, для получения доступа к файлам загруженным на сервер в директорию «`./uploads`»

Метод `Use` используется для вызова промежуточного программного обеспечения и контроля параметров доступа. В данном приложении исполь-

зуется cors.Config для указания браузеру того, что клиент может взаимодействовать с сервером и данными в базе данных. Также в системе используется промежуточное программное обеспечение IsAuthenticate и IsAdmin из пакета middleware. Оно необходимо для проверки прав доступа пользователя, авторизованного в системе. Маршруты, определенные после IsAuthenticate, доступны только авторизованным пользователям, а маршруты, описанные после IsAdmin, доступны только администратору. В листинге 4 описана функция IsAdmin из пакета middleware.

Листинг 4 – Функция IsAdmin

```
func IsAdmin(c *fiber.Ctx) error {
	authHeader := c.Get("Authorization")
	if authHeader == "" {
		return c.Status(fiber.StatusUnauthorized).JSON(fiber.Map{
			"message": "Неавторизован",
		})
	}
	token := strings.TrimPrefix(authHeader, "Bearer ")
	if token == authHeader {
		return c.Status(fiber.StatusUnauthorized).JSON(fiber.Map{
			"message": "Неавторизован",
		})
	}
	userId, isAdmin, err := util.ParseJWT(token)
	if err != nil {
		return c.Status(fiber.StatusUnauthorized).JSON(fiber.Map{
			"message": "Ошибка аутентификации",
		})
	}
	c.Locals("user_id", userId)
	c.Locals("is_admin", isAdmin)
	if isAdmin {
		return c.Next()
	}
	return c.Status(fiber.StatusUnauthorized).JSON(fiber.Map{
		«message»: «Доступ отклонен»,
	})
}
```

Данная функция получает JWT-токен из поля Authorization, декодирует его с помощью функции ParseJWT из пакета util и определяет права доступа в зависимости от содержания полей токена.

3.2. Панель навигации

Панель навигации будет отображаться на всех страницах веб-приложения, за исключением страницы регистрации и аутентификации. Она

должна отображаться в трех состояниях, в зависимости от того, авторизован ли пользователь и является ли он администратором. Для того, чтобы при авторизации или выходе пользователя из системы, панель навигации отображалась в соответствующем состоянии, без необходимости явного обновления страницы пользователем, был определен контекст пользователя, представленный в листинге 5.

Листинг 5 – Компонент `UserContext`

```
export const UserProvider = ({ children }) => {
  const [user, setUser] = useState(() => {
    const storedUser = localStorage.getItem('user');
    return storedUser ? JSON.parse(storedUser) : null;
  });
  const logout = () => {
    setUser(null);
    localStorage.removeItem('user');
  };
  const updateUser = (userData) => {
    setUser(userData);
    localStorage.setItem('user', JSON.stringify(userData));
  };
  useEffect(() => {
    if (user) {
      localStorage.setItem('user', JSON.stringify(user));
    } else {
      localStorage.removeItem('user');
    }
  }, [user]);
  return (
    <UserContext.Provider value={{ user, updateUser, logout }}>
      {children}
    </UserContext.Provider>
  );
};
```

В данном компоненте используется хук `useState` для управления состоянием пользователя. Начальное значение состояния устанавливается с помощью функции, которая загружает данные пользователя из `localStorage`. Если данные существуют, они парсятся из строки JSON, иначе устанавливается `null`.

Также в компоненте объявляется функция `logout` с использованием стрелочной функции. Она очищает состояние пользователя и удаляет данные из `localStorage`. Хук `useEffect` следит за изменениями состояния

user и обновляет localStorage при каждом изменении. В конце компонента возвращается провайдер который дает доступ к состоянию и функциям, определенным в компоненте.

В листинге 6 описывается фрагмент кода компонента NavigationBar.

Листинг 6 – Компонент NavigationBar

```
const location = useLocation();
const navigate = useNavigate();
const { user, logout } = useUser();

const [categories, setCategories] = useState([]);
const [isDropdownOpen, setIsDropdownOpen] = useState(false);

useEffect(() => {
  const fetchCategories = async () => {
    try {
      const response = await axios.get('/api/categories');
      setCategories(response.data.data);
    } catch (error) {
      console.error('Error fetching categories:', error);
    }
  };

  fetchCategories();
}, []);

if (location.pathname === '/login' || location.pathname === '/registra-
tion') {
  return null;
}

const handleSearch = (event) => {
  event.preventDefault();
  const query = event.target.elements.search.value;
  if (query.trim()) {
    navigate(`/search?query=${query}`);
  }
};
```

В данном фрагменте кода клиент получает состояние пользователя из контекста с помощью хука useUser. Также загружаются категории при монтировании компонента, с помощью обращения к обработчику на сервере, и определяется функция для обработки формы поиска. Также происходит проверка, если текущий URL-адрес содержит «/login» или «/registration», панель навигации не отображается.

В дальнейшем, при разметке компонента используется тернарный оператор, который отображает элементы в зависимости от состояния пользователя.

Отображение панели навигационной панели для неавторизованного пользователя представлено на рисунке 1 в приложении Б.

Для неавторизованного пользователя доступен вывод объявлений по категориям. Выпадающее меню выбора категорий представлено на рисунке 9.

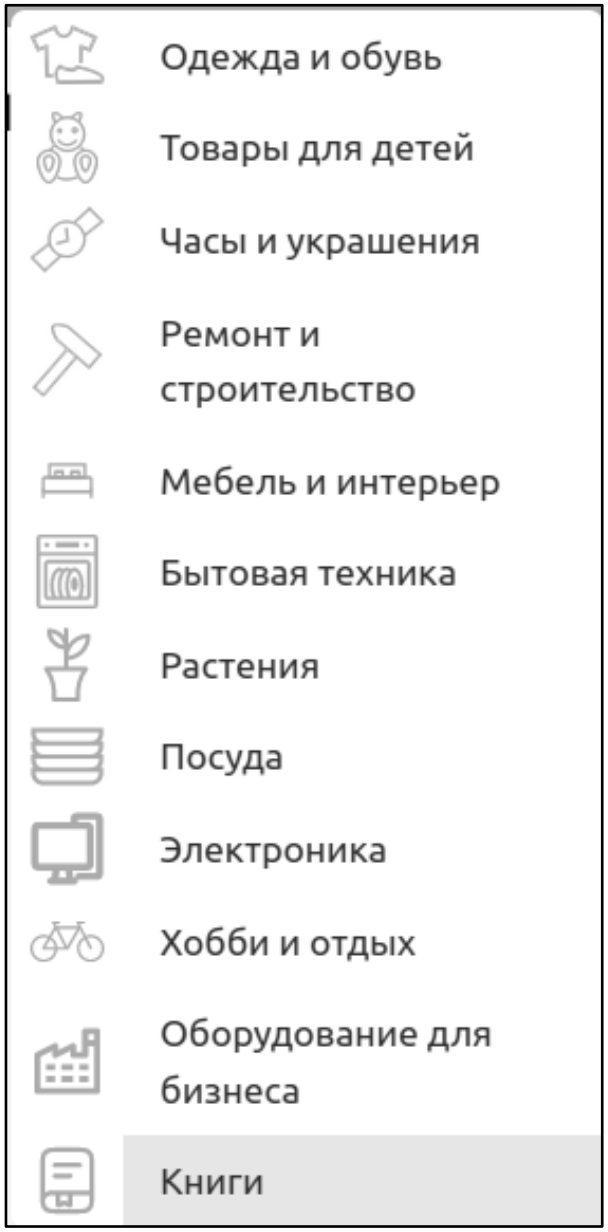

Рисунок 9 – Выпадающее меню, со списком категорий

Также неавторизованный пользователь может осуществлять поиск, по ключевым словам, для этого ему необходимо ввести ключевое слово в

форму поиска и нажать кнопку «Поиск», затем пользователь будет перенаправлен на страницу поиска. Часть кода компонента `SearchResults` представлена в листинге 7.

Листинг 7 – Хук `useEffect` компонента `SearchResults`

```
useEffect(() => {
  const queryParams = new URLSearchParams(location.search);
  const query = queryParams.get('query');
  const fetchResults = async () => {
    try {
      const response = await axios.get(`/api/search-
posts?query=${query}`);
      setResults(response.data.data);
      setLoading(false);
    } catch (error) {
      setError('Ошибка при выполнении поиска');
      setLoading(false);
    }
  };
  fetchResults();
}, [location.search]);
```

Клиент получает значение поискового запроса из текущего URL-адреса, а затем передает этот запрос обработчику `SearchPosts` на сервере. Для неавторизованного пользователя отображается кнопка «Войти», ведущая на страницу авторизации

Отображение панели навигации для авторизованного пользователя представлено на рисунке 2 в приложении Б.

Для авторизованного пользователя доступно то же, что и для неавторизованного, за исключением того, что в углу экрана, вместо кнопки «Войти», отображается его фото профиля, имя и фамилия, ведущие на его страницу пользователя.

Отображение панели навигации для администратора представлено на рисунке 3 в приложении Б.

Администратор имеет возможность проводить модерацию объявлений, выложенных пользователями, управлять категориями, добавлять новых администраторов в систему, а также выйти из профиля.

3.3. Главная страница и отображение объявлений по категориям

Главная страница

Главная страница веб-приложения описана в компоненте `Home`. На главной странице отображаются все объявления, добавленные пользователями, прошедшие модерацию и не находящиеся в архиве. В листинге 8 представлен хук `useEffect` для компонента `Home`.

Листинг 8 – Использование `useEffect` в компоненте `Home`

```
useEffect(() => {
  const fetchPosts = async () => {
    try {
      const response = await axios.get('/api/posts');
      setPosts(response.data.data);
      setLoading(false);
    } catch (error) {
      console.error('Error fetching posts:', error);
      setLoading(false);
    }
  };
  fetchPosts();
}, []);
```

В этом хуке объявлена асинхронная функция `fetchPosts`, которая отправляет запрос серверу, на получение данных о всех объявлениях и записывает ответ в `posts`.

Так как в базе данных URL-адреса всех фотографий объявления хранятся в виде строки нам нужно корректно их получить в виде массива для этого используется фрагмент кода из листинга 9.

Листинг 9 – Парсинг строки в массив URL-адресов

```
let firstImage = '/api/uploads/icons/general/no_photo.svg'; // Default im-
age
try {
  if (post.images) {
    const cleanedString = post.images.slice(1, -1).replace(/\\"/g, '"');
    const withoutBrackets = cleanedString.replace(/^\[|\]$/g, '');
    const images = JSON.parse(`[${withoutBrackets}]`);
    if (Array.isArray(images) && images.length > 0) {
      firstImage = images[0];
    }
  }
} catch (error) {
  console.error('Error parsing images:', error);
}
```

Главная страница веб-приложения представлена на рисунке 1 в приложении В.

Пользователь может переместится на страницу объявления, нажав на него, также он может перейти на страницу пользователя, добавившего объявление, или перейти на страницу категории, нажав на название категории. Также у каждого объявления отображается время его создания.

Адаптивная верстка страницы достигается классами Tailwind CSS. Отображение главной страницы на экране телефона представлено на рисунке 10.

Рисунок 10 – Отображение главной страницы на экране телефона

Отображение объявлений по категориям

Страница с объявлениями определенной категории описана в компоненте `CategoryPosts`. В нем описана асинхронная функция `fetchCategoryAndPosts`, представленная в листинге 10.

Листинг 10 – Функция `fetchCategoryAndPosts`

```
const fetchCategoryAndPosts = async () => {
  try {
    const postsResponse = await axios.get(`/api/category/${id}`);
    setPosts(postsResponse.data);

    const categoriesResponse = await axios.get('/api/categories');
    const category = categoriesResponse.data.data.find(category => cate-
gory.id === parseInt(id));

    if (category) {
      setCategoryName(category.name);
    } else {
      setError('Категория не найдена');
    }

    setLoading(false);
  } catch (error) {
    setError('Ошибка при загрузке данных');
    setLoading(false);
  }
};
```

В рамках данной функции отправляются два GET-запроса на сервер, для определения, существует ли категория с указанным id в URL-адресе, и для получения всех объявлений определенной категории.

Отображение объявлений на странице происходит сходным образом с главной страницей приложения. Скриншот страницы с объявлениями категории «Книги» представлен на рисунке 2 в приложении В.

3.4. Регистрация и авторизация

Страница регистрации пользователя описана в компоненте `Registration`. В нем объявлена асинхронная функция `handleRegister`, она проверяет введенные пользователем данные на корректность, и в случае успеха, посылает POST-запрос на сервер, обработчику `Registration`, описанному в листинге 11.

Листинг 11 – Обработчик `Registration`

```
func Registration(c *fiber.Ctx) error {
	var data map[string]interface{}
	var userData models.User
	if err := c.BodyParser(&data); err != nil {
		fmt.Println(«Невозможно получить тело запроса»)
	}
	if len(data[“password”].(string)) <= 6 {
		c.Status(400)
		return c.JSON(fiber.Map{
			«message»: «Пароль должен быть больше 6 символов»,
		})
	}
	if !validateEmail(strings.TrimSpace(data[“email”].(string))) {
		c.Status(400)
		return c.JSON(fiber.Map{
			“message”: “Неправильный формат Email”,
		})
	}
	database.DB.Where(“email=?”,
strings.TrimSpace(data[“email”].(string))).First(&userData)
	if userData.Id != 0 {
		c.Status(400)
		return c.JSON(fiber.Map{
			«message»: «Аккаунт с этим Email уже зарегистрирован»,
		})
	}
	user := models.User{
		FirstName: data[“first_name”].(string),
		LastName:  data[“last_name”].(string),
		Phone:     data[“phone”].(string),
		Email:     strings.TrimSpace(data[“email”].(string)),
		IsAdmin:   false,
	}
	avatarPath := fmt.Sprintf(“./uploads/profile-pictures/%s.png”,
strings.TrimSpace(data[“email”].(string)))
	if err := util.CreateAvatar(user.FirstName, user.LastName, avatar-
Path); err != nil {
		log.Println(«Ошибка генерации аватара:», err)
	}
	user.ProfilePhoto = fmt.Sprintf(“http://localhost:8080/api/up-
loads/profile-pictures/%s.png”, strings.TrimSpace(data[“email”].(string)))
	user.SetPassword(data[“password”].(string))
	err := database.DB.Create(&user)
	if err != nil {
		log.Println(err)
	}
	log.Println(«Регистрация успешна»)
	c.Status(200)
	return c.JSON(fiber.Map{
		“user”:    user,
		“message”: “Аккаунт успешно создан!”,
	})}
```

Данный обработчик проверяет полученный пароль по длине, сверяет email-адрес по формату, проверяет наличие email-адресов базе данных. Также данный обработчик вызывает функцию для генерации аватара пользователя, состоящего из первых букв его имени и фамилии на цветном фоне.

В конце выполнения обработчик вызывает функцию для хэширования пароля. Форма регистрации пользователя представлена на рисунке 1 в приложении А.

Авторизация

Страница авторизации пользователя описана в компоненте `Login`. В ней определена функция `handleSubmit` которая отправляет POST-запрос на сервер. На стороне бэкенда обработчик проверяет полученные данные на корректность, находит пользователя в базе данных, генерирует JWT-токен и отправляет данные о пользователе и токен клиенту. Клиент, получив ответ, сохраняет данные в `localStorage`, определяет является ли пользователь администратором и в зависимости от этого перенаправляет его в личный профиль или на панель администратора.

Также на страницах `Registration` и `Login`, с помощью `Snackbar` настроены оповещения об ошибках или успешном выполнении запроса. Форма авторизации пользователя представлена на рисунке 2 в приложении А.

3.5. Страница объявления

При нажатии на название объявления, происходит переадресация на страницу объявления, описанную в компоненте `PostDetail`. В компоненте объявлена функция `fetchPost`, код которой представлен в листинге 12.

Листинг 12 – Функция `fetchPost` компонента `PostDetail`

```
const fetchPost = async () => {
  try {
    const response = await axios.get(`/api/post/${id}`);
    setPost(response.data.data);

    const storedUser = JSON.parse(localStorage.getItem('user'));
    if (storedUser && storedUser.user && String(storedUser.user.id) ===
response.data.data.user_id) {
      setIsOwner(true);
    }
    if (storedUser && storedUser.token) {
      setIsAuthenticated(true);
    }

    setLoading(false);
  } catch (error) {
    console.error('Ошибка получения объявлений:', error);
```

```
    setLoading(false);
  }
};
```

Эта функция отправляет GET-запрос на сервер для получения данных об объявлении, также она проверяет, авторизован ли пользователь, и является ли он владельцем объявления.

В разметке страницы используется условный оператор для определения находится ли объявление в архиве. Скриншот архивированного объявления представлен на рисунке 3 в приложении В.

Также в разметке страницы используется еще один условный оператор для корректного отображения страницы, в зависимости от того, является ли пользователь владельцем объявления.

Отображение страницы для пользователя, не являющегося владельцем объявления, представлено на рисунке 4 в приложении В.

К кнопке «Предложить обмен» относится функция `handleOfferExchange`, которая проверяет авторизован ли пользователь. Если нет, перенаправляет его на страницу авторизации. Если же пользователь авторизован, то она вызывает функцию `fetchSimilarPosts`, которая отправляет GET-запрос на сервер, для получения подходящих для обмена объявлений.

По нажатию кнопки авторизованным пользователем, открывается модальное окно с подходящими для обмена объявлениями. Логика определения подходящих объявлений реализована таким образом, что пользователь может обменяться только на ту вещь, оценочная стоимость которой, больше или меньше оценочной стоимости его вещи, не более чем на 20%.

При нажатии на объявление в модальном окне, вызывается функция `handleCreateChat`, которая обращается к серверу с POST-запросом, на создание чата, относящегося к двум объявлениям, затем пользователь перенаправляется на страницу чата. Скриншот модального окна представлен на рисунке 11.

Рисунок 11 – Модальное окно для выбора объявления

Если же авторизованный пользователь является владельцем объявления, для него отображаются две кнопки «Редактировать объявление» и «Удалить объявление». Скриншот объявления, принадлежащего авторизованному пользователю представлен на рисунке 5 в приложении В.

Если объявление уже участвует в каком-либо обмене, то кнопка удаления, отвечает за «мягкое удаление» объявления, то есть оно перестает отображаться на всех страницах приложения, но остается в базе данных, если же объявлением никто не успел заинтересоваться, то по нажатию кнопки оно удаляется из базы данных. Кнопка «Редактировать объявление» перенаправляет пользователя на страницу редактирования объявления

3.6. Добавление и редактирование объявления

Добавление объявления

Для загрузки объявления в систему, на клиенте определен компонент `CreatePost`. Форма для загрузки объявления представлена на рисунке 3 в приложении А.

На странице используется функция `fetchCategories` для получения категорий и дальнейшего размещения их в выпадающем списке. У пользователя есть возможность ввести название объявления, загрузить фотографии, написать описание, выбрать категорию, указать оценочную стоимость вещи и выбрать состояние вещи.

В компоненте также установлены ограничения на длину названия (50 символов), длину описания (500 символов), количество, размер и формат фотографий (максимум 10 фотографий, максимальный размер фото – 5 МБ, доступны к загрузке файлы формата png и jpeg). Также реализована функция предпросмотра фотографий и удаления их из формы до отправки.

После нажатия кнопки «Выложить объявление», вызывается функция `onSubmit`, ее код приводится в листинге 13.

Листинг 13 – Функция `onSubmit` компонента `CreatePost`

```
const onSubmit = async (data) => {
  const user = localStorage.getItem('user');
  const parsedUser = JSON.parse(user);
  if (!parsedUser) {
    setError('Пользователь не авторизован');
    setOpen(true);
    return;
  }
  const token = parsedUser.token;
  const userId = parsedUser.user.id.toString();
  const images = data.images ? Array.from(data.images) : []
  if (images.length > 10) {
    setError('Максимальное количество фото: 10');
    setOpen(true);
    return;
  }
  try {
    const imageUrls = [];
    for (let I = 0; I < images.length; i++) {
      const formData = new FormData();
      formData.append('image', images[i]);
      const response = await axios.post('/api/upload-image', formData, {
        headers: {
          'Authorization': `Bearer ${token}`,
          'Content-Type': 'multipart/form-data',
        },
        withCredentials: true,
      });
      imageUrls.push(response.data.url);
    }
    const post = {
      title: data.title,
      description: data.description,
      category_id: data.categoryId,
      images: JSON.stringify(imageUrls),
      is_used: isUsed,
```

```
      user_id: userId,
      estimated_price: Number(data.estimatedPrice),
    };
    await axios.post('/api/post', post, {
      headers: {
        'Authorization': `Bearer ${token}`,
      },
      withCredentials: true,
    });
    setSuccess('Ваше объявление было создано!');
    setError(null);
    setOpen(true);
    setTimeout(() => {
      navigate('/');
    }, 1500);
  } catch (error) {
    console.error('Ошибка при создании объявления', error);
    setError('Ошибка при создании объявления');
    setSuccess(null);
    setOpen(true);
  }
};
```

В данной функции система получает данные пользователя из `localStorage`. Затем, поочередно загружает фото, добавленные пользователем с помощью обработчика на сервере. Далее передает данные, введенные пользователем, а также токен, для промежуточного программного обеспечения, на сервер.

Редактирование объявления

Компонент `UpdatePost`, для редактирования объявления, реализован сходным с `CreatePost` образом, за исключением того, что мы используем хук `useEffect` с функцией `fetchPost` для того, чтобы получить данные об объявлении на текущий момент. Также функция `onSubmit` в данном компоненте отправляет не POST, а PUT-запрос на сервер для изменения текущих данных. Форма редактирования объявления с предзагруженными данными представлена на рисунке 4 в приложении А.

3.7. Профиль пользователя

Для отображения профиля пользователя определен компонент `UserDetail`. В нем используется хук `useEffect` в котором объявлены две функции `fetchUserData` и `fetchUserPosts`. Первая используется для полу-

чения данных о пользователе и проверки, является ли авторизованный пользователь владельцем аккаунта, а вторая используется для получения объявлений пользователя. Отображение страницы профиля пользователя представлено на рисунке 6 в приложении В.

Пользователь может просматривать информацию другого пользователя, а также его активные и архивные объявления. Отображение страницы профиля пользователя для владельца аккаунта представлено на рисунке 7 в приложении В.

Для владельца аккаунта отображаются три кнопки: «Выложить объявление», «Ваши чаты» и «Выйти из аккаунта». Первая перенаправляет пользователя на страницу создания объявления, вторая на страницу с его чатами, а третья вызывает функцию `handleLogout`, которая удаляет данные пользователя из сессии и перенаправляет его на главную страницу. Также авторизованный пользователь может видеть свои объявления, которые находятся на этапе модерации, они обозначены надписью «На проверке».

3.8. Реализация чатов

Для реализации работы с пользовательскими чатами, на стороне клиента определены два компонента `UserChats` и `Chat`, первый отвечает за отображение всех чатов, в которых участвует пользователь, а второй за страницу конкретного чата. В компоненте `UserChats` используется хук `useEffect` с функцией `fetchChats`, которая отправляет запрос на сервер, для получения всех чатов пользователя. Скриншот страницы с чатами пользователя представлен на рисунке 8 в приложении В.

По нажатию на интересующий чат, пользователь перенаправляется на страницу этого чата.

В компоненте `ChatPage` определен хук `useEffect` с функциями `fetchChat`, `fetchNewMessages` и `fetchExchangeStatus` для получения данных чата, сообщений и обмена. Скриншот страницы чата представлен на рисунке 9 в приложении В.

Пользователю отображается информация об объявлениях, участвующих в обмене, сообщения, разнесенные по разным сторонам экрана, в зависимости от отправителя, поле для ввода сообщения, кнопка для его отправки и кнопка для подтверждения обмена.

При нажатии на кнопку «Подтвердить обмен», статус обмена меняется, и пользователь видит сообщение «Обмен подтвержден, ожидается подтверждение от другого пользователя», кнопка становится неактивной, а другой пользователь видит сообщение «Другой пользователь подтвердил обмен, ожидается подтверждение от вас». После того как оба пользователя подтвердили обмен, объявления переходят в статус архивированных и больше не могут участвовать в обменах.

3.9. Панель администратора

Для создания защищенных маршрутов был создан файл `ProtectedRoutes`, его код приведен в листинге 14.

Листинг 14 – Файл `ProtectedRoutes`

```
import React from 'react';
import { Navigate, Outlet } from 'react-router-dom';
import { useUser } from './UserContext';
const AdminRoute = () => {
  const { user } = useUser();
  return user && user.user.is_admin ? <Outlet /> : <Navigate to="/" />;
};
const UserRoute = () => {
  const { user } = useUser();
  return user && !user.user.is_admin ? <Outlet /> : <Navigate to="/" />;
};
export { AdminRoute, UserRoute };
```

Система использует контекст пользователя для определения является ли он администратором или обычным пользователем. Структура маршрутов для администратора представлена в листинге 15.

Листинг 15 – Структура маршрутов для администратора

```
<Route path="/admin/*" element={<AdminRoute />}>
  <Route path="" element={<AdminPanel />} />
  <Route path="categories" element={<ManageCategories />} />
  <Route path="create-admin" element={<CreateAdmin />} />
</Route>
```

Модерация объявлений

В компоненте `Moderation` реализован хук `useEffect`, содержащий функцию `fetchNonSubmitPosts` для получения с сервера неподтвержденных объявлений. Также определены функции `handleSubmitPost` и `handleDeletePost` для подтверждения или отклонения объявления. Скриншот страницы модерации представлен на рисунке 10 в приложении В.

Управление категориями

В компоненте `ManageCategories` описан хук `useEffect`, включающий функцию `fetchCategories` для получения категорий с сервера, а также реализованы функции `handleCreateOrUpdateCategory` и `handleDeleteCategory`, для создания, обновления или удаления категории. Скриншот страницы управления категориями представлен на рисунке 11 в приложении В.

Регистрация администратора

В компоненте `CreateAdmin` описывается страница регистрации администратора. Форма администратора схожа с тем, что описывалось в компоненте `Registration`, за исключением того, что при создании администратора вызывается другой обработчик, который автоматически выставляет значение поля `is_admin` равным `true`. Также, при регистрации администратора, нет необходимости ставить согласие с условиями обработки персональных данных. Форма регистрации администратора представлена на рисунке 5 в приложении А.

Вывод по третьей главе

В рамках данной главы были описаны ключевые этапы разработки системы. Подключение и создание базы данных, создание компонентов веб-приложения.

4. ТЕСТИРОВАНИЕ

Функциональное тестирование

Функциональное тестирование приложения проводится с целью проверки работоспособности функций приложения, в соответствии с установленными требованиями. Функциональные тесты приложения представлены в таблице 2.

Таблица 2 – Функциональные тесты приложения

№	Название теста	Действие	Ожидаемый результат	Тест пройден?
1	Показ всех объявлений	Переход на главную страницу	Корректное отображение всех объявлений, среди которых нет находящихся в архиве или непроверенных администратором	Да
2	Показ объявлений по категориям	Переход на страницу категории	Корректное отображение всех объявлений, выбранной категории	Да
3	Поиск объявления	В строке поиска вводится ключевое слово, нажатие кнопки «Поиск»	Отображаются все объявления, содержащие это ключевое слова	Да
4	Переход на страницу профиля	Нажатие на ссылку, ведущую к профилю пользователя	Корректное отображение информации о пользователе, верный показ его активных и архивных объявлений	Да
5	Переход на страницу объявления	Нажатие на объявление	Корректное отображение информации об объявлении, показ кнопок, в зависимости от его статуса	Да
6	Успешная регистрация	Попытка регистрации пользователя с верными данными	Пользователь получает сообщение об успехе, запись о нем добавляется в базу данных, происходит перенаправление на страницу авторизации	Да
7	Регистрация с неверными данными	Попытка регистрации с неверными данными или с уже существующим email адресом	Пользователь получает сообщение об ошибке с пояснением	Да

Продолжение таблицы 2

№	Название теста	Действие	Ожидаемый результат	Тест пройден?
8	Успешная авторизация пользователя	Попытка войти в систему с верными данными	Пользователь получает сообщение об успехе, происходит перенаправление на страницу профиля	Да
9	Успешная авторизация администратора	Попытка войти в систему с верными данными администратора	Администратор получает сообщение об успехе, происходит перенаправление на панель администратора	Да
10	Авторизация с неверными данными	Попытка войти в систему, указав неверные данные	Пользователь получает сообщение об ошибке с пояснением	Да
11	Переход в личный профиль	Нажатие на ссылку, ведущую в личный профиль	Отображение личных данных, кнопок, активных, находящихся на модерации и архивных объявлений	Да
12	Добавление объявления	Заполнение формы, нажатие кнопки «Выложить объявление»	Пользователь получает сообщение об успехе, данные сохранены в базу данных	Да
13	Переход в личное объявление	Нажатие на объявление, принадлежащее пользователю	Корректное отображение информации об объявлении, показ кнопок в зависимости от статуса объявления	Да
14	Редактирование объявления	Заполнение формы, нажатие кнопки «Редактировать объявление»	Заполнение формы, нажатие кнопки «Редактировать объявление»	Да
15	Удаление объявления	Заполнение формы, нажатие кнопки «Редактировать объявление»	Пользователь получает сообщение об успехе, данные обновлены в базе данных	Да
16	«Мягкое удаление» объявления	Находясь на странице личного объявления с ненулевым статусом обмена, нажать кнопку «Удалить объявление»	Пользователь получает сообщение об успехе, поле deleted_at обновлено	Да
17	Запрос на обмен	Находясь на странице объявления, нажать кнопку «Предложить обмен»	Отображение объявлений доступных для обмена	Да

Окончание таблицы 2

№	Название теста	Действие	Ожидаемый результат	Тест пройден?
18	Выбор объявления для обмена	Нажатие на объявление в модальном окне, с подходящими для обмена объявлениями	Создание чата между пользователями, создание обмена со статусом «В процессе», перенаправление на страницу чата	Да
19	Отправка сообщений	Пользователь вводит текст сообщения в поле для ввода, затем нажимает кнопку «Отправить»	Сообщение отображается в чате и сохранено в базе данных	Да
20	Подтверждение обмена	Нажатие кнопки «Подтвердить обмен» на странице чата	Пользователь получает информацию о статусе обмена, кнопка недоступна для нажатия	Да
21	Модерация объявлений	Администратор переходит на страницу «Модерация объявлений»	Корректное отображение всех неподтвержденных объявлений	Да
22	Подтверждение объявления	Администратор нажимает кнопку для подтверждения объявления	Объявление подтверждено, поле is_submit обновлено в базе данных	Да
23	Управление категориями	Изменение, удаление или добавление категории администратором	Данные в базе данных обновлены	Да
24	Добавление нового администратора	Администратор заполняет форму, нажимает кнопку	Сообщение об успехе, новый администратор добавлен в базу данных	Да

Тестирование верстки веб-приложения

При тестировании, веб-приложение было открыто с различными размерами окна в таких браузерах как: Firefox (126.0.1), Яндекс Браузер (24.1.1.940), Google Chrome (25.0.6422.141), Opera One (110.0.5130.66). Во всех браузерах интерфейс оставался идентичным.

Вывод по четвертой главе

В рамках данной главы было проведено функциональное тестирование веб-приложения, а также была протестирована верстка. Все тесты были успешно пройдены.

ЗАКЛЮЧЕНИЕ

Целью данной выпускной квалификационной работы была разработка веб-приложения для обмена вещами между пользователями.

В ходе выполнения данной работы были решены следующие задачи:

1) осуществлен анализ предметной области;
2) выбраны средства для реализации проекта;
3) проведен обзор аналогичных систем;
4) осуществлено проектирование веб-приложения;
5) проведена разработка веб-приложения;
6) осуществлено тестирование веб-приложения.

В рамках данной выпускной квалификационной работы было разработано веб-приложение для обмена вещами между пользователями, также были выполнены все поставленные задачи.

В дальнейшем планируется расширять функционал приложения, добавить возможность указывать геолокацию для объявления, ставить оценки пользователям. Также возможно развертывание приложения на удаленном сервере, улучшение его надежности и возможности масштабирования.

ЛИТЕРАТУРА

1. Botsman R., Rogers R. What’s Mine Is Yours: The Rise of Collaborative Consumption. // Harper Business, 2010. – 304 p.

2. Resale Report 2023 | thredUp. [Электронный ресурс] URL: https://cf-assets-tup.thredup.com/resale_report/2023/thredUP_2023_Resale_Report_FINAL.pdf (дата обращения: 10.02.2024 г.).

3. Маркетинговое исследование. Рынок C2C-онлайн торговли 2020 | Data Insight. [Электронный ресурс] URL: https://datainsight.ru/DI_Avito_C2C_2020 (дата обращения: 10.02.2024 г.).

4. Документация языка программирования Go. [Электронный ресурс] URL: https://go.dev/doc/ (дата обращения: 11.02.2024 г.).

5. Документация фреймворка Fiber. [Электронный ресурс] URL: https://docs.gofiber.io/ (дата обращения: 12.02.2024 г.).

6. Документация библиотеки GORM. [Электронный ресурс] URL: https://gorm.io/docs/ (дата обращения: 12.02.2024 г.).

7. Документация фреймворка Tailwind CSS. [Электронный ресурс] URL: https://v2.tailwindcss.com/docs (дата обращения: 13.02.2024 г.).

8. Документация языка программирования JavaScript. [Электронный ресурс] URL: https://devdocs.io/javascript/ (дата обращения: 14.02.2024 г.).

9. Документация фреймворка React.js. [Электронный ресурс] URL: https://react.dev/learn/ (дата обращения: 14.02.2024 г.).

10. Документация СУБД PostgreSQL. [Электронный ресурс] URL: https://www.postgresql.org/docs/ (дата обращения: 14.02.2024 г.).

11. Karmitt. [Электронный ресурс] URL: https://karmitt.com/ (дата обращения: 15.02.2024 г.).

12. Меняю24. [Электронный ресурс] URL: https://menau24.ru/ (дата обращения: 15.02.2024 г.).

13. SwopShop. [Электронный ресурс] URL: https://swopshop.com/ (дата обращения: 15.02.2024 г.).

14. Chen P.P.-S. The Entity-Relationship Model – Toward a Unified View of Data. // ACM Transactions on Database Systems, 1976. – Vol. 1, № 1 – 9–36 pp.

15. Полывянный Э.Я. Информативность нотаций, используемых при ER-моделировании. // Вестник научных конференций, 2015. – №41(4). – С. 140–142.

ПРИЛОЖЕНИЯ

Приложение А. Формы веб-приложения

На рисунках 1–5 представлены скриншоты форм веб-приложения, доступных для заполнения пользователем.

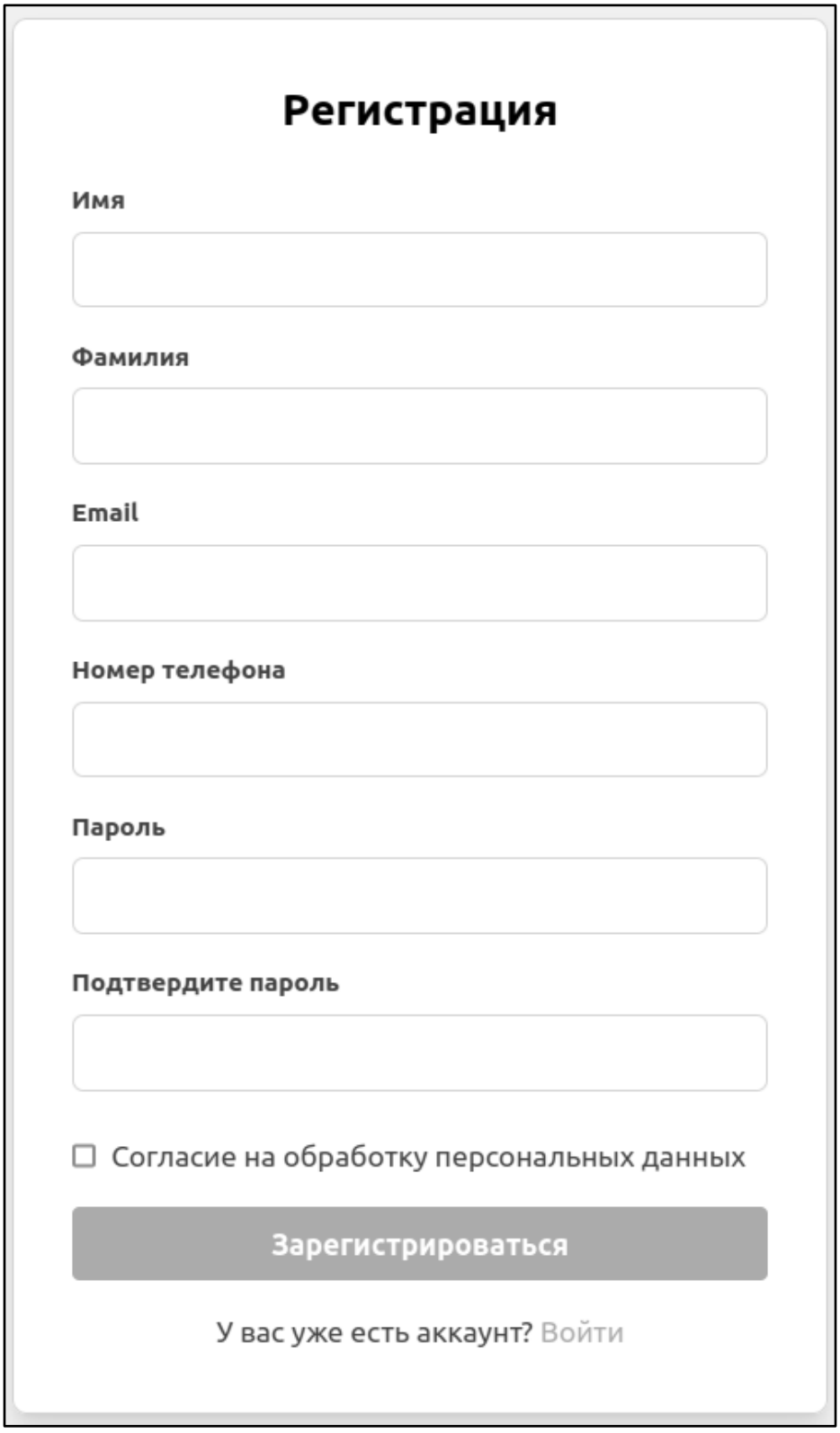

Рисунок 1 – Форма регистрации пользователя

Рисунок 2 – Форма авторизации пользователя

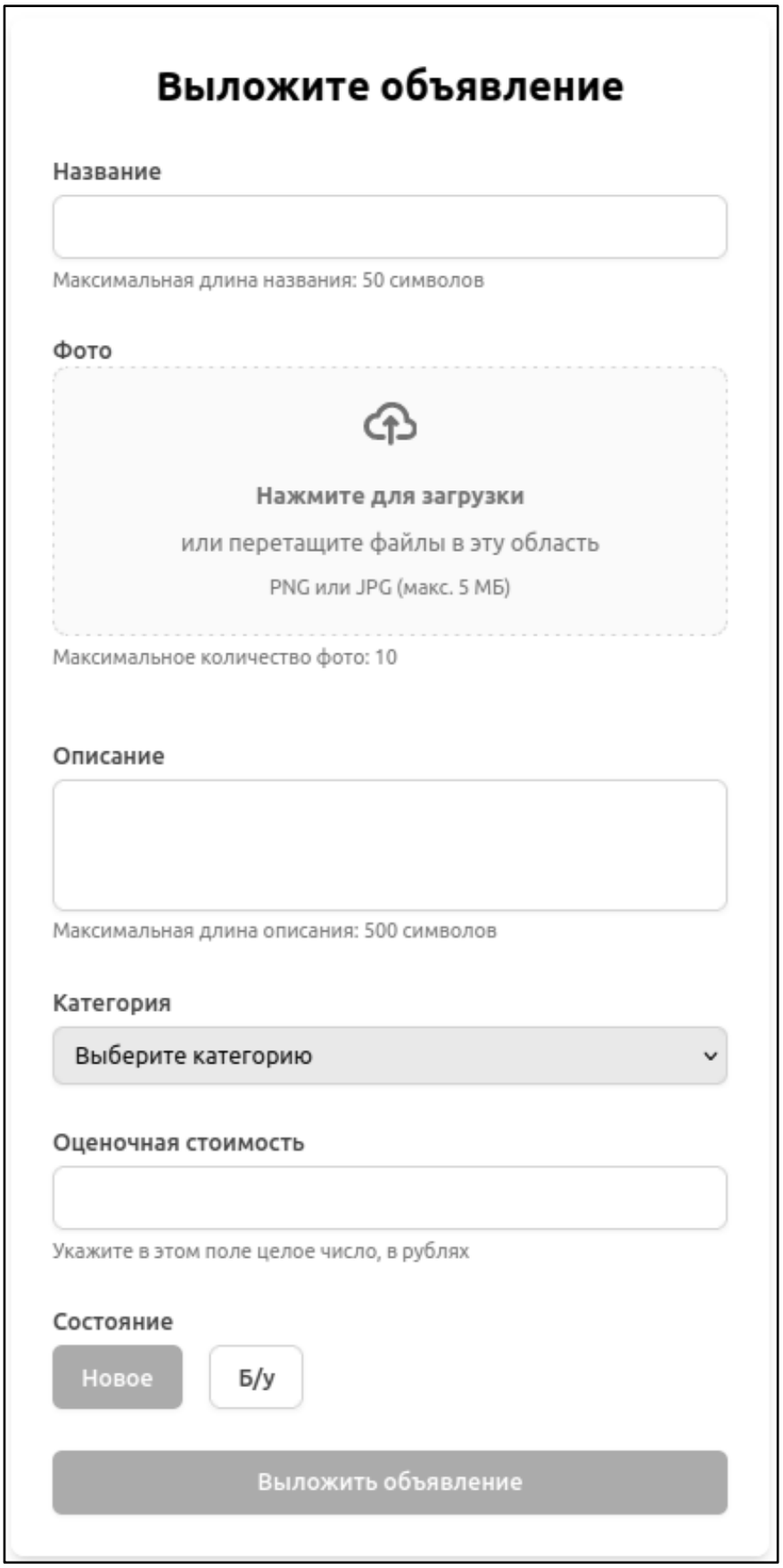

Рисунок 3 – Форма для загрузки объявления

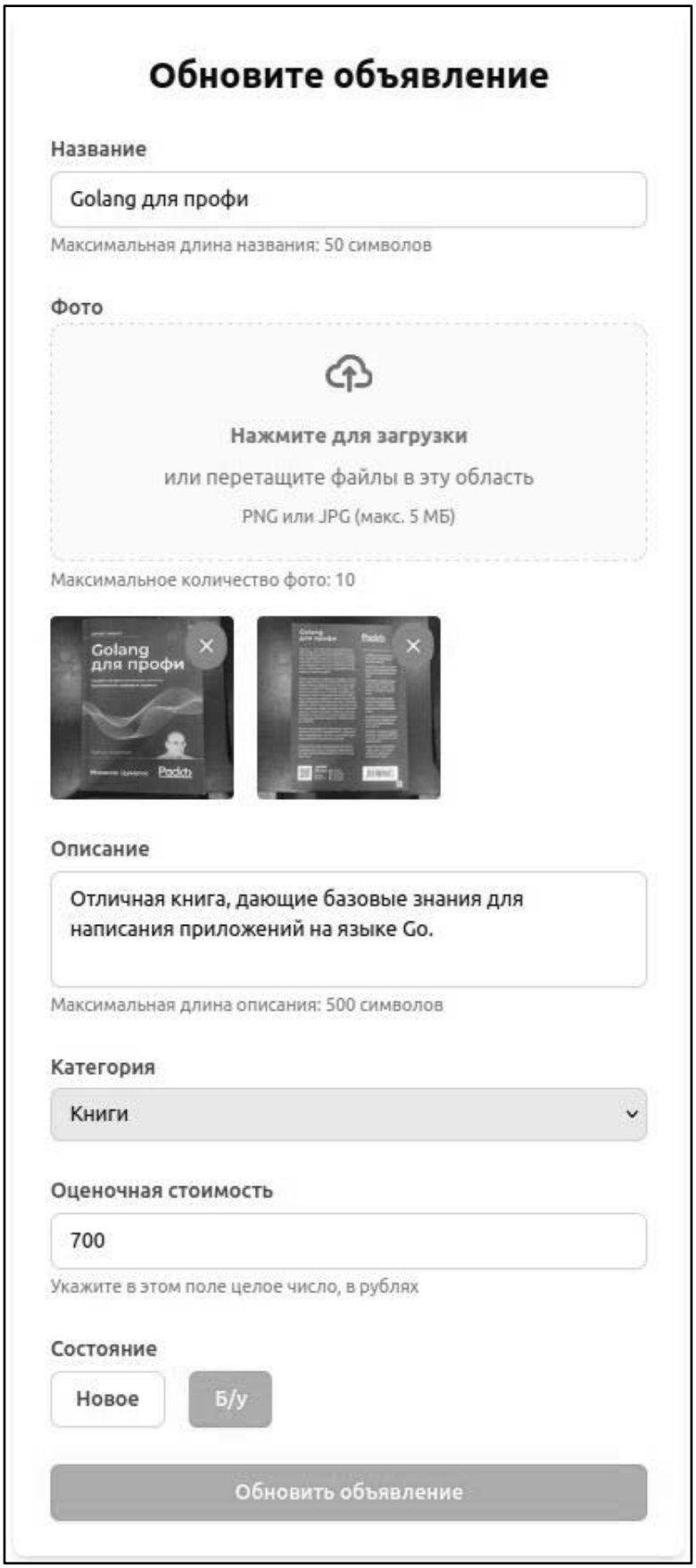

Рисунок 4 – Форма редактирования объявления

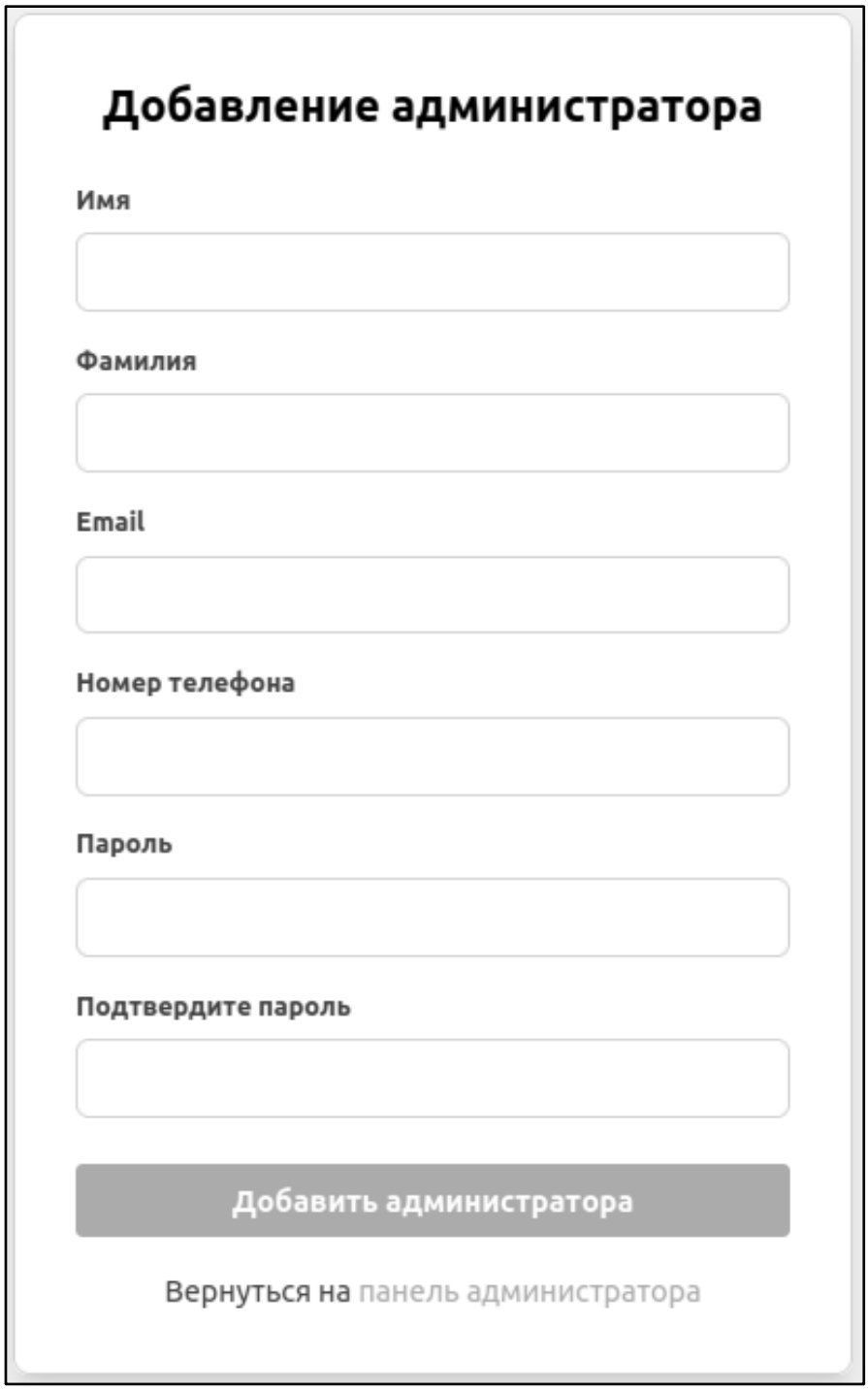

Рисунок 5 – Форма регистрации администратора

Приложение Б. Состояния панели навигации

На рисунках 1–3 представлены скриншоты разных состояний панели навигации.

Рисунок 1 – Навигационная панель для неавторизированного пользователя

Рисунок 2 – Панель навигации для авторизованного пользователя

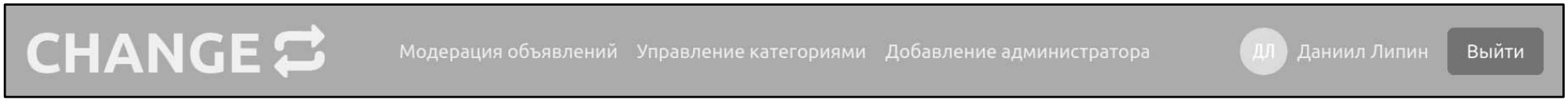

Рисунок 3 – Панель навигации для администратора

Приложение В. Скриншоты страниц веб-приложения

На рисунках 1–11 представлены скриншоты страниц веб-приложения.

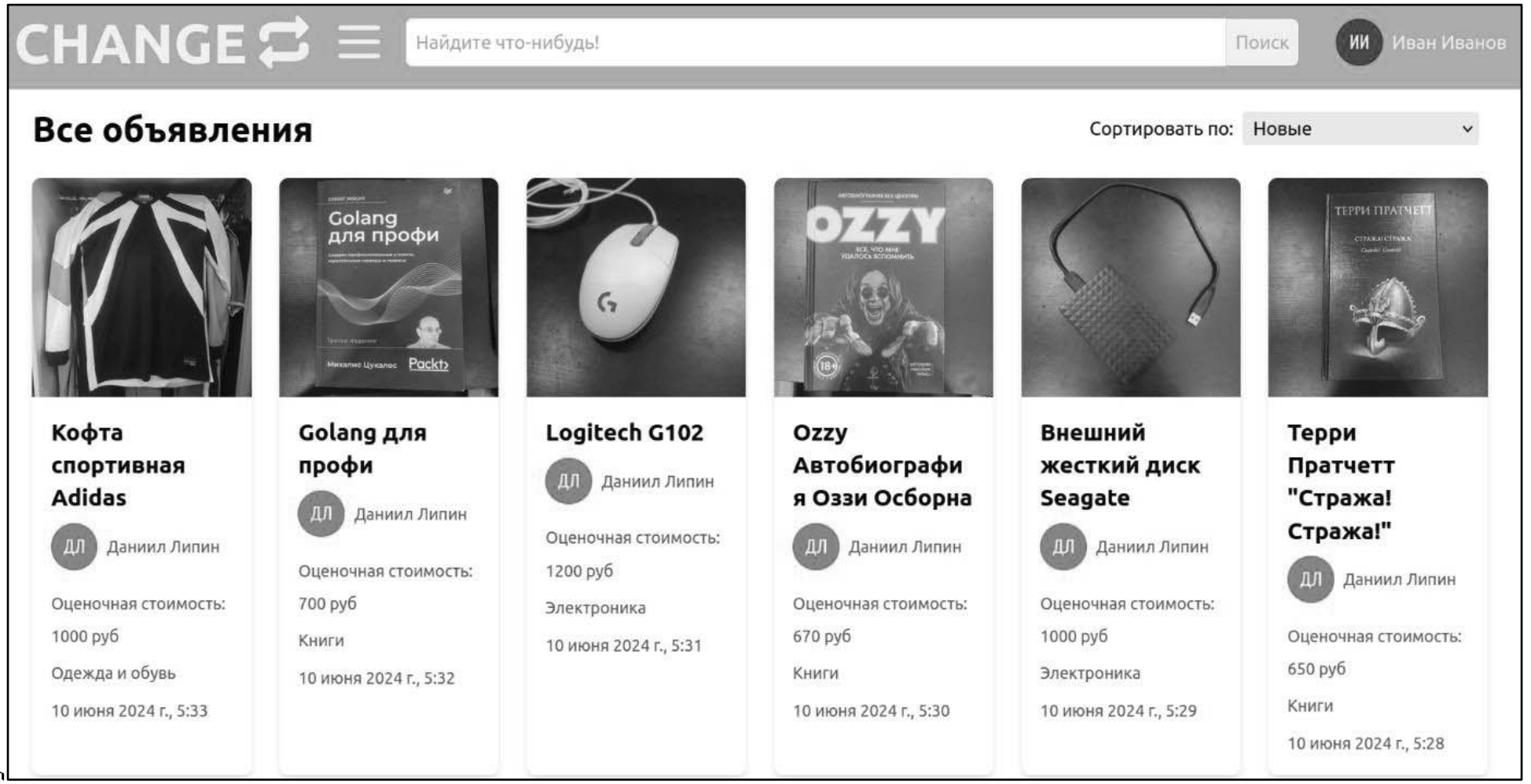

Рисунок 1 – Главная страница приложения

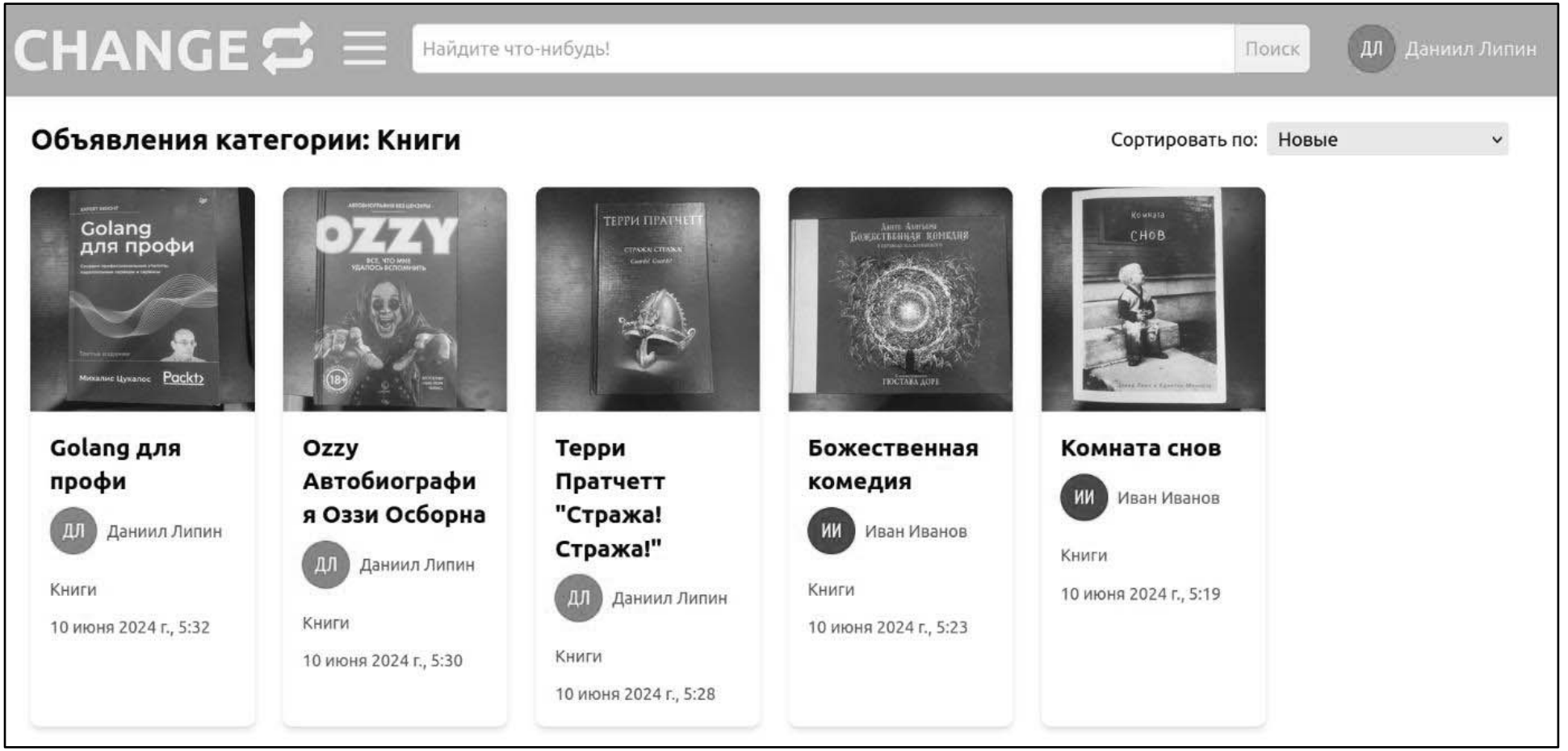

Рисунок 2 – Страница с объявлениями категории «Книги»

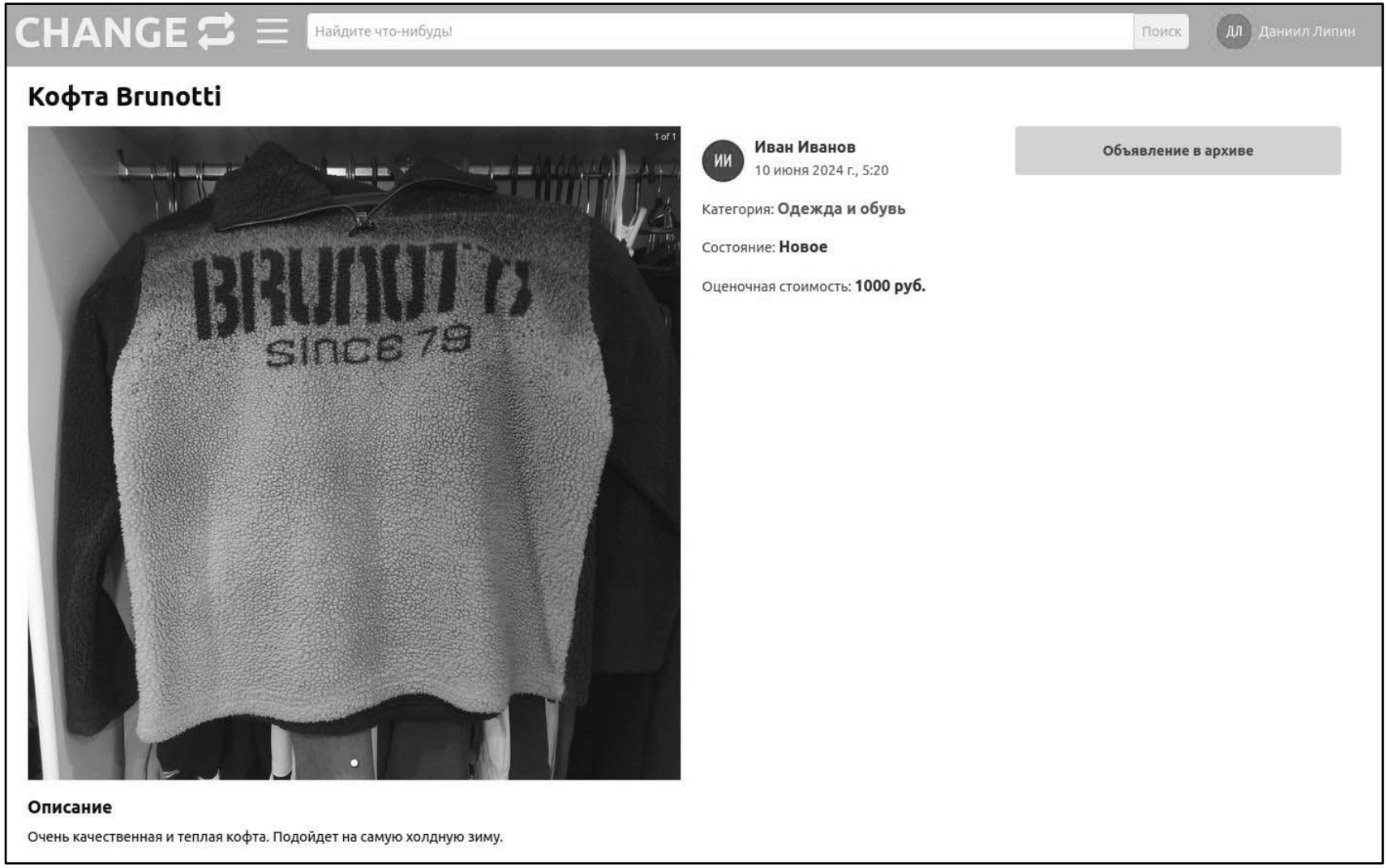

Рисунок 3 – Архивированное объявление

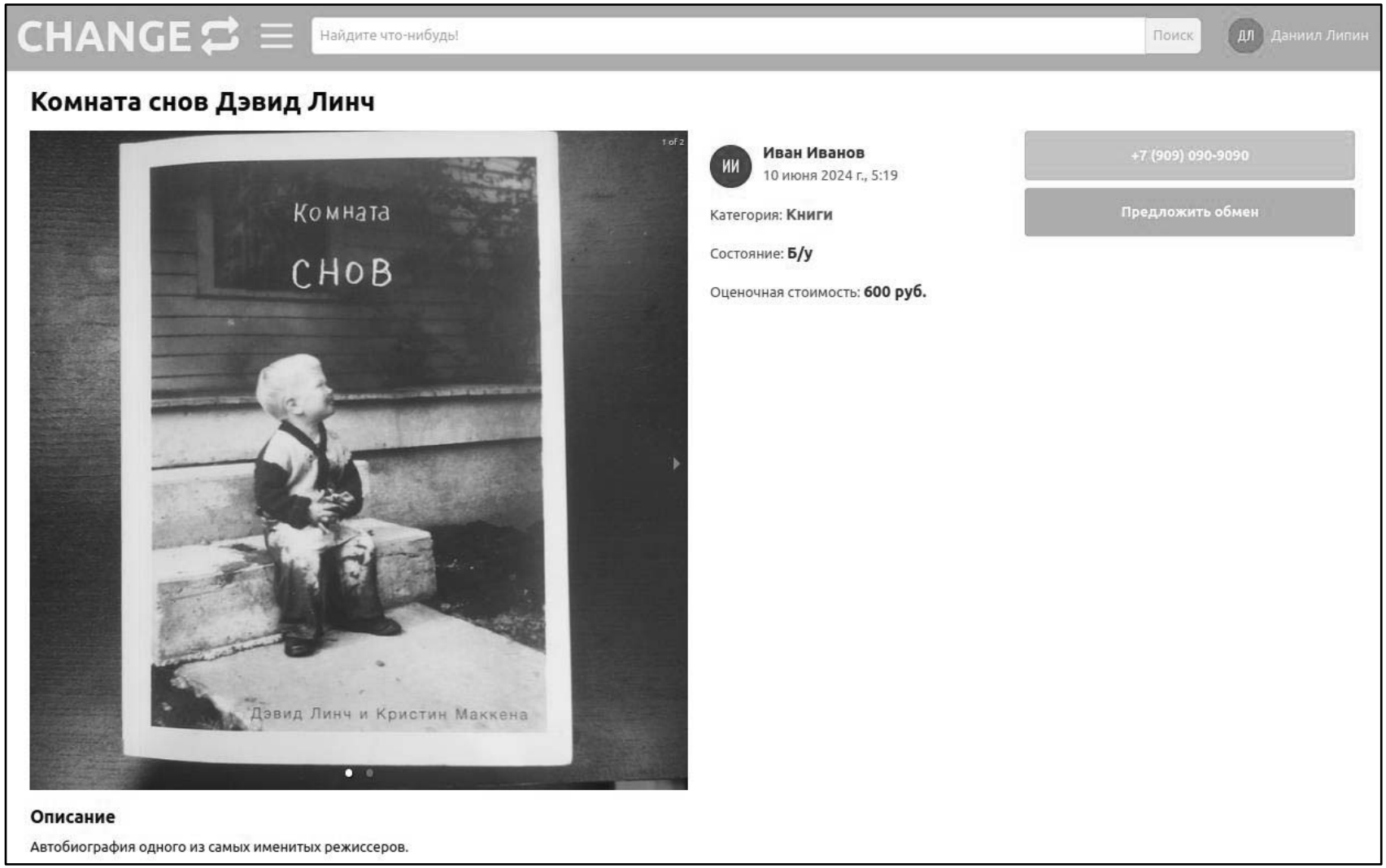

Рисунок 4 – Страница объявления

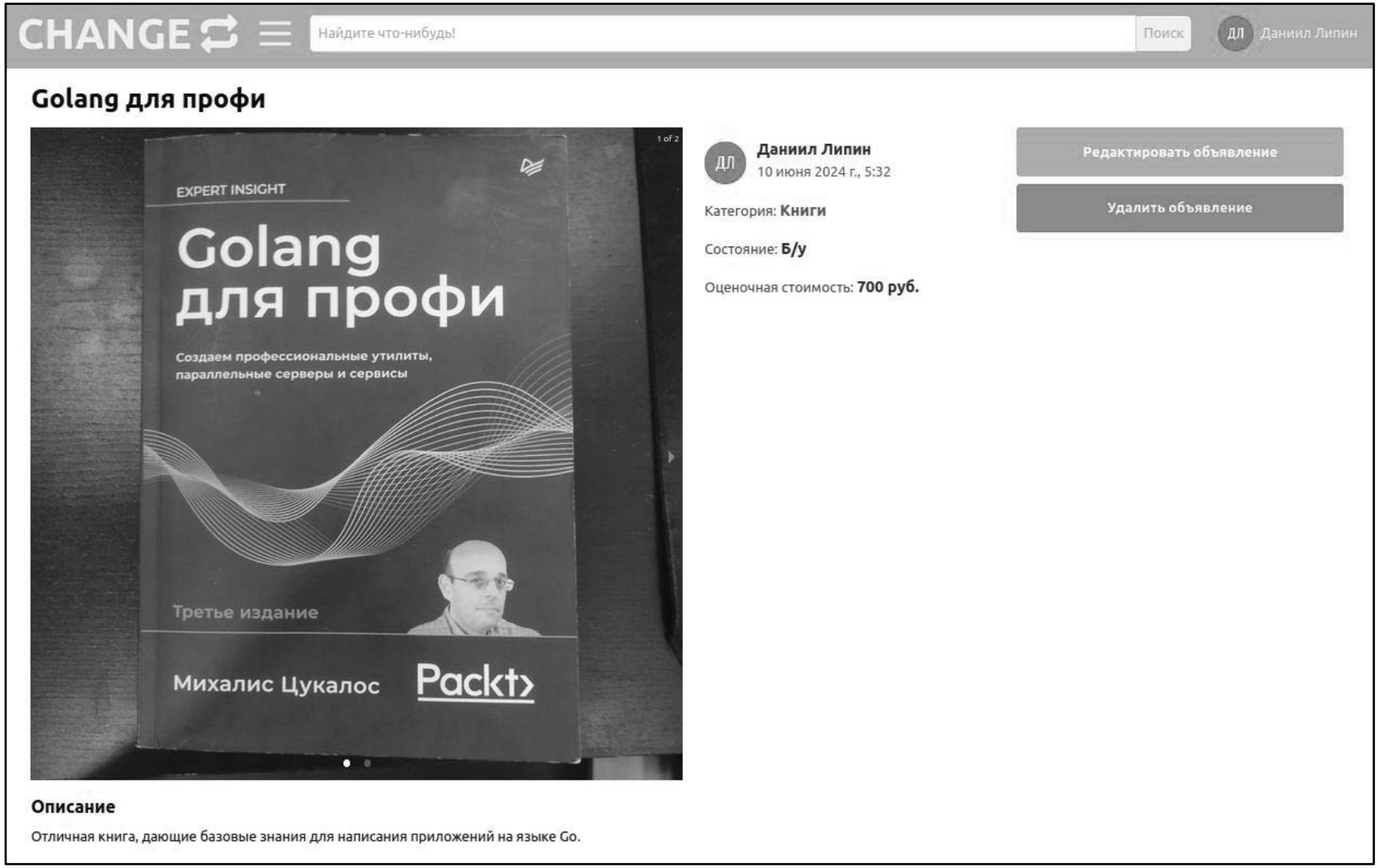

Рисунок 5 – Отображение страницы объявления, для его владельца

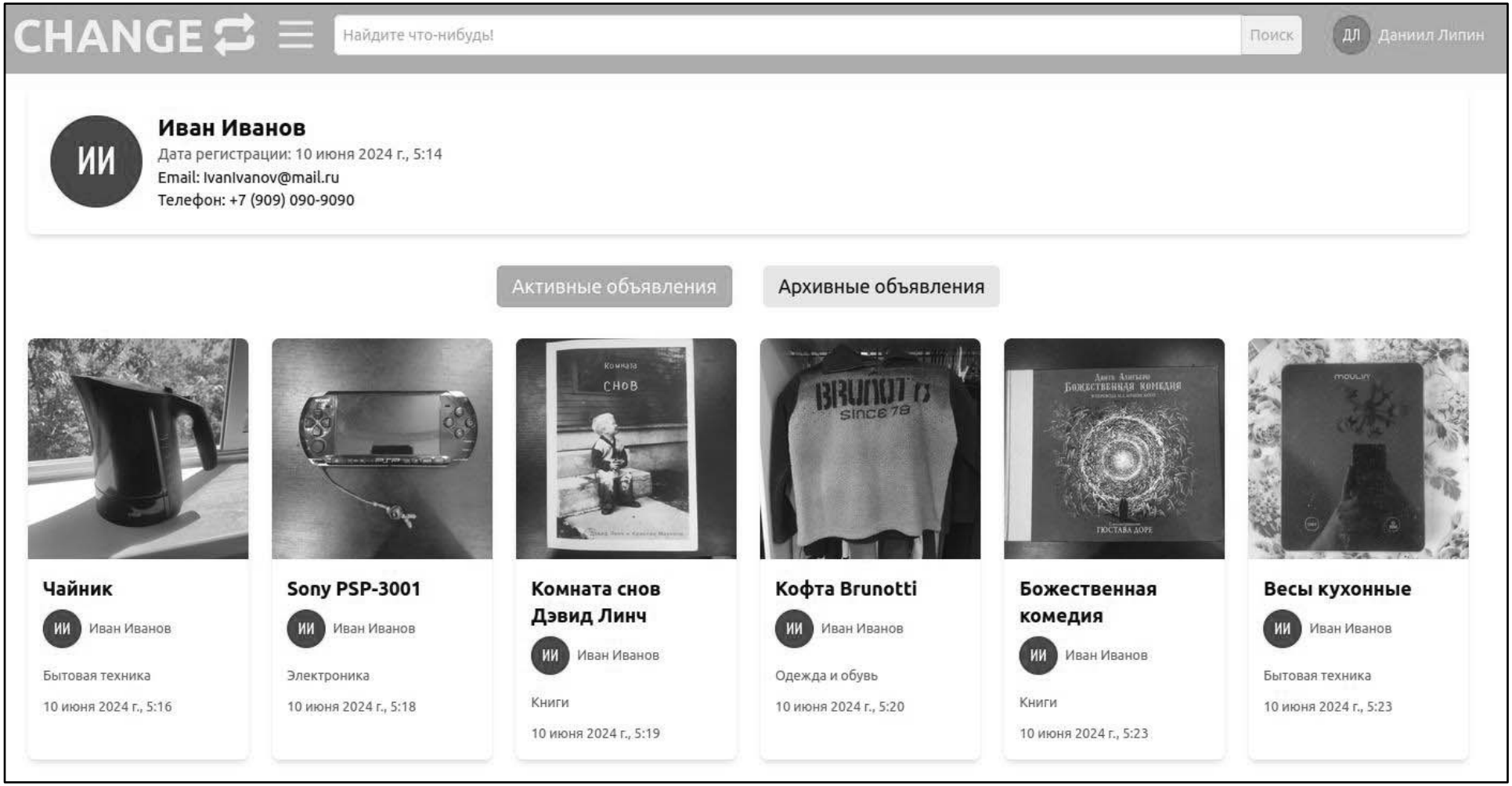

Рисунок 6 – Страница профиля

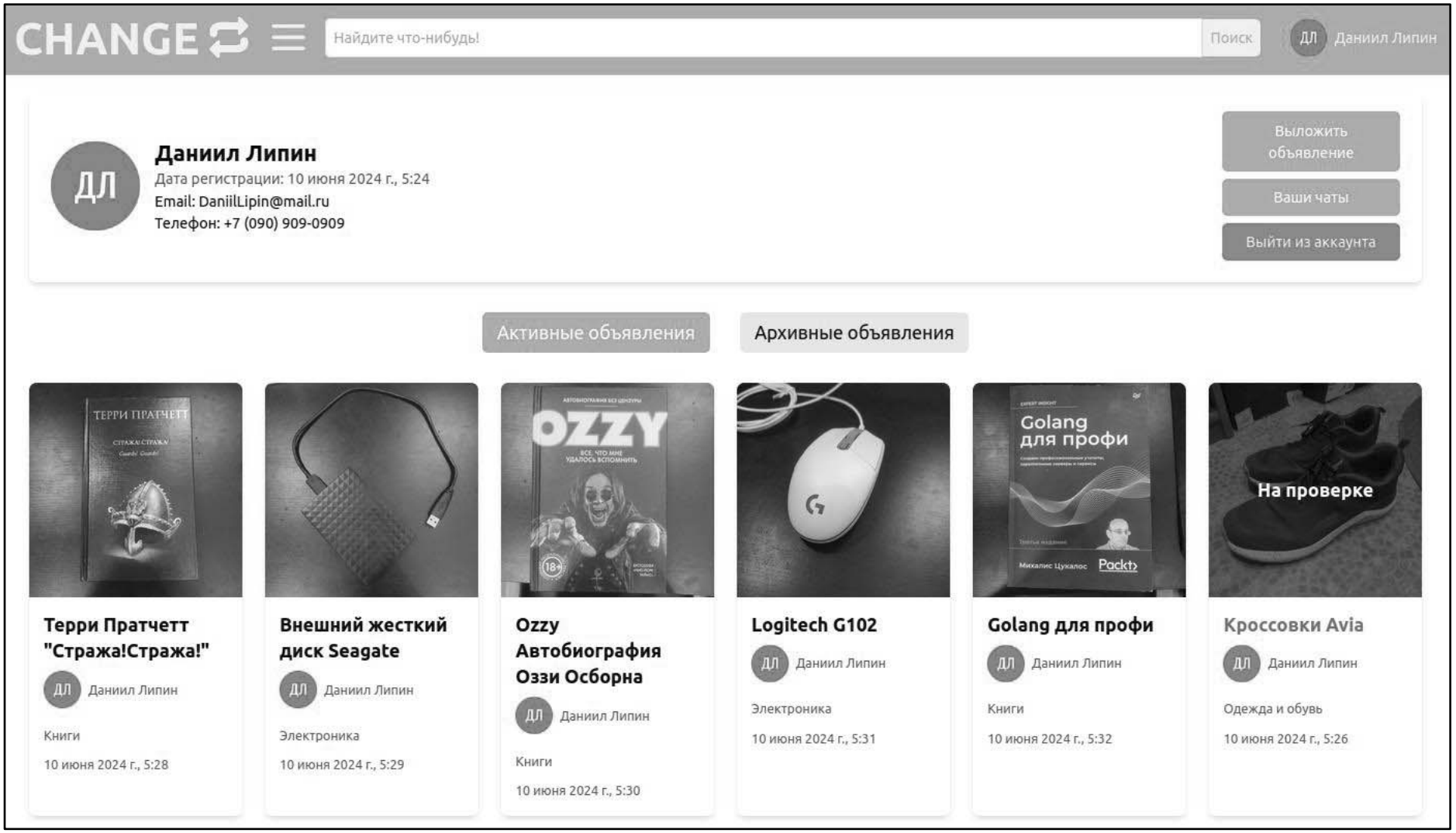

Рисунок 7 – Страница личного аккаунта пользователя

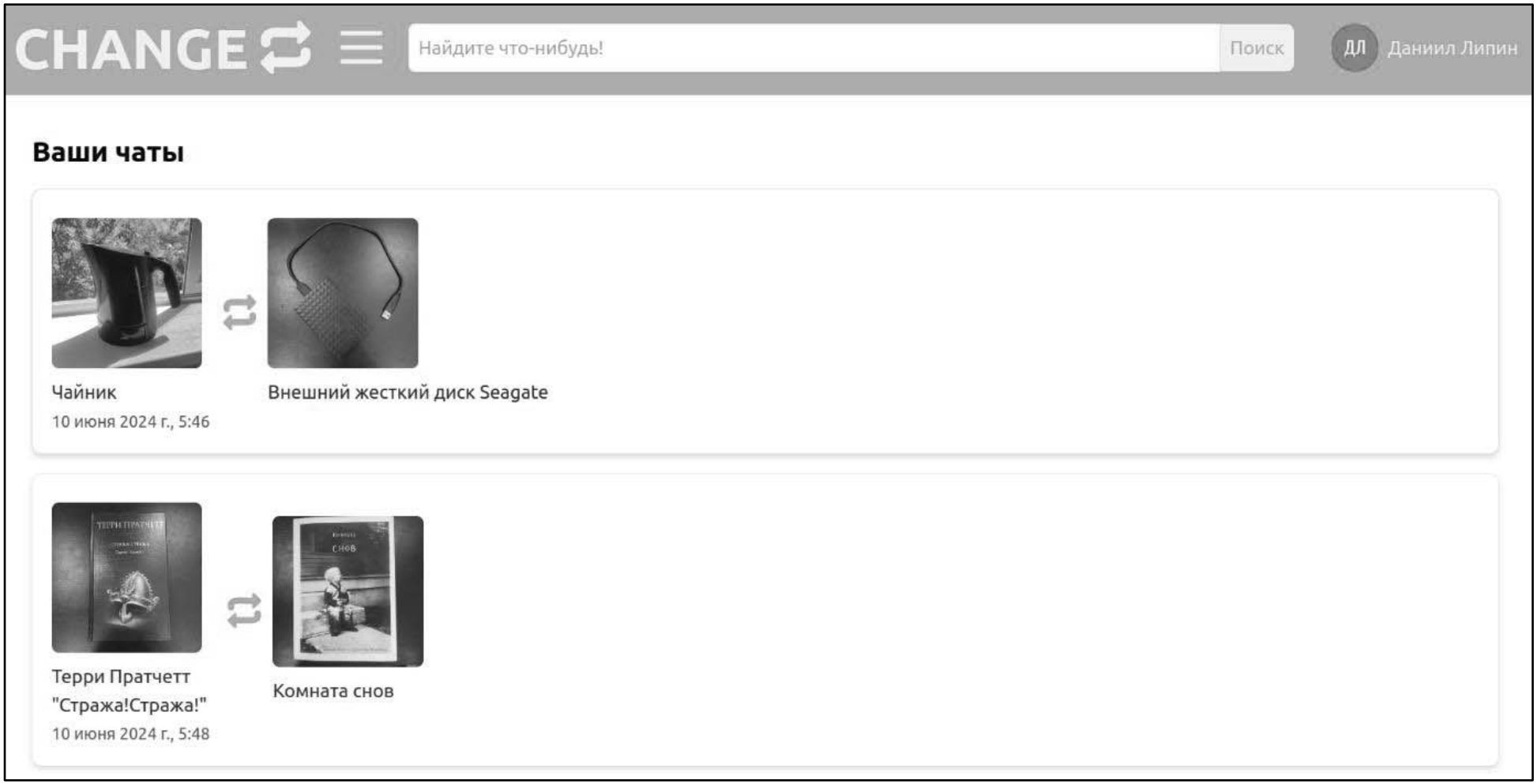

Рисунок 8 – Чаты пользователя

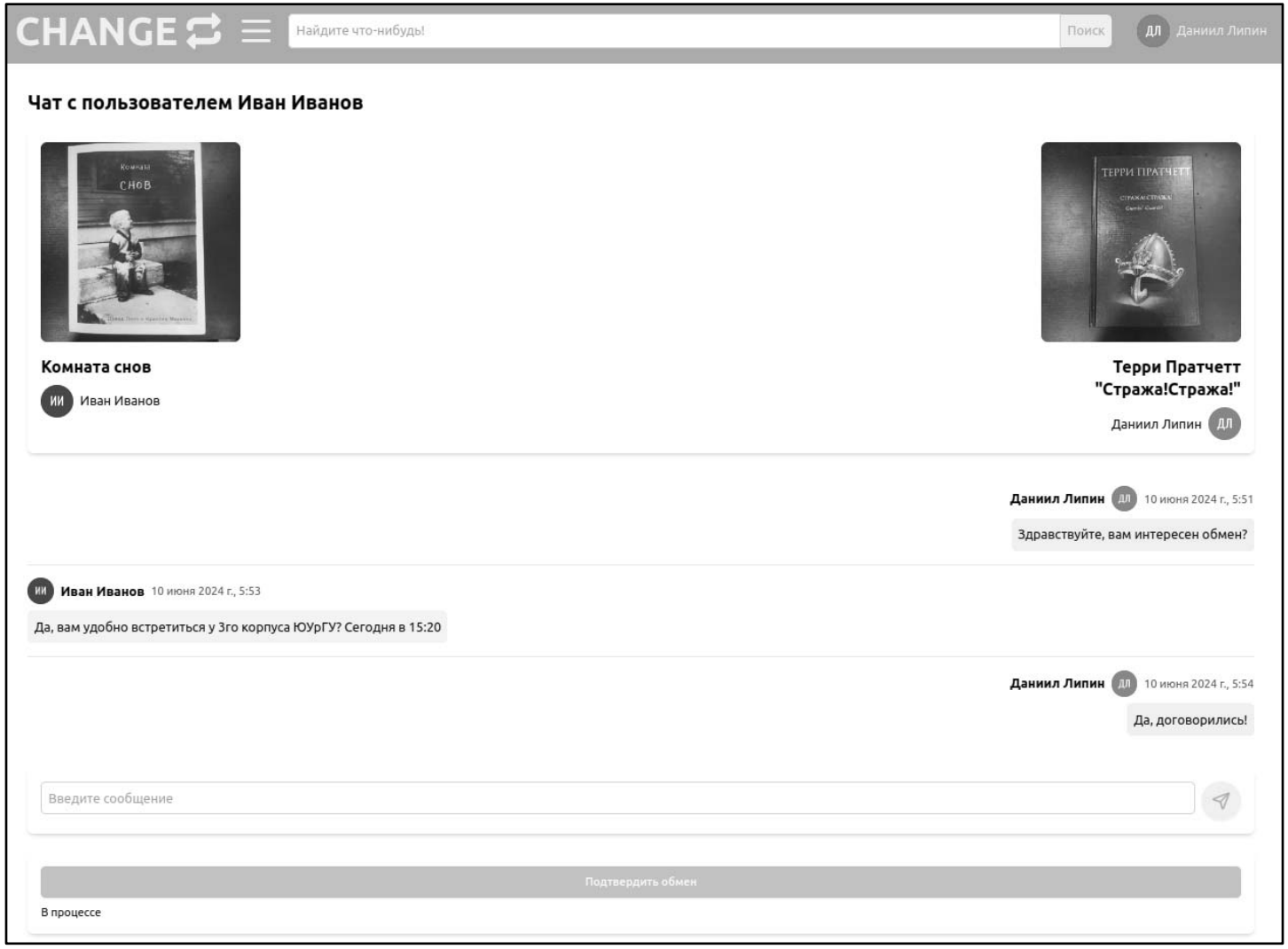

Рисунок 9 – Страница чата

Рисунок 10 – Страница модерации объявлений

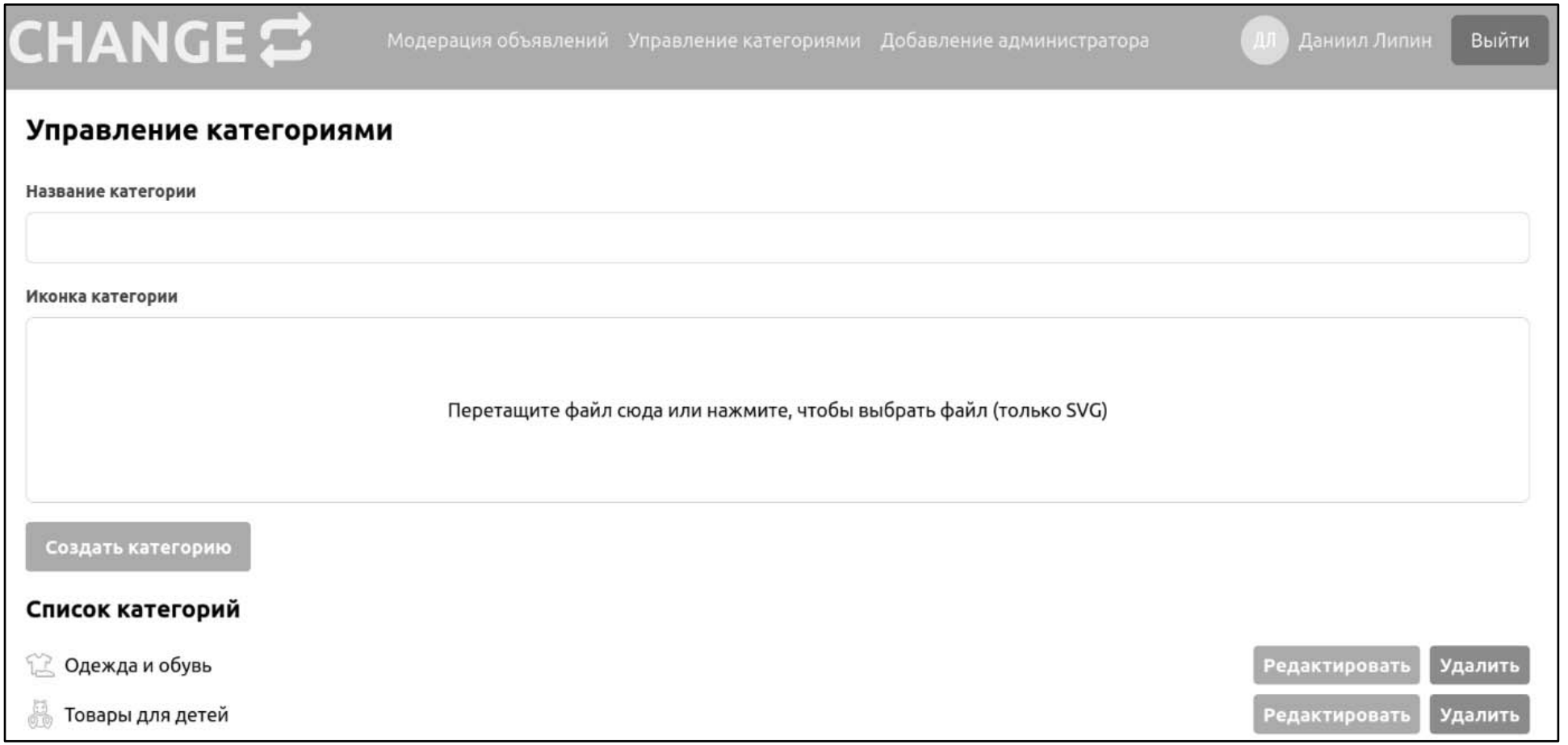

Рисунок 11 – Страница управления категориями