



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

КУРСОВАЯ РАБОТА

по дисциплине: Разработка клиент-серверных приложений

по профилю: Разработка программных продуктов и проектирование информационных систем

направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: «Разработка клиент-серверного фуллстек веб-приложения для фотогалереи»

Студент: Пак Сергей Андреевич

Группа: ИКБО-16-20

Работа представлена к защите _____ 19.05.2023 _____ / Куликов А.А. /

Руководитель: _____ доцент Куликов А.А.

Работа допущена к защите _____ / Куликов А.А. /

Оценка по итогам защиты: _____

_____ / _____ /

_____ / _____ /

Москва 2023 г.



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА - Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине: Разработка клиент-серверных приложений
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: Программная инженерия (09.03.04)
Студент: Пак Сергей Андреевич
Группа: ИКБО-16-20
Срок представления к защите: 11.05.2023
Руководитель: к.т.н, доцент Куликов А.А.

Тема: «Разработка клиент-серверного фуллстек веб-приложения для фотогалереи»

Исходные данные: MongoDB, Express.js, React, Node.js, Bootstrap, Git, GitHub.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:

1. Провести анализ предметной области для выбранной темы с обоснованием выбора клиент-серверной архитектуры для разрабатываемого приложения; 2. Выбрать программный стек для реализации фуллстек разработки; 3. Дать описание архитектуры разрабатываемого клиент-серверного приложения, с помощью UML нотаций; 4. Провести реализацию фронтенд и бекенд части клиент-серверного приложения; 5. Разместить проект клиент-серверного приложения в репозитории GitHub с приложением в тексте отчёте ссылки на данный репозиторий; 6. Провести проверку функционирования минимально жизнеспособного продукта с использованием сгенерированных тестовых данных; 7. Разработать презентацию с графическими материалами; 8. Интегрировать проект на GitHub с Render, с целью развёртывания разработанного клиент-серверного приложения в облаке.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: _____ / Болбаков Р. Г. /, «_____» _____ 2023 г.

Задание на КР выдал: _____ / Куликов А.А. /, «_____» _____ 2023 г.

Задание на КР получил: _____ / Пак С.А. /, «_____» _____ 2023 г.

АННОТАЦИЯ

Курсовая работа посвящена созданию клиент-серверного фуллстек веб-приложения для фотогалереи.

В курсовой работе рассматриваются прикладное программное обеспечение, необходимое для разработки интернет-ресурса, производится анализ предметной области, формирование основных требований к нему, выбор клиент-серверной архитектуры приложения, описание архитектуры приложения с помощью UML нотаций.

Интернет-ресурс обладает межстраничной навигацией, адаптивным дизайном, а его клиентская и серверная части реализованы на языке JavaScript.

Работа содержит 52 страницы отчёта, 30 рисунков, 12 листингов, 1 таблицу, 14 источников.

СОДЕРЖАНИЕ

ГЛОССАРИЙ	7
ВВЕДЕНИЕ	8
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	9
1.1 Описание предметной области	9
1.2 Выводы к разделу 1	11
2 ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЙ	12
2.1 Выбор технологий для клиентской части	12
2.2 Выбор технологий для серверной части	12
2.3 Выбор технологий для управления базой данных	13
2.4 Выбор технологий для контроля версий и развёртывания приложения	13
2.5 Выводы к разделу 2	14
3 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ	15
3.1 Схема архитектуры приложения	15
3.2 UML-диаграмма вариантов использования	15
3.3 Проектирование базы данных	16
3.4 Выводы к разделу 3	17
4 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ	18
4.1 Создание проекта и добавление зависимостей	18
4.2 Создание схем коллекций	18
4.3 Создание конечных точек	22
4.3 Реализация аутентификации в приложении	33
4.4 Выполнение версионного контроля	33
4.5 Выводы к разделу 4	33
5 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ	34

5.1 Создание проекта и добавление зависимостей	34
5.2 Создание компонентов и страниц	35
5.3 Реализация навигации по страницам	35
5.4 Отображение созданных страниц	36
5.5 Выполнение версионного контроля	41
5.6 Выводы к разделу 5	41
6 РАЗВЁРТЫВАНИЕ В ОБЛАЧНОМ ХРАНИЛИЩЕ	42
6.1 Развёртывание проекта	42
6.2 Выводы к разделу 6	43
7 ПРОВЕРКА ФУНКЦИОНИРОВАНИЯ ПРОГРАММНОГО ПРОДУКТА .	44
7.1 Проверка работоспособности серверной части	44
7.2 Проверка работоспособности клиентской части	45
ЗАКЛЮЧЕНИЕ	49
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	51

ГЛОССАРИЙ

CSS	– Cascading Style Sheets
HTML	– HyperText Markup Language
JS	– JavaScript
SQL	– Structured Query Language
UML	– Unified Modeling Language
БД	– база данных
ПК	– профессиональная компетенция
СУБД	– система управления базами данных

ВВЕДЕНИЕ

Фотографии помогают нам запечатлеть самые запоминающиеся, тёплые и просто хорошие моменты в нашей жизни. Фотографии можно хранить как физически, например, в альбоме, так и электронно с помощью различных приложений.

Целью данной курсовой работы является разработка клиент-серверного фуллстек веб-приложения для фотогалереи с возможностью регистрации и авторизации для публикации фотографий, объединения их в альбомы.

Для достижения этой цели необходимо решить следующие задачи:

1. Проведение анализа предметной области;
2. Формирование основных требований к веб-приложению;
3. Выбор клиент-серверной архитектуры приложения;
4. Выбор программного стека и технологий для реализации веб-приложения;
5. Описание архитектуры приложения с помощью UML нотаций;
6. Реализация серверной и клиентской частей приложения;
7. Обеспечение версионного контроля процесса разработки с помощью GitHub;
8. Развёртывание веб-приложения на облачной платформе Render.

В ходе выполнения курсовой работы должно быть проведено освоение профессиональных компетенций, предусмотренных федеральным образовательным стандартом, а именно ПК-1, в том числе ПК-1.2, ПК-1.12, ПК-1.16.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Описание предметной области

Предметной областью данной курсовой работы являются приложения или веб-приложения, реализующие добавление фотографий, организацию их в альбомы, а также их публикацию.

Одним из примером такого интернет-ресурса является «Gallerix» [1] (Рисунок 1.1.1). На данном сайте есть возможность зарегистрироваться и авторизоваться, посмотреть фотографии, которые группируются по художникам.

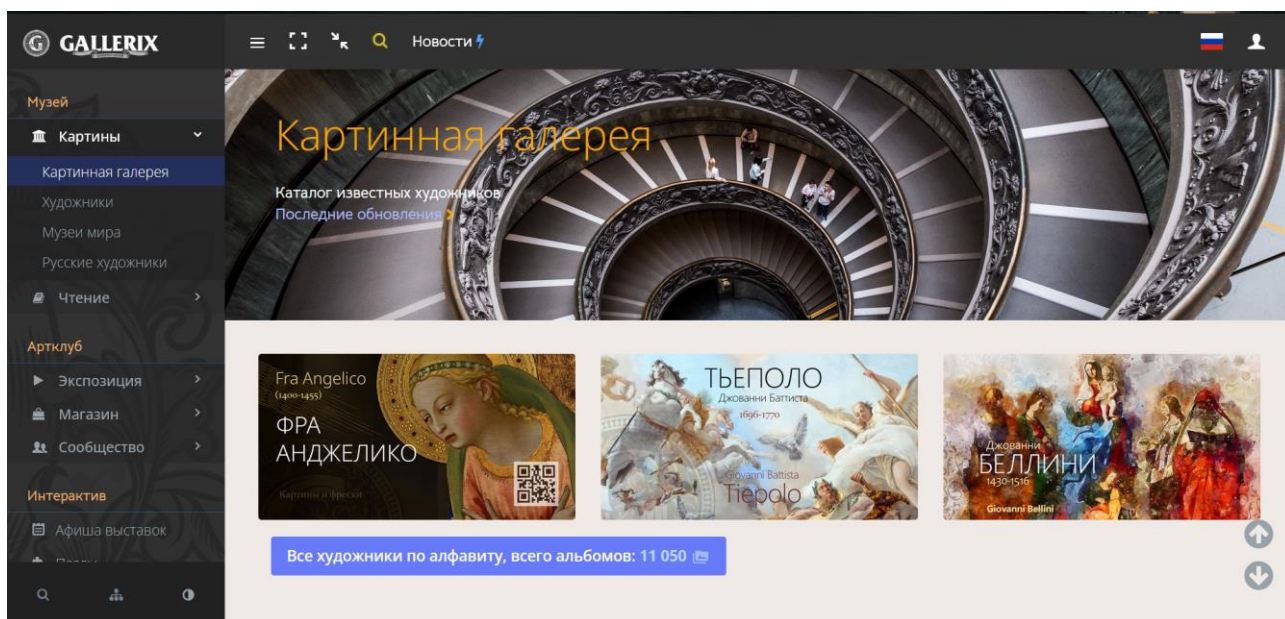


Рисунок 1.1.1 – Главная страница «Gallerix»

Другим примером может быть сайт «Art Online 24» [2] (Рисунок 1.1.2). Данный ресурс является не только галереей, но и площадкой для продажи и покупки картин. Здесь также есть возможность регистрации, авторизации, просмотра картины и её данных (например, автор, материалы, стиль, год и т.д.). Конечно же, присутствует возможность группировки по стилю, автору и другим параметрам.

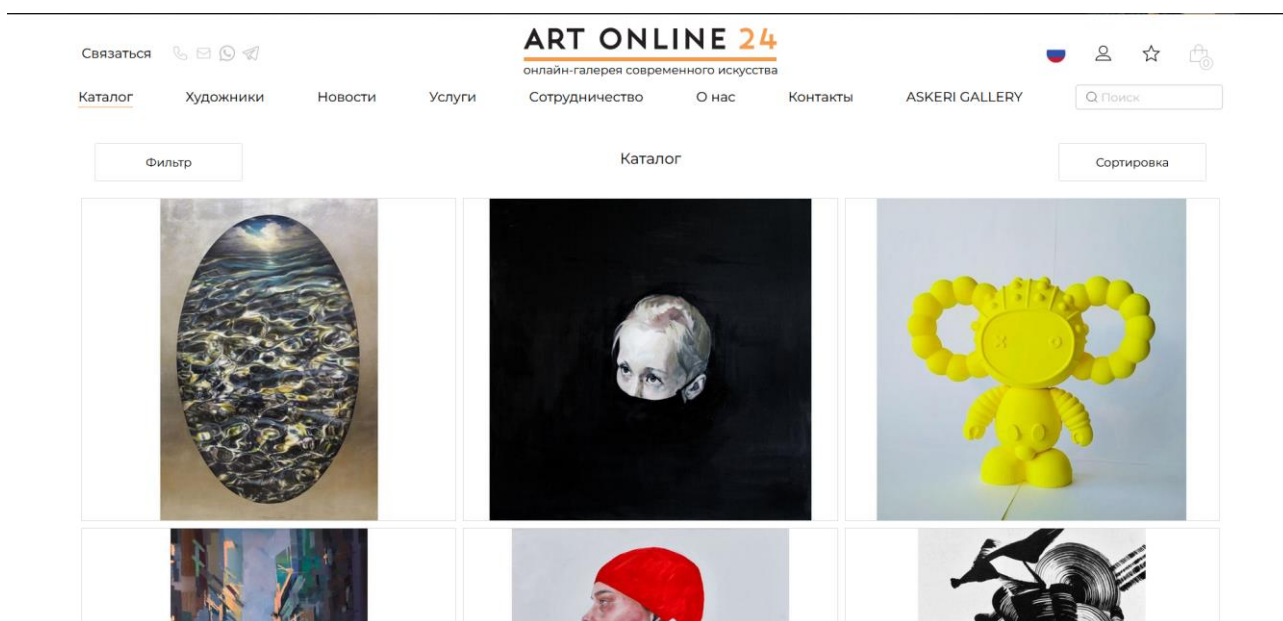


Рисунок 1.1.2 – Главная страница «Art Online 24»

Ещё одним примером является сайт «Artmajeur» [3] (Рисунок 1.1.3). Данный сайт также позволяет покупать картины и не только. Можно найти и купить скульптуры, рисунки, фотографии, каждые из которых сгруппированы по художникам. Для указания данных пользователя при покупке существуют возможности регистрации и авторизации.

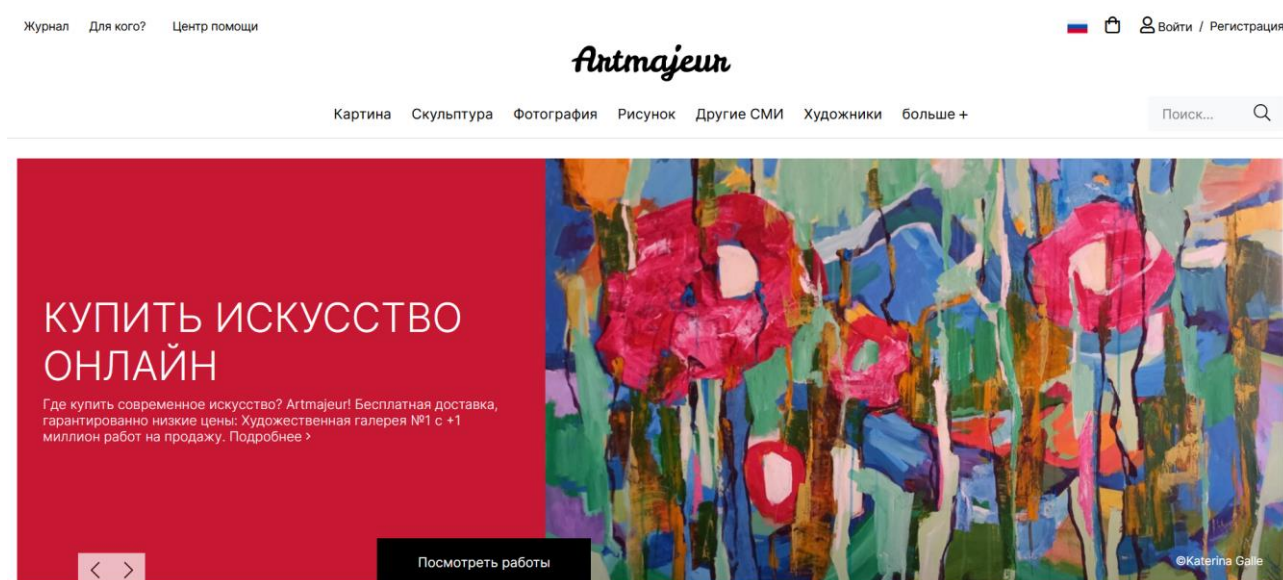


Рисунок 1.1.3 – Главная страница «Artmajeur»

В ходе анализа предметной области были выявлены плюсы и минусы рассмотренных сайтов. Эти плюсы и минусы представлены в таблице 1.1.1.

Таблица 1.1.1 – Плюсы и минусы сайтов фотогалерей

Сайт	Наличие личного кабинета	Адаптивный дизайн	Группировка по параметрам	Возможность оставлять комментарии
Gallerix	+	-	+	+
Art Online 24	+	+	+	-
Artmajeur	+	+	+	-

Таким образом, были сформированы требования к веб-приложению:

1. Реализовать личный кабинет пользователя;
2. Сделать дизайн сайта адаптивным для различных устройств;
3. Сгруппировать публикации по каким-либо параметрам;
4. Возможность публикации фотографии;
5. Возможность добавления фотографий в альбом;
6. Возможность оставить комментарий под фотографией.

1.2 Выводы к разделу 1

В данном разделе был проведён анализ предметной области, в ходе которого были рассмотрены различные сайты, реализующие функционал галереи.

Для реализации упомянутых требований можно использовать клиент-серверную архитектуру. Такая архитектура обладает несколькими преимуществами. Например, снижение требований к устройствам клиентов, так как все вычисления выполняются на сервере, все данные хранятся на сервере, который защищён лучше большинства клиентов. Также на сервере проще организовать контроль полномочий [4]. Серверная часть будет получать информацию из базы данных.

Функционал и требования к веб-приложению определены, поэтому можно перейти к выбору и обоснованию технологий для разработки приложения.

2 ВЫБОР И ОБОСНОВАНИЕ ТЕХНОЛОГИЙ

2.1 Выбор технологий для клиентской части

Для реализации клиентской части веб-приложения были выбраны следующие технологии:

1. React – JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов [5];
2. Bootstrap – это мощный, расширяемый и многофункциональный набор инструментов для создания сайтов и веб-приложений [6];
3. WebStorm – это интегрированная среда для разработки на JavaScript и связанных с ним технологиях, обладающая возможностями отладки, тестирования, навигации и поиска, автодополнения, рефакторинга кода [7];
4. Axios – это HTTP-клиент, основанный на Promise для Node.js и браузера [8]. С его помощью заметно упрощается выполнение запросов к различным ресурсам, а также есть возможность асинхронного их выполнения.

React позволяет создавать пользовательские компоненты, что, в свою очередь, позволяет использовать их в нескольких местах сразу. Также React обладает так называемой реактивностью – это явление, когда обновляется не весь DOM, а только та его, часть, которая была изменена. Это очень удобно при разработке.

Bootstrap содержит большой набор уже готовых и стилизованных компонентов для пользовательского интерфейса, что значительно ускоряет процесс разработки.

2.2 Выбор технологий для серверной части

Для реализации серверной части веб-приложения были выбраны следующие технологии:

1. Mongoose – это JavaScript библиотека, создающая соединение между MongoDB и Node.js [9]. Эта библиотека существенно упрощает работу с коллекциями и самими документами в базе данных;
2. Express – минималистичный и гибкий веб-фреймворк для приложений Node.js, предоставляющий обширный набор функций для мобильных и веб-приложений [10]. Этот фреймворк позволяет легко и быстро создать надёжный прикладной программный интерфейс и внедрить какие-либо другие технологии;
3. Argon2 – это функция хэширования паролей, которая является победителем Password Hashing Competition [11]. Данная функция позволяет настраивать параметры хэширования.

2.3 Выбор технологий для управления базой данных

Для разработки серверной части веб-приложения была выбрана СУБД MongoDB. MongoDB – документоориентированная система управления базами данных, не требующая описания схемы таблиц [12].

За счёт того, что это NoSQL система, не нужно писать громоздкие SQL-запросы для получения каких-либо данных.

Также база данных на MongoDB очень легко и быстро развёртывается на облачной платформе за счёт технологии MongoDB Atlas.

2.4 Выбор технологий для контроля версий и развёртывания приложения

В качестве системы контроля версий для данной работы была выбрана система Git с использованием GitHub.

Git – это бесплатная распределённая система контроля версий с открытым исходным кодом [13].

GitHub – это крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки [14]. Веб-сервис обладает широким инструментарием для управления репозиторием и контроля за разработкой.

Для развёртывания приложения была выбрана облачная платформа Render, так как она бесплатная, поддерживает приложения, сделанные на Node.js, и её легко использовать.

2.5 Выводы к разделу 2

Во втором разделе были выбраны технологии для разработки клиентской части, серверной части веб-приложения. Также были выбраны технологии для управления базами данных, контроля версий и развёртывания приложения. Выбор был основан на преимуществах, которые были упомянуты.

После выбора технологий для разработки веб-приложения можно перейти к проектированию клиент-серверного приложения.

3 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

3.1 Схема архитектуры приложения

Клиентское устройство с помощью браузера делает запросы к серверу React, который в свою очередь отправляет их серверу приложений. Сервер приложений с помощью Express.js и Mongoose обрабатывает данные и отправляет их серверу базы данных. Сервер базы данных обрабатывает запрос и выполняет определённые действия с данными.

Далее формируется ответ, который передаётся в обратном направлении, т.е. сервер базы данных, выполнив манипуляции с данными, отправляет ответ серверу приложений. Сервер приложений – серверу React, который, заметив какие-либо изменения, их отображает в виртуальной структуре документа, которая затем отображается на клиентском устройстве.

На рисунке 3.1.1 представлена схема, описывающая архитектуру приложения.

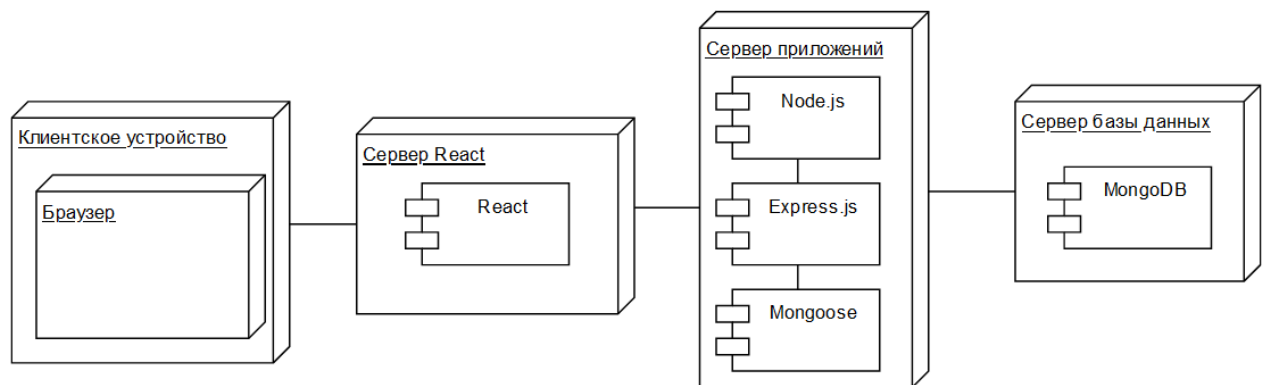


Рисунок 3.1.1 – Диаграмма развёртывания приложения

3.2 UML-диаграмма вариантов использования

Пользователь, не зарегистрированный в системе, может:

- зарегистрироваться;
- посмотреть список публикаций других пользователей;
- посмотреть данные какой-то публикации;

- посмотреть комментарии к публикации.

Зарегистрированный пользователь обладает всеми возможностями незарегистрированного пользователя. Также он может:

- авторизоваться;
- посмотреть персональные данные в личном кабинете;
- изменить персональные данные в личном кабинете;
- добавить публикацию;
- редактировать свою публикацию;
- удалить свою публикацию;
- оставить комментарий к любой публикации.

На рисунке 3.2.1 представлена диаграмма вариантов использования.



Рисунок 3.2.1 – Диаграмма вариантов использования

3.3 Проектирование базы данных

База данных веб-приложения находится под управлением СУБД MongoDB.

В MongoDB, в отличие от SQL баз данных, нет такого понятия, как таблицы. Вместо них есть коллекции.

Для всех сущностей создаются соответствующие коллекции в базе данных: users, photos, comments, categories, albums. Поля документов в этих коллекциях соответствуют полям классов сущностей.

На рисунке 3.3.1 представлена логическая схема базы данных.

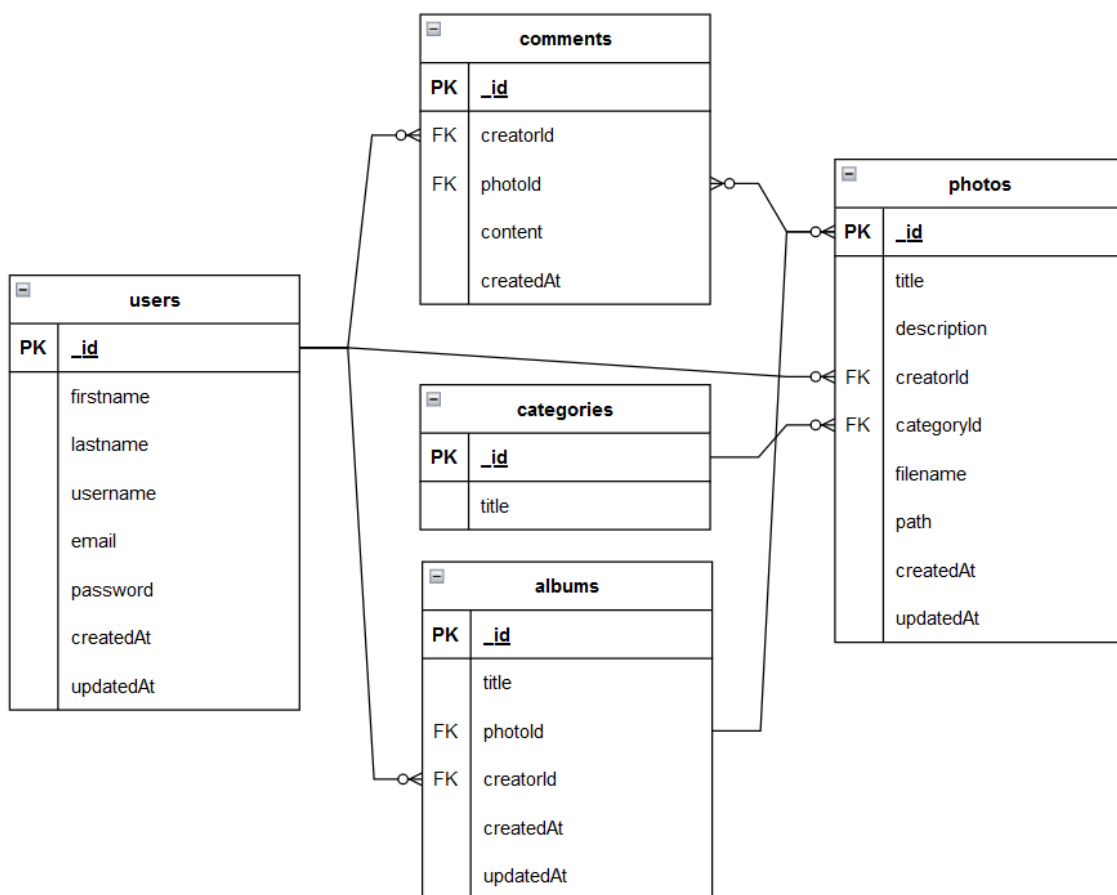


Рисунок 3.3.1 – Логическая схема базы данных

3.4 Выводы к разделу 3

В этом разделе была описана архитектура разрабатываемого приложения, из каких компонентов она состоит, и как они взаимодействуют друг с другом. С помощью диаграммы вариантов использования были определены функциональные возможности веб-приложения. Для описания структуры базы данных была составлена физическая схема базы данных.

4 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ

4.1 Создание проекта и добавление зависимостей

Для создания проекта необходимо просто создать директорию, и, перейдя в эту директорию в командной строке, инициализировать проект с помощью пакетного менеджера Node.js NPM. Также, используя NPM, нужно установить следующие зависимости:

1. Express для разработки программного прикладного интерфейса;
2. Mongoose для работы с базой данных;
3. Argon2 для хэширования паролей.

Все эти зависимости указаны в файле `package.json`, содержимое которого представлено на листинге 4.1.1.

Листинг 4.1.1 – `package.json`

```
{
  "name": "v1",
  "version": "1.0.0",
  "description": "",
  "main": "src/index.js",
  "type": "module",
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "argon2": "^0.30.3",
    "express": "^4.18.2",
    "mongoose": "^7.0.4",
  }
}
```

4.2 Создание схем коллекций

Схема коллекции – это описание полей, из которых состоят документы, содержащиеся в коллекции и сущности. Из этих схем в дальнейшем можно создать модели, с помощью которых осуществляется взаимодействие с коллекциями.

Документы из коллекции `users` должны содержать следующие поля:

1. `_id` – идентификатор пользователя;
2. `firstname` – имя пользователя;

3. `lastname` – фамилия;
4. `username` – логин пользователя;
5. `email` – адрес электронной почты;
6. `password` – пароль;
7. `createdAt` – дата регистрации;
8. `updatedAt` – дата изменения данных пользователя.

На листинге 4.2.1 представлена схема коллекции `users`.

Листинг 4.2.1 – Схема коллекции `users`

```
import mongoose, {model} from "mongoose";
import {Schema} from "mongoose";

/**
 * Пользователь ресурса
 * @field {String} firstname          имя пользователя
 * @field {String} lastname          фамилия
 * @field {String} username          имя пользователя в системе
 * @field {String} email             адрес электронной почты
 * @field {String} password          пароль (хранится в зашифрованном
виде)
 * @field {Date} createdAt           дата регистрации
 * @field {Date} updatedAt           дата внесения последних изменений
 */
const User = new Schema({
  firstname: String,
  lastname: String,
  username: String,
  email: String,
  password: String,
  createdAt: Date,
  updatedAt: Date,
});
```

Документы из коллекции `categories` должны содержать следующие поля:

1. `_id` – идентификатор категории;
2. `title` – название категории.

На листинге 4.2.2 представлена схема коллекции `categories`.

Листинг 4.2.2 – Схема коллекции `categories`

```
import mongoose, {model} from "mongoose";
import {Schema} from "mongoose";

/**
 * Категории фотографий
 * @field {String} title              название категории
 */
const Category = new Schema({
  title: String,
});
```

Документы из коллекции photos должны содержать следующие поля:

1. `_id` – идентификатор фотографии;
2. `title` – название;
3. `description` – описание;
4. `creatorId` – идентификатор создателя фотографии;
5. `categoryId` – идентификатор категории;
6. `filename` – название файла;
7. `path` – путь к файлу;
8. `createdAt` – дата создания фотографии;
9. `updatedAt` – дата изменения данных фотографии.

На листинге 4.2.3 представлена схема коллекции photos.

Листинг 4.2.3 – Схема коллекции photos

```
import mongoose, {model} from "mongoose";
import {Schema} from "mongoose";

/**
 * Фотография
 * @field {String} title           название фотографии
 * @field {String} description     описание фотографии
 * @field {ObjectId} creatorId     идентификатор автора
 * @field {ObjectId} categoryId    идентификатор категории
 * @field {String} filename        название файла с расширением
 * @field {String} path            путь к фотографии
 * @field {Date} createdAt         дата публикации фотографии
 * @field {Date} updatedAt         дата изменения данных фотографии
 */
const Photo = new Schema({
  title: String,
  description: String,
  creatorId: mongoose.Types.ObjectId,
  categoryId: mongoose.Types.ObjectId,
  filename: String,
  path: String,
  createdAt: Date,
  updatedAt: Date,
});
```

Документы коллекции comments должны содержать следующие поля:

1. `_id` – идентификатор комментария;
2. `creatorId` – идентификатор автора;
3. `photoId` – идентификатор фотографии;
4. `content` – содержание комментария;
5. `createdAt` – дата опубликования комментария.

На листинге 4.2.4 представлена схема коллекции comments.

Листинг 4.2.4 – Схема коллекции comments

```
import mongoose, {model} from "mongoose";
import {Schema} from "mongoose";

/**
 * Комментарий к фотографии
 * @field {ObjectId} creatorId           идентификатор автора комментария
 * @field {ObjectId} photoId           идентификатор фотографии
 * @field {String} content              содержание комментария
 * @field {Date} createdAt              дата и время создания
 *                                     комментария
 */
const Comment = new Schema({
  creatorId: mongoose.Types.ObjectId,
  photoId: mongoose.Types.ObjectId,
  content: String,
  createdAt: Date,
});
```

Документы коллекции albums должны содержать следующие поля:

1. `_id` – идентификатор альбома;
2. `title` – название;
3. `photos` – массив идентификаторов фотографий;
4. `creatorId` – идентификатор создателя альбома;
5. `createdAt` – дата создания альбома;
6. `updatedAt` – дата изменения данных альбома.

На листинге 4.2.5 представлена схема коллекции albums.

Листинг 4.2.5 – Схема коллекции albums

```
import mongoose, {model} from "mongoose";
import {Schema} from "mongoose";

/**
 * Альбом (коллекция фотографий)
 * @field {String} title                название альбома
 * @field {Array<ObjectId>} photos      идентификаторы фотографий, принадлежащих
 *                                     альбому
 * @field {Date} createdAt              дата и время создания альбома
 */
const Album = new Schema({
  title: String,
  photos: [mongoose.Types.ObjectId],
  creatorId: mongoose.Types.ObjectId,
  createdAt: Date,
  updatedAt: Date,
});
```

4.3 Создание конечных точек

Для сущности User, описывающей пользователя, необходимо создать следующие конечные точки для:

1. Добавления пользователя в коллекцию;
2. Получения списка пользователей;
3. Получения информации о конкретном пользователе;
4. Обновления данных пользователя;
5. Удаления данных пользователя из коллекции;
6. Авторизации пользователя.

На листинге 4.3.1 представлена реализация описанных выше конечных точек.

Листинг 4.3.1 – Реализация конечных точек для сущности User

```
import {Router} from "express";
import {UserModel} from "../models.js";
import mongoose from "mongoose";
import argon2 from "argon2";

const userRouter = Router();

userRouter.post("/", async (request, response) => {
  console.log(`POST /api/v1/users/`);
  console.log(request.body);

  let {
    firstname,
    lastname,
    username,
    email,
    password,
    roleId
  } = request.body;
  /*===== ДОБАВЛЕНИЕ ПОЛЬЗОВАТЕЛЯ В БАЗУ ДАННЫХ =====*/
  const hashedPwd = await argon2.hash(password);

  const user = new UserModel({
    firstname: firstname,
    lastname: lastname,
    username: username,
    email: email,
    password: hashedPwd,
    roleId: mongoose.Types.ObjectId.createFromHexString(roleId),
    createdAt: new Date(),
    updatedAt: new Date(),
  });
  UserModel
    .create(user)
    .then(result => {
```

Продолжение листинга 4.3.1

```
UserModel
  .create(user)
  .then(result => {
    response.status(201);
    response.json(result);
  });
});

userRouter.get("/", (_, response) => {
  console.log(`GET /api/v1/users/`);

  UserModel
    .aggregate([
      {
        $lookup: {
          from: "roles",
          localField: "roleId",
          foreignField: "_id",
          as: "role",
        },
      },
    ])
    .then(result => {
      response.status(200);
      response.json(result);
    });
});

userRouter.get("/:id", (request, response) => {
  const userId = request.params.id;
  console.log(`GET /api/v1/users/${userId}`);

  UserModel
    .aggregate([
      {
        $match: { _id: mongoose.Types.ObjectId.createFromHexString(userId) },
      },
      {
        $lookup: {
          from: "roles",
          localField: "roleId",
          foreignField: "_id",
          as: "role",
        },
      },
    ])
    .then(result => {
      if (result.length === 0) {
        response.status(404);
        response.json({
          error: `Пользователь с id = ${userId} не найден!`,
        });

        return;
      }
      response.status(200);
      response.json(result[0]);
    });
});

userRouter.put("/:id", async (request, response) => {
  const userId = request.params.id;
```

Продолжение листинга 4.3.1

```
console.log(`PUT /api/v1/users/${userId}`);
console.log(request.body);

if (request.body.password) {
  request.body.password = await argon2.hash(request.body.password);
}

request.body.updatedAt = new Date();
UserModel
  .findByIdAndUpdate(userId, request.body)
  .then(result => {
    response.status(200);
    response.json(result);
  });
});
userRouter.delete("/:id", (request, response) => {
  const userId = request.params.id;
  console.log(`DELETE /api/v1/users/${userId}`);

  UserModel
    .deleteOne({_id: userId})
    .then(result => {
      response.status(200);
      response.json(result);
    });
});
userRouter.post("/login", async (request, response) => {
  console.log(`POST /api/v1/users/login/`);

  let {
    usernameOrEmail,
    password,
  } = request.body;

  /*===== ПРОВЕРКА ДАННЫХ ПОЛЬЗОВАТЕЛЯ =====*/
  const filter = (usernameOrEmail.includes("@"))
    ? { email: usernameOrEmail }
    : { username: usernameOrEmail };
  const user = (await UserModel.find(filter, "_id username email password"))[0];
  if (!user) {
    response.status(403);

    response.json({
      errors: ["Неверный логин или email!"],
    });

    return;
  }
  if (!(await argon2.verify(user.password, password))) {
    response.status(403);

    response.json({
      errors: ["Неверный пароль!"],
    });

    return;
  }
  response.status(200);
  response.json({
```


Продолжение листинга 4.3.1

```
    userId: user._id.toString(),  
    username: user.username,  
  });  
});  
export default userRouter;
```

Для сущности Photo, описывающей фотографию, необходимо создать следующие конечные точки для:

1. Добавления фотографию в коллекцию;
2. Получения списка фотографий;
3. Получения информации о конкретной фотографии;
4. Обновления данных фотографии;
5. Удаления данных фотографии из коллекции.

На листинге 4.3.2 представлена реализация описанных выше конечных точек.

Листинг 4.3.2 – Реализация конечных точек для сущности Photo

```
import {Router} from "express";  
import {PhotoModel} from "../models.js";  
import {v4} from "uuid";  
import {dirname} from "path";  
import mongoose from "mongoose";  
const UPLOAD_DIR = "C:\\Users\\Park  
Sergey\\Documents\\mirea\\Projects\\dream-gallery\\public\\upload";  
const photoRouter = Router();  
  
photoRouter.post("/", async (request, response) => {  
  console.log(`POST /api/v1/photos/`);  
  console.log(request.body);  
  
  let {  
    title,  
    description,  
    creatorId,  
    categoryId,  
  } = request.body;  
  /*===== ДОБАВЛЕНИЕ ФОТОГРАФИИ В БАЗУ ДАННЫХ =====*/  
  const fileName = `${v4()}${dirname(image.name)}`;   
  const uploadPath = `${UPLOAD_DIR}\\${fileName}`;  
  
  await image.mv(uploadPath, error => {  
    if (error) {  
      response.status(500);  
      response.json({  
        errors: [error.message]  
      });  
    }  
  });  
  const photo = new PhotoModel({
```

Продолжение листинга 4.3.2

```
    title: title,
    description: description,
    creatorId: mongoose.Types.ObjectId.createFromHexString(creatorId),
    categoryId: mongoose.Types.ObjectId.createFromHexString(categoryId),
    filename: fileName,
    path: uploadPath,
    published: false,
    createdAt: new Date(),
    updatedAt: new Date(),
  });

  PhotoModel
    .create(photo)
    .then(result => {
      response.status(201);
      response.json(result);
    });
});

photoRouter.get("/", (request, response) => {
  console.log(`GET /api/v1/photos/`);

  let filter = {};

  // получение фотографий, добавленных указанным пользователем
  if (request.query.creatorId) {
    filter.creatorId =
mongoose.Types.ObjectId.createFromHexString(request.query.creatorId);
  }

  // получение фотографий из указанной категории
  if (request.query.categoryId) {
    filter.categoryId =
mongoose.Types.ObjectId.createFromHexString(request.query.categoryId);
  }
  PhotoModel
    .aggregate([
      {
        $match: filter,
      },
      {
        $lookup: {
          from: "users",
          localField: "creatorId",
          foreignField: "_id",
          as: "creator",
        }
      },
      {
        $lookup: {
          from: "categories",
          localField: "categoryId",
          foreignField: "_id",
          as: "category"
        }
      },
    ])
    .then(result => {
      response.status(200);
      response.json(result);
    });
});
```

Продолжение листинга 4.3.2

```
});
photoRouter.get("/:id", (request, response) => {
  const photoId = request.params.id;
  console.log(`GET /api/v1/photos/${photoId}`);

  PhotoModel
    .aggregate([
      {
        $match: { _id:
mongoose.Types.ObjectId.createFromHexString(photoId) },
      },
      {
        $lookup: {
          from: "users",
          localField: "creatorId",
          foreignField: "_id",
          as: "creator",
        }
      },
      {
        $lookup: {
          from: "categories",
          localField: "categoryId",
          foreignField: "_id",
          as: "category",
        }
      },
    ])
    .then(result => {
      response.status(200);
      response.json(result[0]);
    });
});

photoRouter.put("/:id", async (request, response) => {
  const photoId = request.params.id;
  console.log(`PUT /api/v1/photos/${photoId}`);
  console.log(request.body);
  console.log(request.file);
  request.body.updatedAt = new Date();
  if (request.files) {
    const image = request.files.filename;
    const fileName = `${v4()}${extname(image.name)}`;
    const uploadPath = `${UPLOAD_DIR}\\${fileName}`;

    request.body.filename = fileName;
    request.body.path = uploadPath;

    await image.mv(uploadPath, error => {
      if (error) {
        response.status(500);

        response.json({
          errors: [error.message]
        });
      }
    });
  }
});

PhotoModel
  .findByIdAndUpdate(photoId, request.body)
  .then(result => {
```

Продолжение листинга 4.3.2

```
        response.status(200);
        response.json(result);
    });
});
photoRouter.delete("/:id", (request, response) => {
    const photoId = request.params.id;
    console.log(`DELETE /api/v1/photos/${photoId}`);
    PhotoModel
        .deleteOne({_id: photoId})
        .then(result => {
            response.status(200);
            response.json(result);
        });
});
export default photoRouter;
```

Для сущности Comment, описывающей комментарий, необходимо создать следующие конечные точки для:

1. Добавления комментария в коллекцию;
2. Получения списка комментариев;
3. Получения информации о конкретном комментарии.

На листинге 4.3.3 представлена реализация описанных выше конечных точек.

Листинг 4.3.3 – Реализация конечных точек для сущности Comment

```
import {Router} from "express";
import {CommentModel} from "../models.js";
import mongoose from "mongoose";

const commentRouter = Router();

/**
 * POST /api/v1/comments/
 * Добавление комментария в базу данных
 */
commentRouter.post("/", async (request, response) => {
    console.log(`POST /api/v1/comments/`);
    console.log(request.body);

    let {
        creatorId,
        photoId,
        content
    } = request.body;
    /*===== ДОБАВЛЕНИЕ КОММЕНТАРИЯ В БАЗУ ДАННЫХ =====*/
    const comment = new CommentModel({
        creatorId: mongoose.Types.ObjectId.createFromHexString(creatorId),
        photoId: mongoose.Types.ObjectId.createFromHexString(photoId),
        content: content,
        createdAt: new Date(),
    });
});
```

Продолжение листинга 4.3.3

```
CommentModel
  .create(comment)
  .then(result => {
    response.status(201);
    response.json(result);
  });
});

commentRouter.get("/", (request, response) => {
  console.log(`GET /api/v1/comments/`);

  let filter = {};

  // получение комментариев от указанного пользователя
  if (request.query.creatorId) {
    filter.creatorId =
mongoose.Types.ObjectId.createFromHexString(request.query.creatorId);
  }

  // получение комментариев к указанной фотографии
  if (request.query.photoId) {
    filter.photoId =
mongoose.Types.ObjectId.createFromHexString(request.query.photoId);
  }
  CommentModel
    .aggregate([
      {
        $match: filter
      },
      {
        $lookup: {
          from: "users",
          localField: "creatorId",
          foreignField: "_id",
          as: "creator",
        }
      },
      {
        $lookup: {
          from: "photos",
          localField: "photoId",
          foreignField: "_id",
          as: "photo",
        }
      },
    ])
    .then(result => {
      response.status(200);
      response.json(result);
    });
});

commentRouter.get("/:id", (request, response) => {
  const commentId = request.params.id;
  console.log(`GET /api/v1/comments/${commentId}`);

  CommentModel
    .aggregate([
      {
        $match: {_id:
mongoose.Types.ObjectId.createFromHexString(commentId) }
      },

```

Продолжение листинга 4.3.3

```
        {
          $lookup: {
            from: "users",
            localField: "creatorId",
            foreignField: "_id",
            as: "creator",
          }
        },
        {
          $lookup: {
            from: "photos",
            localField: "photoId",
            foreignField: "_id",
            as: "photo",
          }
        },
      ],
    })
    .then(result => {
      response.status(200);
      response.json(result[0]);
    });
  });
export default commentRouter;
```

Для сущности Album, описывающей альбом, необходимо создать следующие конечные точки для:

1. Добавления альбома в коллекцию;
2. Получения списка альбомов;
3. Получения информации о конкретном альбоме;
4. Обновление данных альбома;
5. Удаление альбома.

На листинге 4.3.4 представлена реализация описанных выше конечных точек.

Листинг 4.3.4 – Реализация конечных точек для сущности Album

```
import {Router} from "express";
import {AlbumModel} from "../models.js";
import mongoose from "mongoose";

const albumRouter = Router();
albumRouter.post("/", async (request, response) => {
  console.log("POST /api/v1/albums/");
  console.log(request.body);

  let {
    title,
    creatorId,
  } = request.body;
```

Продолжение листинга 4.3.4

```
let photos = request.body["photos[]"];

/*===== ДОБАВЛЕНИЕ АЛЬБОМА =====*/
const album = new AlbumModel({
  title: title,
  photos: photos,
  creatorId: creatorId,
  createdAt: new Date(),
  updatedAt: new Date(),
});

AlbumModel
  .create(album)
  .then(result => {
    response.status(201);
    response.json(result);
  });
});

albumRouter.get("/", (request, response) => {
  console.log("GET /api/v1/albums/");
  let filter = {};

  // получение альбомов указанного пользователя
  if (request.query.creatorId) {
    filter.creatorId =
      mongoose.Types.ObjectId.createFromHexString(request.query.creatorId);
  }

  AlbumModel
    .aggregate([
      {
        $match: filter
      },
      {
        $lookup: {
          from: "users",
          localField: "creatorId",
          foreignField: "_id",
          as: "creator",
        }
      },
      {
        $lookup: {
          from: "photos",
          localField: "photos",
          foreignField: "_id",
          as: "p",
        }
      },
    ])
    .then(result => {
      response.status(200);
      response.json(result);
    });
});

albumRouter.get("/:id", (request, response) => {
  const albumId = request.params.id;
  console.log(`GET /api/v1/albums/${albumId}`);
  AlbumModel
    .aggregate([
```

Продолжение листинга 4.3.4

```
    {
      $match: { _id:
mongoose.Types.ObjectId.createFromHexString(albumId) }
    },
    {
      $lookup: {
        from: "users",
        localField: "creatorId",
        foreignField: "_id",
        as: "creator",
      }
    },
    {
      $lookup: {
        from: "photos",
        localField: "photos",
        foreignField: "_id",
        as: "p",
      }
    },
  ])
  .then(result => {
    response.status(200);
    response.json(result[0]);
  });
});

albumRouter.put("/:id", (request, response) => {
  const albumId = request.params.id;
  console.log(`PUT /api/v1/categories/${albumId}`);
  console.log(request.body);

  if (!Array.isArray(request.body["photos[]"])) {
    request.body["photos[]"] = [request.body["photos[]"]];
  }

  request.body = {
    title: request.body.title,
    creatorId: request.body.creatorId,
    photos: request.body["photos[]"],
  };

  request.body.updatedAt = new Date();
  AlbumModel
    .findByIdAndUpdate(albumId, request.body)
    .then(result => {
      response.status(200);
      response.json(result);
    });
});

albumRouter.delete("/:id", (request, response) => {
  const albumId = request.params.id;
  console.log(`DELETE /api/v1/categories/${albumId}`);
  AlbumModel
    .deleteOne({ _id: albumId })
    .then(result => {
      response.status(200);
      response.json(result);
    });
});
export default albumRouter;
```


4.3 Реализация аутентификации в приложении

После ввода пароля пользователем при успешной проверке данных на клиентской стороне создаётся куки, содержащая идентификатор пользователя в базе данных. Эта куки сохраняется до завершения сеанса пользователем, то есть до закрытия окна браузера.

4.4 Выполнение версионного контроля

Из-за большого объёма кода, написанного при разработке веб-приложения, прилагается ссылка на репозиторий GitHub с исходным кодом проекта: <https://github.com/Sergey-hub02/dream-gallery>.

4.5 Выводы к разделу 4

В данном разделе были описаны зависимости, необходимые для разработки веб-приложения, сущности и коллекции, разработанные для взаимодействия с базой данных, процесс аутентификации и выполнение версионного контроля серверной части приложения.

5 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ

5.1 Создание проекта и добавление зависимостей

Для создания проекта необходимо использовать утилиту `create-react-app`, которое создаёт React проекты с установленными для него зависимостями.

После создания проекта необходимо установить дополнительные зависимости:

1. `Axios` для удобного выполнения запросов к серверной части;
2. `Bootstrap` для ускорения разработки клиентской части;
3. `Universal Cookie` для реализации процесса авторизации;
4. `React-router-dom` для реализации межстраничной навигации.

На листинге 5.1.1 представлен код файла `package.json`.

Листинг 5.1.1 – Файл `package.json`

```
{
  "name": "dream-gallery",
  "version": "0.1.0",
  "dependencies": {
    "axios": "^1.4.0",
    "bootstrap": "^5.2.3",
    "react": "^18.2.0",
    "react-bootstrap": "^2.7.4",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.11.1",
    "react-scripts": "5.0.1",
    "universal-cookie": "^4.0.4",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start front": "react-scripts start",
    "start back": "node api/v1/src/index.js",
    "build": "react-scripts build"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ]
  },
}
```

5.2 Создание компонентов и страниц

После создания проекта и установки всех нужных зависимостей необходимо создать компоненты, которые в дальнейшем будут использоваться для создания страниц. На рисунке 5.2.1 представлены созданные компоненты.

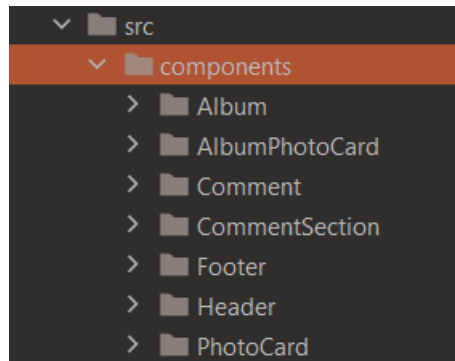


Рисунок 5.2.1 – Компоненты

После создания компонентов нужно создать страницы (Рисунок 5.2.2).

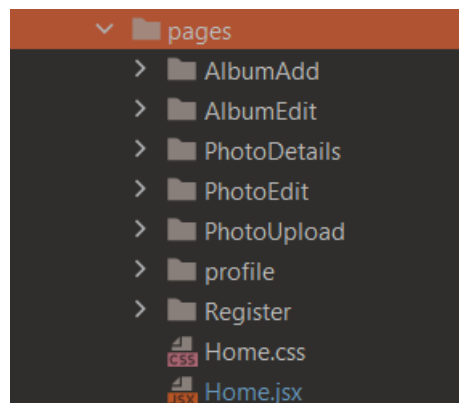


Рисунок 5.2.2 – Страницы

5.3 Реализация навигации по страницам

Для реализации межстраничной навигации необходимо использовать компоненты `BrowserRouter`, `Routes`, `Route` из библиотеки `react-router-dom`. Для задания маршрута в компоненте `Route` нужно указать путь к странице и компонент, который будет отображаться при переходе по указанному пути.

Все маршруты указаны в файле `App.jsx`, содержимое которого представлено на листинге 5.3.1.

Листинг 5.3.1 – Файл App.jsx

```
import React from "react";
import {Routes, Route} from "react-router-dom";
import {Home} from "../pages/Home";
import {Register} from "../pages/Register/Register";
import {Login} from "../pages/Register/Login";
import {UserProfile} from "../pages/profile/UserProfile";
import {PhotoUpload} from "../pages/PhotoUpload/PhotoUpload";
import {PhotoDetails} from "../pages/PhotoDetails/PhotoDetails";
import {PhotoEdit} from "../pages/PhotoEdit/PhotoEdit";
import {AlbumAdd} from "../pages/AlbumAdd/AlbumAdd";
import "../App.css";
import {AlbumEdit} from "../pages/AlbumEdit/AlbumEdit";
/**
 * Определяет маршруты на сайте
 * @returns {JSX.Element}
 * @constructor
 */
function App() {
  return (
    <div className="App">
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/register" element={<Register />} />
        <Route path="/login" element={<Login />} />
        <Route path="/profile" element={<UserProfile />} />

        <Route path="/photos/upload" element={<PhotoUpload />} />
        <Route path="/photos/:photoId" element={<PhotoDetails />} />
        <Route path="/photos/edit/:photoId" element={<PhotoEdit />} />

        <Route path="/albums/add" element={<AlbumAdd />} />
        <Route path="/albums/edit/:albumId" element={<AlbumEdit />} />
      </Routes>
    </div>
  );
}
export default App;
```

5.4 Отображение созданных страниц

Пользователь при первом посещении попадает на главную страницу (Рисунок 5.4.1). На главной странице отображаются фотографии из каждой категории.

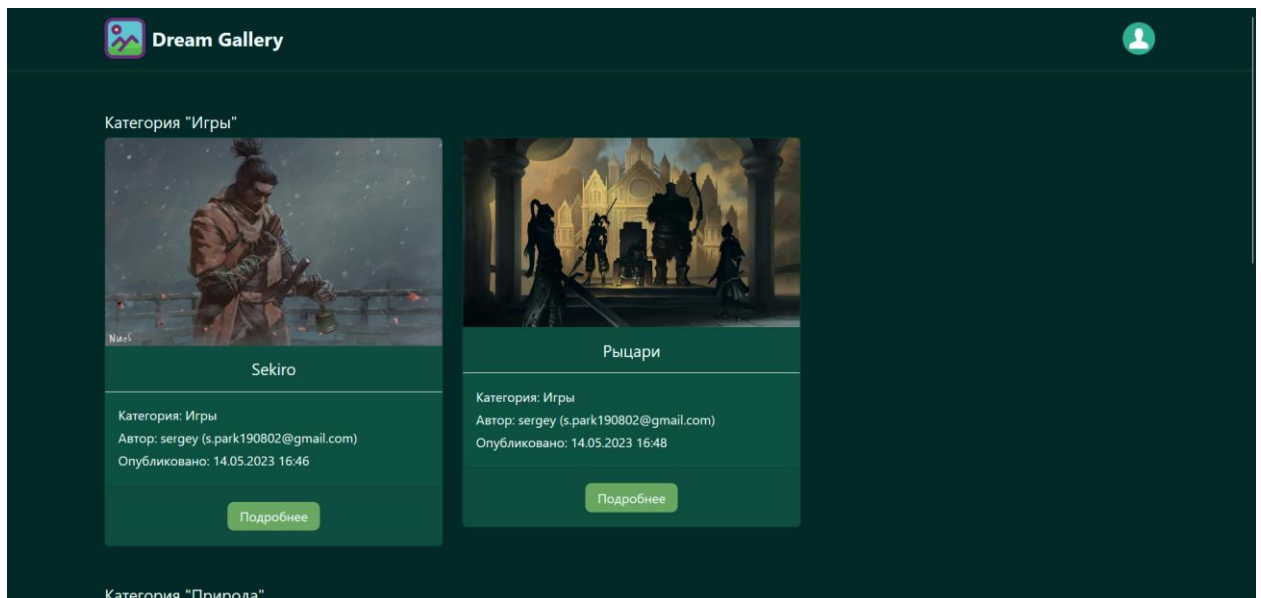


Рисунок 5.4.1 – Главная страница

При нажатии на карточку фотографии отображается страница деталей фотографии (Рисунок 5.4.2). На этой странице отображается сама фотография, её данные, описание, комментарии.

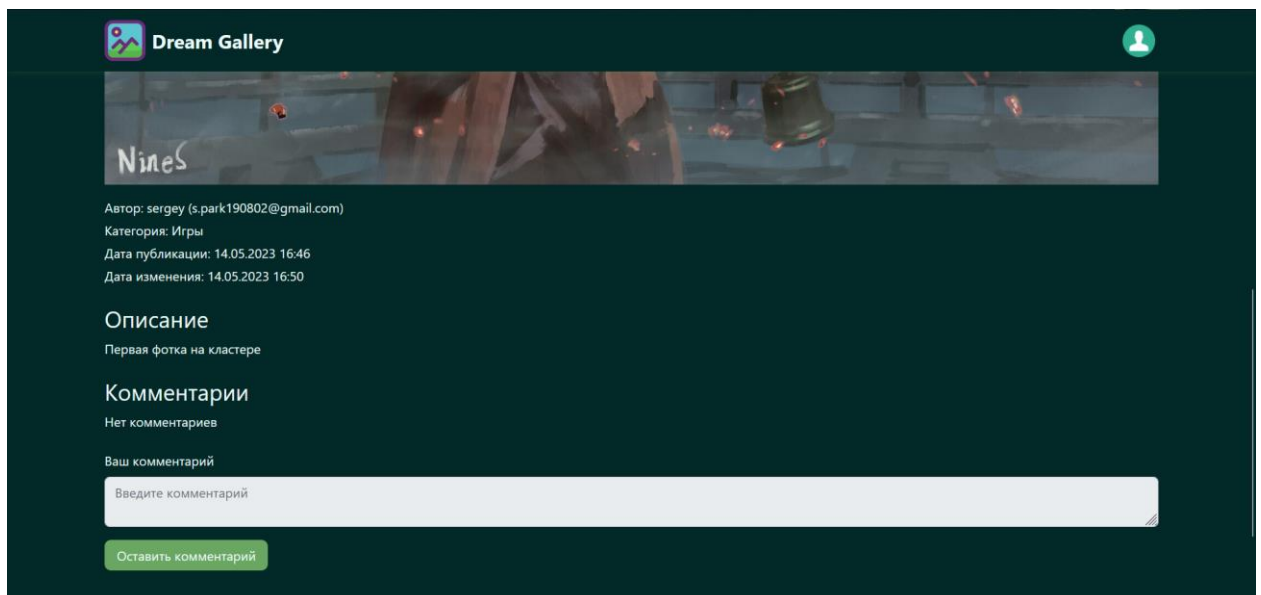
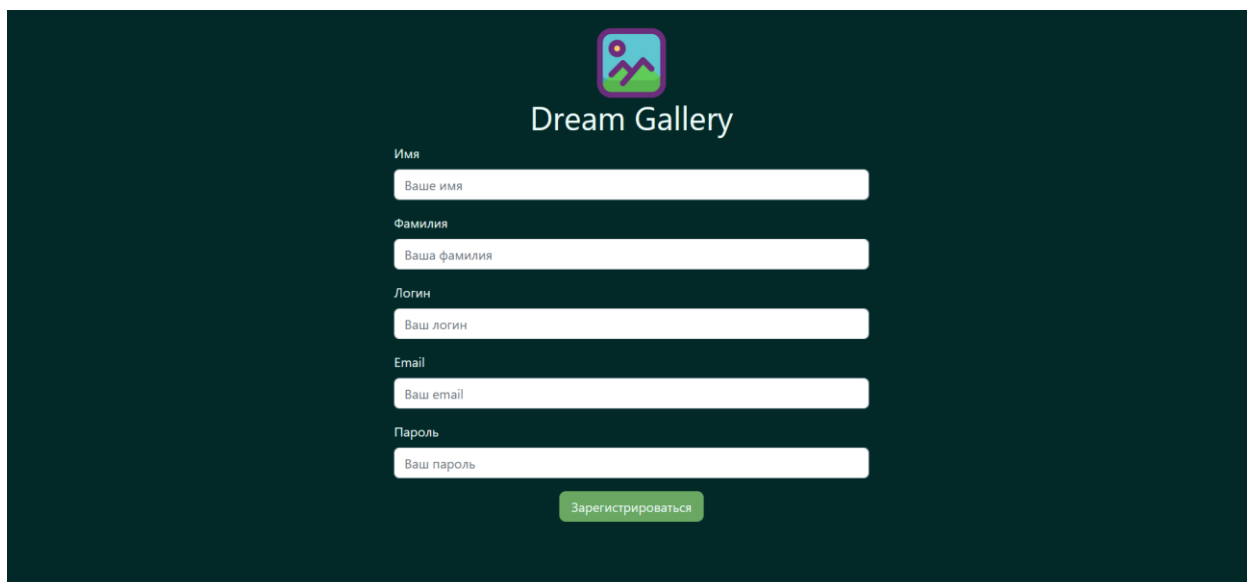


Рисунок 5.4.2 – Страница фотографии

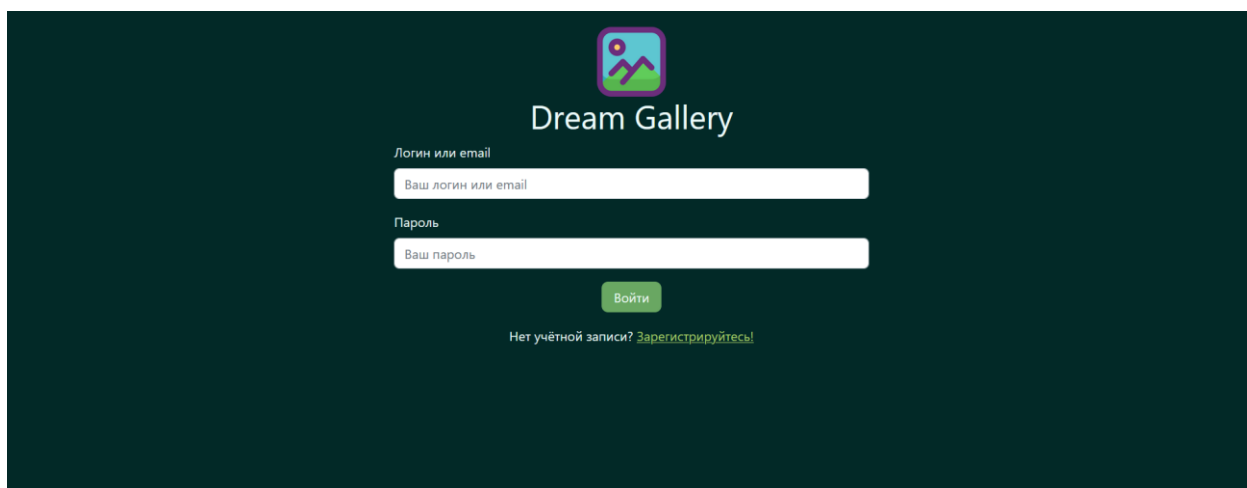
Пользователь может зарегистрироваться. Для этого ему необходимо перейти на страницу регистрации (Рисунок 5.4.3).



The registration form for Dream Gallery is centered on a dark green background. At the top is a logo consisting of a purple square with a white circle and a green mountain silhouette. Below the logo is the text "Dream Gallery" in white. The form consists of six white input fields with labels in Russian: "Имя" (Name), "Фамилия" (Surname), "Логин" (Login), "Email", and "Пароль" (Password). Each field contains a placeholder text: "Ваше имя", "Ваша фамилия", "Ваш логин", "Ваш email", and "Ваш пароль" respectively. Below the password field is a green button with the text "Зарегистрироваться" (Register).

Рисунок 5.4.3 – Страница регистрации

При наличии учётной записи пользователь имеет возможность авторизоваться (Рисунок 5.4.4).



The authorization form for Dream Gallery is centered on a dark green background. At the top is the same logo as in the registration form. Below the logo is the text "Dream Gallery" in white. The form consists of two white input fields with labels in Russian: "Логин или email" (Login or email) and "Пароль" (Password). Each field contains a placeholder text: "Ваш логин или email" and "Ваш пароль" respectively. Below the password field is a green button with the text "Войти" (Login). Below the button is a link in green text: "Нет учётной записи? [Зарегистрируйтесь!](#)".

Рисунок 5.4.4 – Страница авторизации

При авторизации становится доступной страница личного кабинета пользователя (Рисунок 5.4.5). Здесь пользователь может редактировать свои данные, а также посмотреть загруженные им фотографии.

Dream Gallery petrovich

Пётр Петров
petrovich (petrov@mail.com)
Регистрация: 14.05.2023 17:40
Изменено: 14.05.2023 17:40

Имя
Пётр

Фамилия
Петров

Логин
petrovich

Email
petrov@mail.com

Новый пароль
Ваш новый пароль

Сохранить

Мои фотографии ([Добавить](#))
Вы, пока, не загрузили ни одной фотографии!

Рисунок 5.4.5 – Личный кабинет пользователя

На рисунке 5.4.6 представлена страница добавления фотографии.

Dream Gallery petrovich

Загрузка фотографии

Название
Название фотографии

Описание
Описание фотографии

Категория
Игры

Файл
Обзор... Файл не выбран.

Загрузить

Рисунок 5.4.6 – Страница добавления фотографии

На рисунке 5.4.7 представлена страница редактирования данных фотографии. На этой странице также можно удалить фотографию.

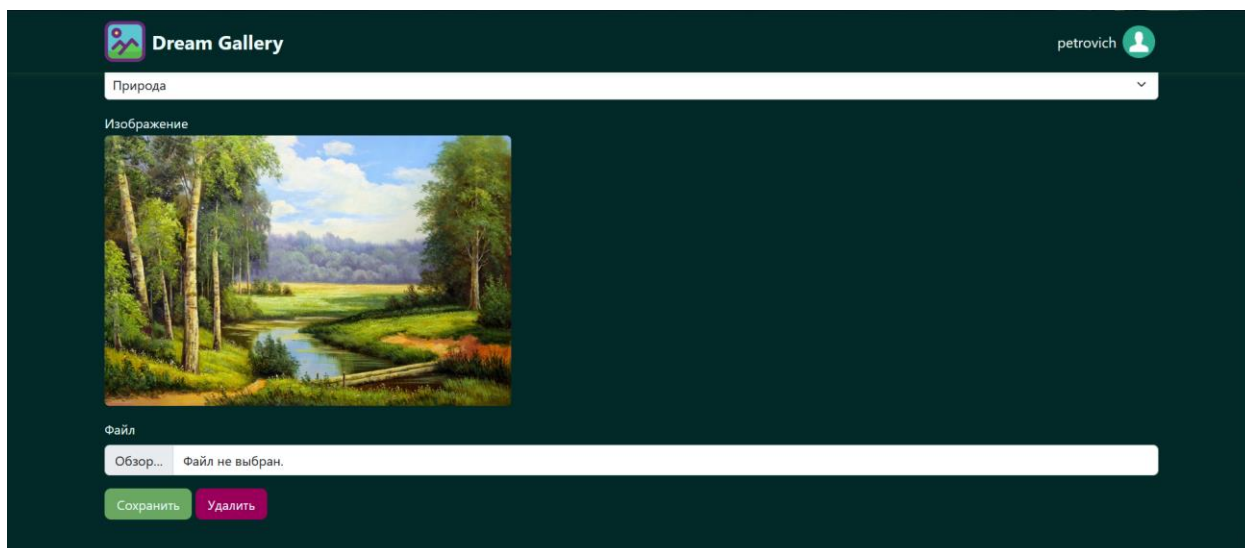


Рисунок 5.4.7 – Страница редактирования данных фотографии

На рисунке 5.4.8 представлена страница создания альбома.

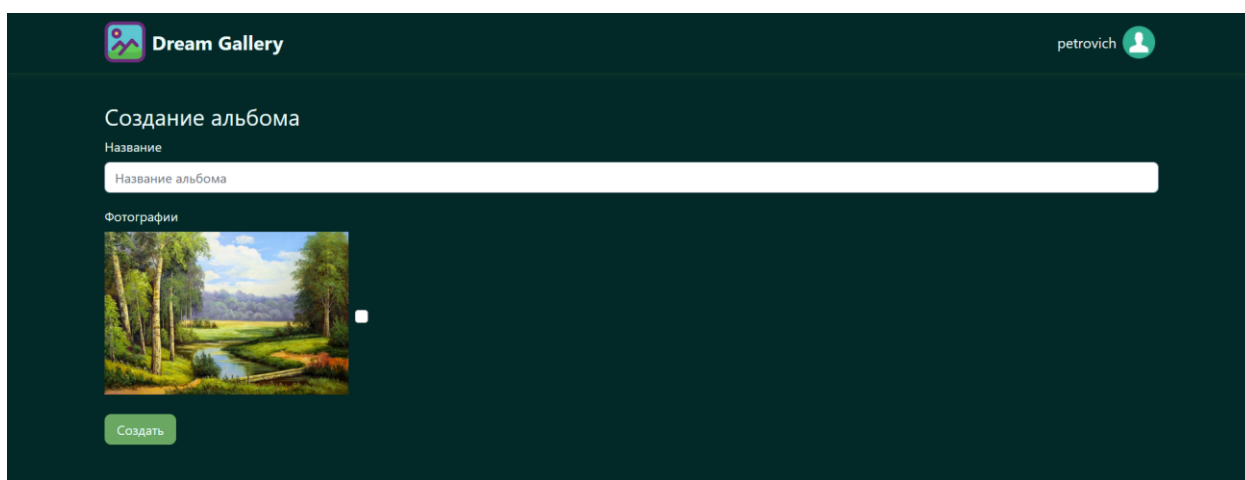


Рисунок 5.4.8 – Страница создания альбома

На рисунке 5.4.9 представлена страница изменения данных альбома.

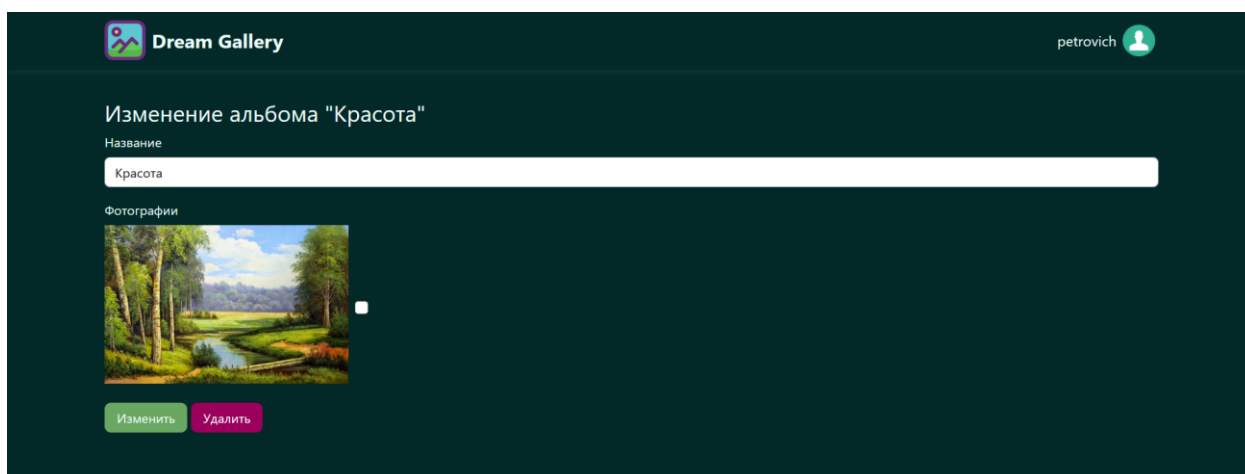


Рисунок 5.4.9 – Страница изменения альбома

5.5 Выполнение версионного контроля

Из-за большого объёма кода, написанного при разработке веб-приложения, прилагается ссылка на репозиторий GitHub с исходным кодом проекта: <https://github.com/Sergey-hub02/dream-gallery>.

5.6 Выводы к разделу 5

В ходе разработки клиентской части приложения был создан проект и установлены необходимые для разработки зависимости. С помощью разработанных компонентов было создано 9 страниц.

6 РАЗВЁРТЫВАНИЕ В ОБЛАЧНОМ ХРАНИЛИЩЕ

6.1 Развёртывание проекта

Развёртывание проекта будет проходить с помощью облачного хранилища Render.

Для развёртывания необходимо зарегистрироваться, войти в учётную запись, указать ссылку на репозиторий GitHub проекта.

На рисунке 6.1.1 представлена панель развёрнутых приложений.

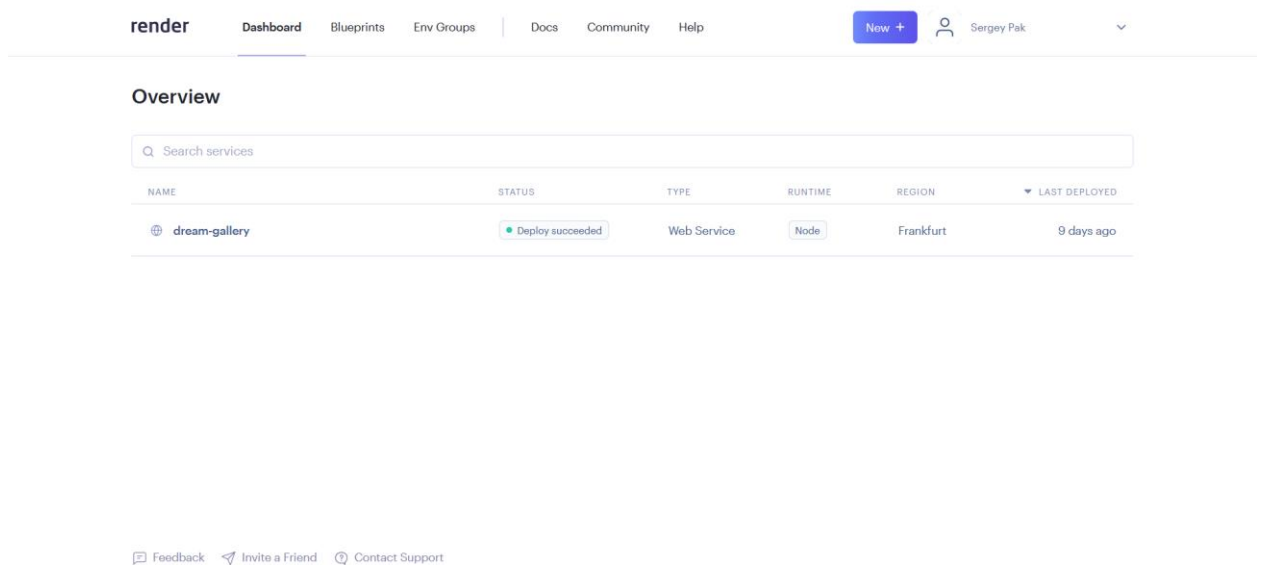


Рисунок 6.1.1 – Список развёрнутых приложений

После этого веб-приложение станет доступно по адресу <https://dream-gallery.onrender.com>. При переходе по ссылке отобразится главная страница сайта (Рисунок 6.1.2).

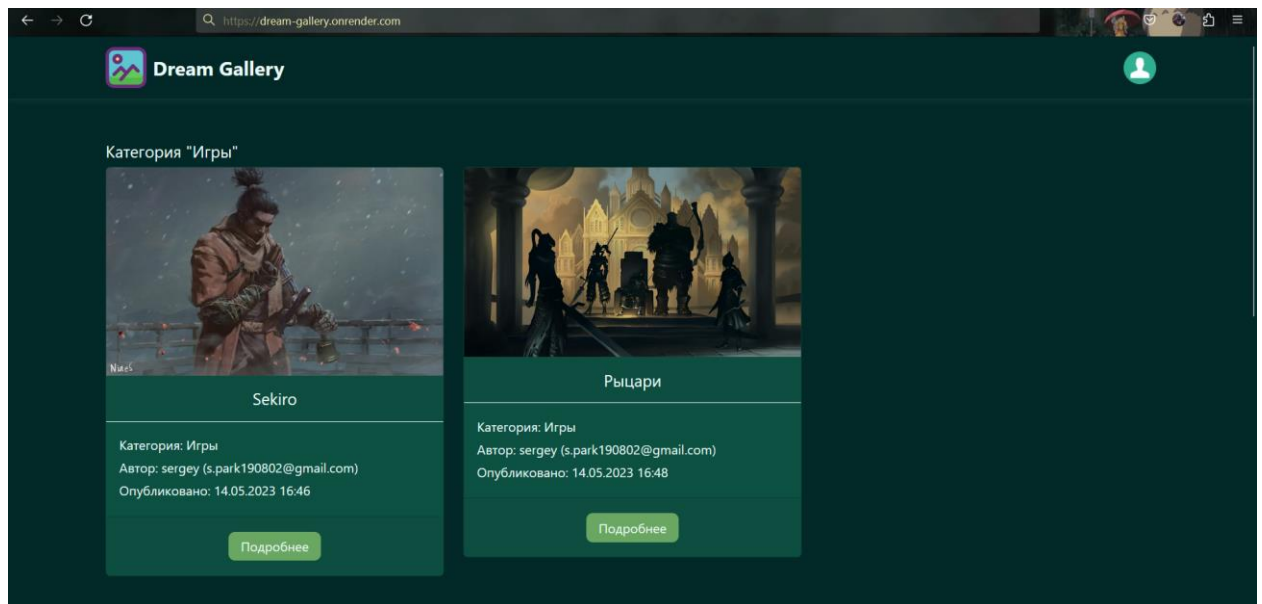


Рисунок 6.1.2 – Главная страница после развёртывания

6.2 Выводы к разделу 6

В данном разделе с помощью облачной платформы Render было развёрнуто разработанное веб-приложение.

7 ПРОВЕРКА ФУНКЦИОНИРОВАНИЯ ПРОГРАММНОГО ПРОДУКТА

7.1 Проверка работоспособности серверной части

Тестирование будет проходить с помощью с фреймворка Mocha. Для тестирования был создан файл `test.js`, в котором тестируется каждая разработанная конечная точка (Рисунок 7.1.1).

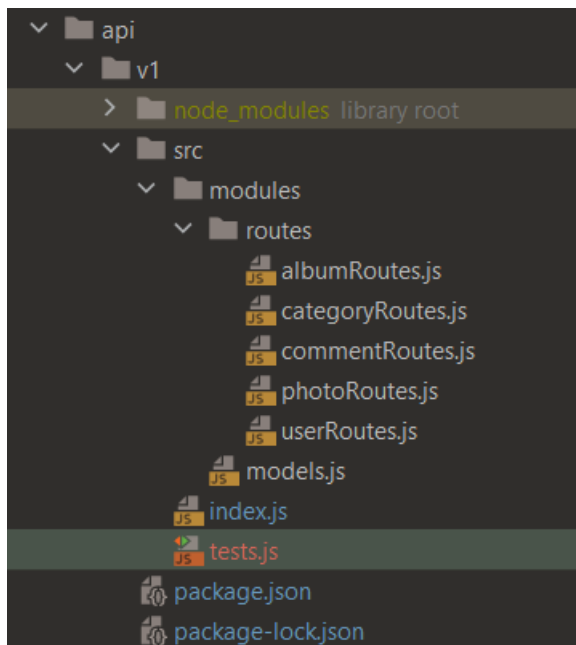


Рисунок 7.1.1 – Расположение файла `test.js`

На рисунках 7.1.2 – 7.1.5 представлены результаты тестирования каждой конечной точки.

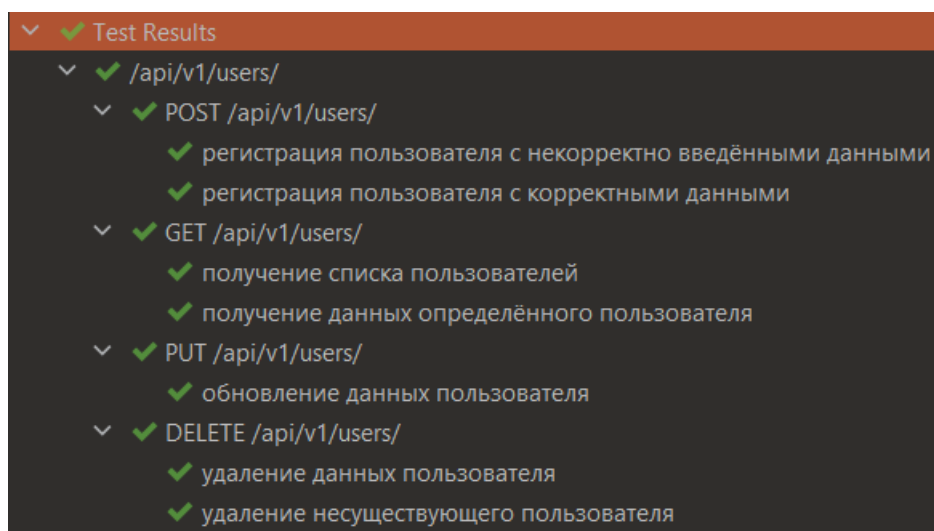


Рисунок 7.1.2 – Тестирование конечных точек для сущности User

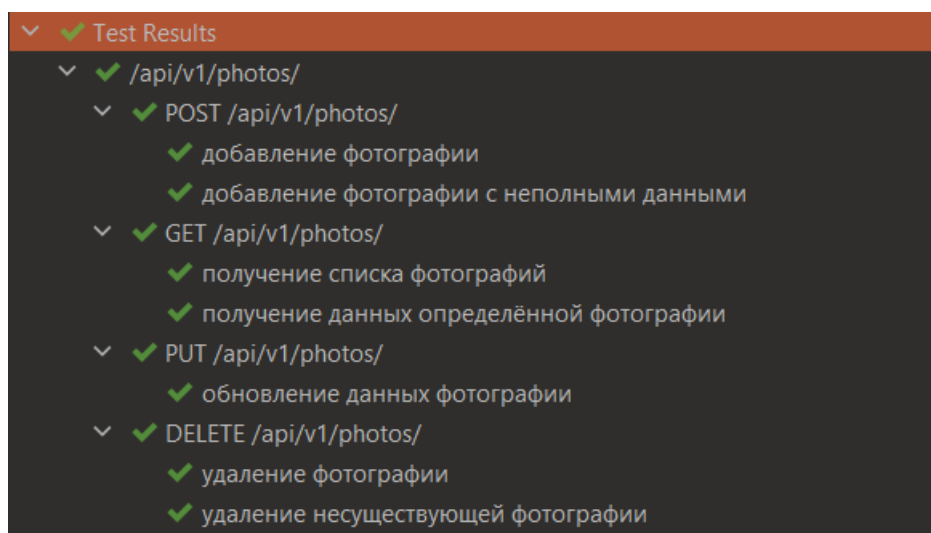


Рисунок 7.1.3 – Тестирование конечных точек для сущности Photo

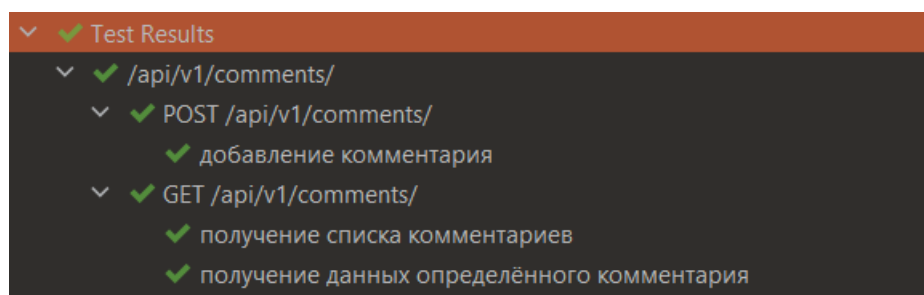


Рисунок 7.1.4 – Тестирование конечных точек для сущности Comment

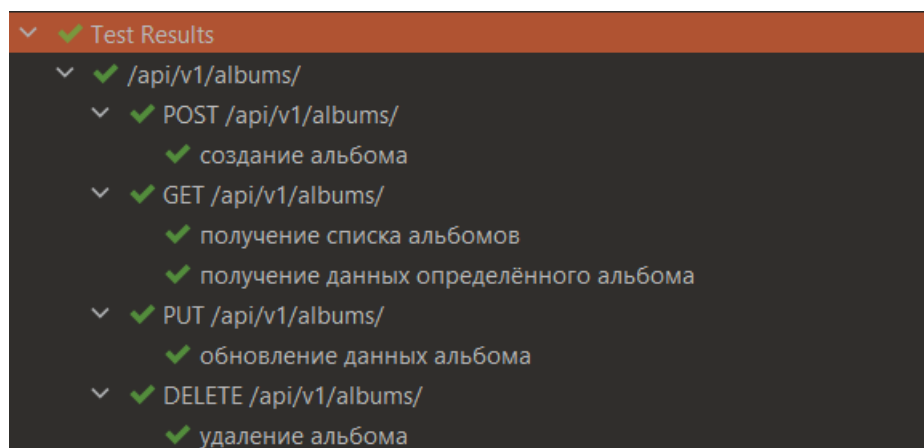
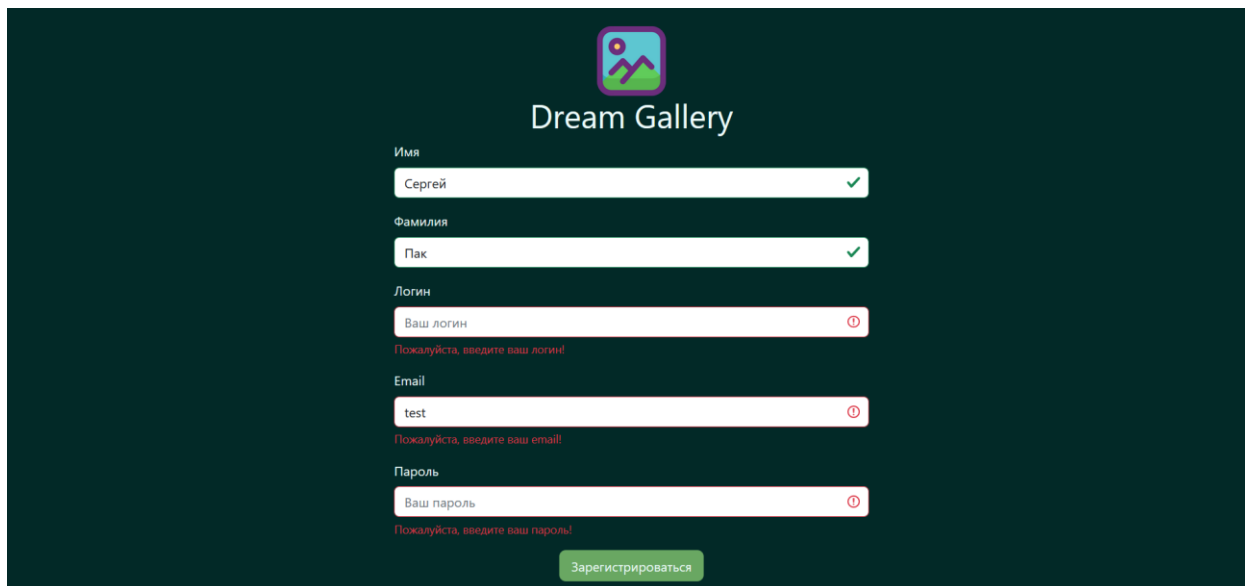


Рисунок 7.1.5 – Тестирование конечных точек для сущности Album

7.2 Проверка работоспособности клиентской части

Для проверки работоспособности клиентской части воспользуемся методом «чёрного ящика». Необходимо проверить отзывчивость интерфейса

при изменяющихся условиях. Результаты тестирования представлены на рисунках 7.2.1 – 7.2.6.



The screenshot shows the registration page of 'Dream Gallery'. The form includes fields for Name, Surname, Login, Email, and Password. The 'Имя' (Name) and 'Фамилия' (Surname) fields are filled with 'Сергей' and 'Пак' respectively, both marked with green checkmarks. The 'Логин' (Login) field contains 'Ваш логин' and shows a red error icon with the message 'Пожалуйста, введите ваш логин!'. The 'Email' field contains 'test' and shows a red error icon with the message 'Пожалуйста, введите ваш email!'. The 'Пароль' (Password) field contains 'Ваш пароль' and shows a red error icon with the message 'Пожалуйста, введите ваш пароль!'. A green 'Зарегистрироваться' (Register) button is at the bottom.

Рисунок 7.2.1 – Введение некорректных данных при регистрации



The screenshot shows the photo upload page of 'Dream Gallery'. The form includes fields for Name, Description, Category, and File. The 'Название' (Name) field contains 'Название фотографии' and shows a red error icon with the message 'Пожалуйста, введите название фотографии!'. The 'Описание' (Description) field contains 'Описание фотографии' and is marked with a green checkmark. The 'Категория' (Category) field is set to 'Другое' and is marked with a green checkmark. The 'Файл' (File) field shows 'Обзор...' and 'Файл не выбран.' and shows a red error icon with the message 'Пожалуйста, выберите файл!'. A green 'Загрузить' (Upload) button is at the bottom.

Рисунок 7.2.2 – Введение некорректных данных при загрузке фотографии

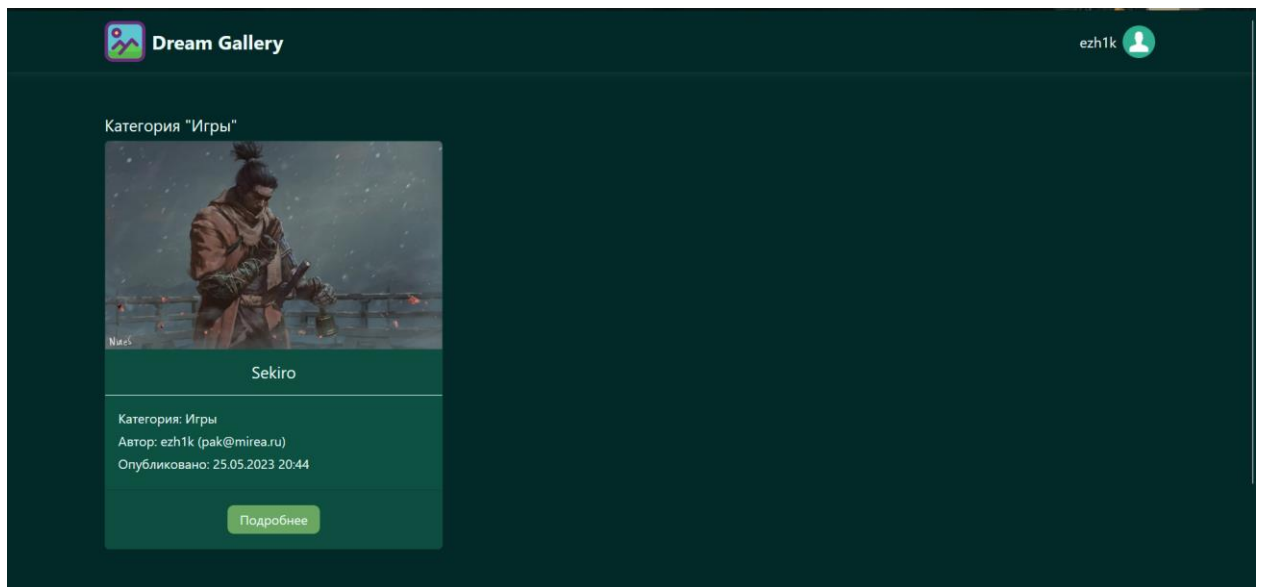


Рисунок 7.2.3 – Отображение загруженной фотографии

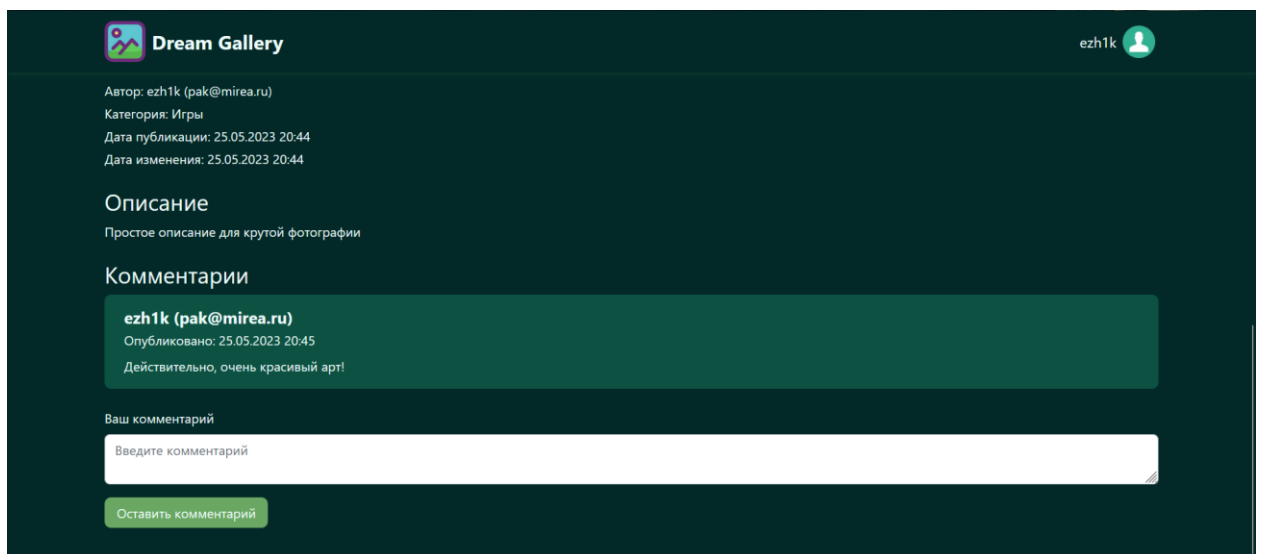


Рисунок 7.2.4 – Отображение комментария

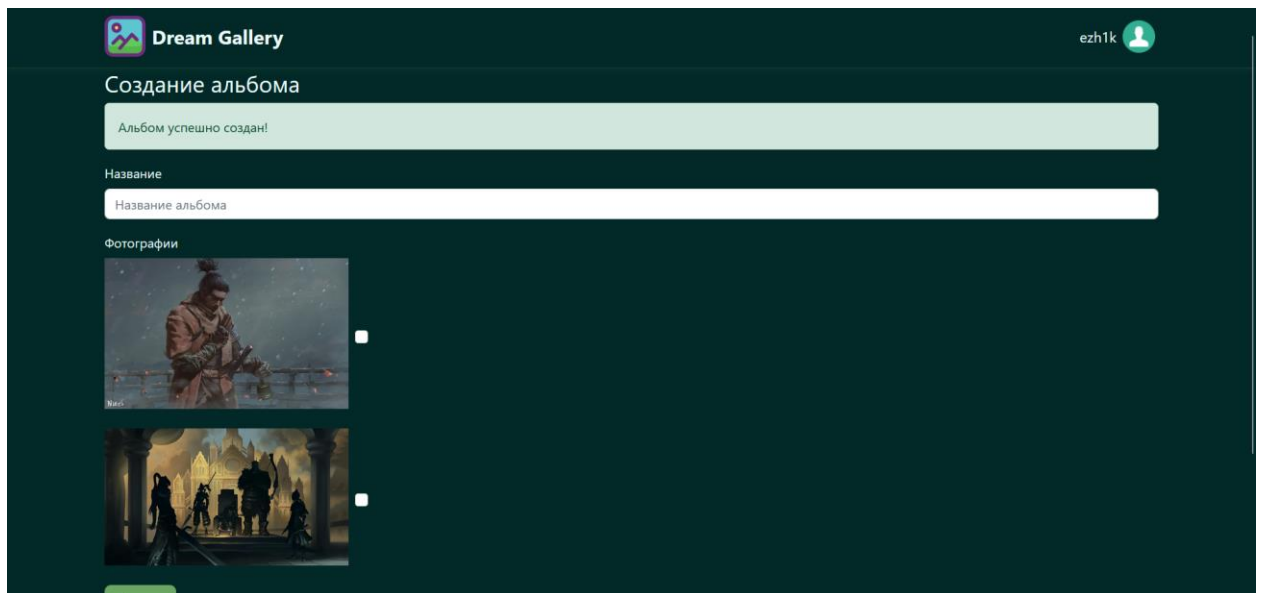


Рисунок 7.2.5 – Успешное создание альбома

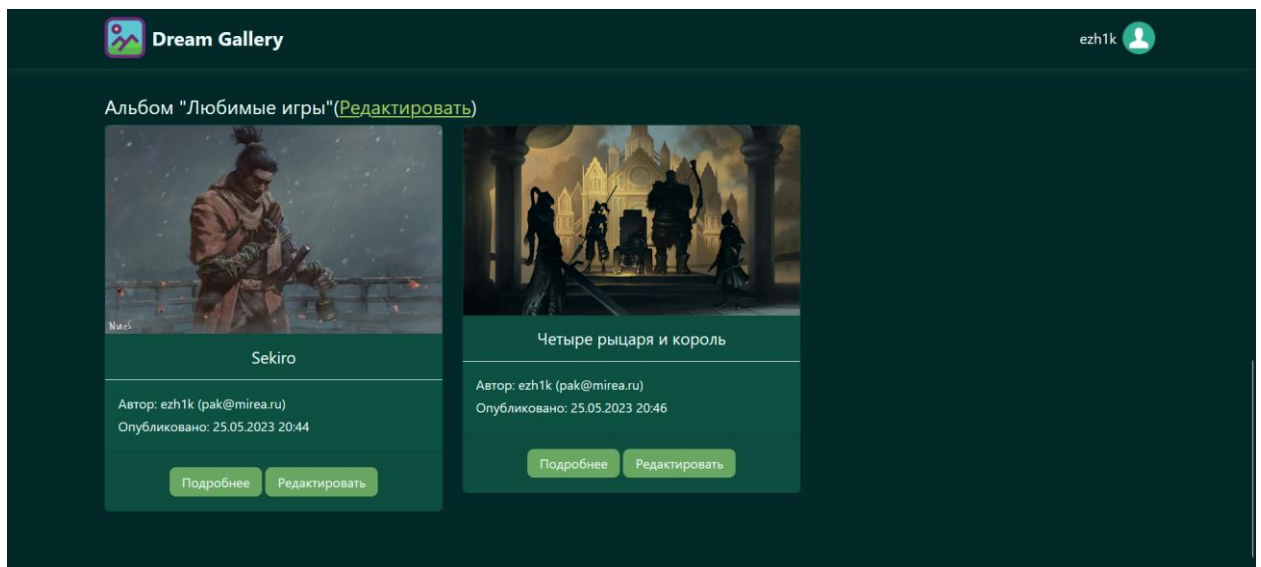


Рисунок 7.2.6 – Отображение альбома в личном кабинете

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы в соответствии с разработанными требованиями было создано клиент-серверное фуллстек веб-приложение для фотогалереи.

Для определения функционала веб-приложения была создана диаграмма вариантов использования, на которой также были отображены роли пользователей. С помощью диаграммы развёртывания была описана архитектура разрабатываемого приложения, на которой были отображены компоненты и то, как они взаимодействуют. Также с помощью логической схемы базы данных была описана её структура.

Для реализации серверной части веб-приложения были разработаны все необходимые конечные точки, взаимодействующие с базой данных.

Аутентификация была реализована с помощью технологии куки. При успешном верификации пользователя создаётся куки, содержащая идентификатор пользователя в базе данных.

Для реализации клиентской части были созданы 9 страниц (для регистрации, авторизации, загрузки фотографии, редактирования данных фотографии, просмотра фотографии, редактирования альбома, добавления альбома, личный кабинет, главная).

Для выполнения версионного контроля использовалась система версионного контроля Git. Код веб-приложения размещён в репозитории GitHub по ссылке <https://github.com/Sergey-hub02/dream-gallery>.

Приложение было развёрнуто на облачной платформе Render, и доступно по ссылке <https://dream-gallery.onrender.com/>.

Было проведено тестирование для серверной и клиентской частей веб-приложение. Тестирование показало, что все функции работают корректно.

В ходе выполнения курсовой работы были приобретены следующие компетенции:

1. ПК-1 – способностью демонстрации общенаучных базовых знаний естественных наук, математики и информатики, понимание основных фактов, концепций, принципов теорий, связанных с прикладной математикой и информатикой;
2. ПК-1.2 – способностью использования основного и вспомогательного оборудования и материала;
3. ПК-1.12 – способностью осуществлять наладку, настройку, регулировку и опытную проверку оборудования и систем в лабораторных условиях и на объектах;
4. ПК-1.16 – способностью осуществлять контроль качества выполняемых работ;
5. ПК-13 – готовностью к использованию методов и инструментальных средств исследования объектов профессиональной деятельности;
6. ПК-14 – способность создавать программные интерфейсы;
7. ПК-19 – понимание стандартов и моделей жизненного цикла;
8. ПК-20 – навыки проведения практических занятий с пользователями программных систем;
9. ОПК-9 – способность использовать навыки работы с компьютером, владеть методами информационных технологий, соблюдать основные требования информационной безопасности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Живопись, картины и художники – онлайн музей Gallerix // Gallerix URL: <https://gallerix.ru/> (дата обращения: 09.05.2023)
2. Главная – Art Online 24 – онлайн галерея современного искусства // Art Online 24 URL: <https://artonline24.ru/0> (дата обращения 09.05.2023)
3. Artmajeur – Художественная галерея №1 // Artmajeur URL: <https://www.artmajeur.com/ru/> (дата обращения 09.05.2023)
4. Клиент-сервер – Википедия // Википедия URL: https://ru.wikipedia.org/wiki/Клиент_-_сервер (дата обращения 09.05.2023)
5. React – JavaScript-библиотека для создания пользовательских интерфейсов // ReactJS URL: <https://ru.legacy.reactjs.org/> (дата обращения 09.05.2023)
6. Bootstrap The most popular HTML, CSS, and JS library in the world // Bootstrap URL: <https://getbootstrap.com/> (дата обращения 09.05.2023)
7. Умная среда разработки для JavaScript, созданная в JetBrains // WebStorm URL: <https://www.jetbrains.com/ru-ru/webstorm/> (дата обращения 09.05.2023)
8. Введение | Axios Docs // Axios URL: <https://axios-http.com/ru/docs/intro> (дата обращения 09.05.2023)
9. Mongoose (MongoDB) – Wikipedia // Wikipedia URL: [https://en.wikipedia.org/wiki/Mongoose_\(MongoDB\)](https://en.wikipedia.org/wiki/Mongoose_(MongoDB)) (дата обращения 09.05.2023)
10. Express – фреймворк веб-приложений Node.js // ExpressJS URL: <https://expressjs.com/ru/> (дата обращения 09.05.2023)
11. Победитель Password Hashing Competition Argon2 или ещё раз о медленном хэшировании // Хабр URL: <https://habr.com/ru/articles/281569/> (дата обращения 09.05.2023)
12. MongoDB – Википедия // Википедия URL: <https://ru.wikipedia.org/wiki/MongoDB> (дата обращения 09.05.2023)

13. Git // Git URL: <https://git-scm.com/> (дата обращения 09.05.2023)
14. GitHub – Википедия // Википедия URL:
<https://ru.wikipedia.org/wiki/GitHub> (дата обращения 09.05.2023)