МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Web-приложение «Конструктор баннеров»

Выполнил студент Селицкий Николай Евгеньевич

(Ф.И.О.)

Руководитель проекта к.т.н., доцент Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доцент Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2025

[Введение 4](#_Toc198509194)

[1. Постановка задачи и обзор аналогичных решений 5](#_Toc198509195)

[1.1 Постановка задачи 5](#_Toc198509196)

[1.2 Аналитический разбор аналогов 5](#_Toc198509197)

[1.2.1 Онлайн-редактор Canva 5](#_Toc198509198)

[1.2.2 Онлайн-редактор Crello (VistaCreate) 6](#_Toc198509199)

[1.3 Выводы по разделу 7](#_Toc198509200)

[2. Проектирование web-приложения 8](#_Toc198509201)

[2.1 Функциональность web-приложения 8](#_Toc198509202)

[2.2 Структура базы данных 11](#_Toc198509203)

[2.3 Архитектура web-приложения 13](#_Toc198509204)

[2.4 Выводы по разделу 15](#_Toc198509205)

[3. Реализация web-приложения 16](#_Toc198509206)

[3.1 Программная платформа Node.js 16](#_Toc198509207)

[3.2 NoSQL база данных MongoDB 16](#_Toc198509208)

[3.3 Средство моделирования объектного документа 16](#_Toc198509209)

[3.4 Программные библиотеки 21](#_Toc198509210)

[3.5 Структура серверной части 23](#_Toc198509211)

[3.6 Реализация функций для менеджера 25](#_Toc198509212)

[3.6.1 Регистрация 25](#_Toc198509213)

[3.6.2 Авторизация 25](#_Toc198509214)

[3.6.3 Просмотр ресурсов 26](#_Toc198509215)

[3.6.4 Создание ресурса 26](#_Toc198509216)

[3.6.5 Обновление ресурса 26](#_Toc198509217)

[3.6.6 Удаление ресурса 27](#_Toc198509218)

[3.6.7 Просмотр шаблонов 27](#_Toc198509219)

[3.6.8 Создание шаблона 28](#_Toc198509220)

[3.6.9 Обновление шаблона 28](#_Toc198509221)

[3.6.10 Удаление шаблона 29](#_Toc198509222)

[3.6.11 Просмотр страниц 29](#_Toc198509223)

[3.6.12 Создание страницы 30](#_Toc198509224)

[3.6.13 Обновление страницы 30](#_Toc198509225)

[3.6.14 Удаление страницы 31](#_Toc198509226)

[3.6.15 Просмотр сгенерированной страницы 31](#_Toc198509227)

[3.7 Структура клиентской части 31](#_Toc198509228)

[3.8 Выводы по разделу 33](#_Toc198509229)

[4. Тестирования web-приложения 34](#_Toc198509230)

[4.1 Функциональное тестирование 34](#_Toc198509231)

Введение

Web-приложение — это программное обеспечение с клиент-серверной архитектурой, где клиент и сервер взаимодействуют друг с другом по протоколу HTTP.

Web-приложение «Конструктор баннеров» представляет собой инструмент для создания визуальных рекламных и информационных баннеров в режиме онлайн. Оно позволяет пользователям разрабатывать креативные графические материалы с помощью встроенного редактора, используя готовые шаблоны и ресурсы.

Цели проекта:

1. Разработать удобный и интуитивно понятный интерфейс для создания и редактирования баннеров;
2. Обеспечить возможность создание шаблонов и добавления изображений, текста и скриптов;
3. Реализовать функцию генерации готовых баннеров и возможность просматривать их;
4. Предоставить пользователям возможность сохранять свои проекты и возвращаться к редактированию в любое время.

Для достижения поставленных целей были сформулированы следующие задачи:

1. провести анализ существующих онлайн-конструкторов баннеров, выявить их преимущества и недостатки (глава 1);
2. разработать архитектуру и структуру web-приложения, обосновать выбор технологий и спроектировать базу данных (глава 2);
3. реализовать ключевые функции приложения: интерфейс редактора, работа с графикой, управление шаблонами и проектами (глава 3);
4. провести тестирование web-приложения для проверки стабильности и корректности работы основных функций (глава 4);
5. подготовить руководство пользователя, описывающее процесс создания баннеров и использования возможностей системы (глава 5).

Целевая аудитория проекта включает малый и средний бизнес, специалистов в области маркетинга, дизайнеров и обычных пользователей, которым необходимо быстро и без навыков профессионального дизайна создать привлекательный визуальный контент.

В качестве фреймворков выбраны **Next.js** и **NestJS**. Next.js используется для создания быстрого и современного пользовательского интерфейса, а NestJS — для организации масштабируемой и структурированной серверной логики на базе Node.js.

1. Постановка задачи и обзор аналогичных решений
2. Постановка задачи

Web-приложение «Конструктор баннеров» предполагает наличие ролей: пользователь и администратор.

Пользователь должен иметь возможность зарегистрироваться и аутентифицироваться, создавать, изменять и удалять шаблоны, ресурсы и баннеры.

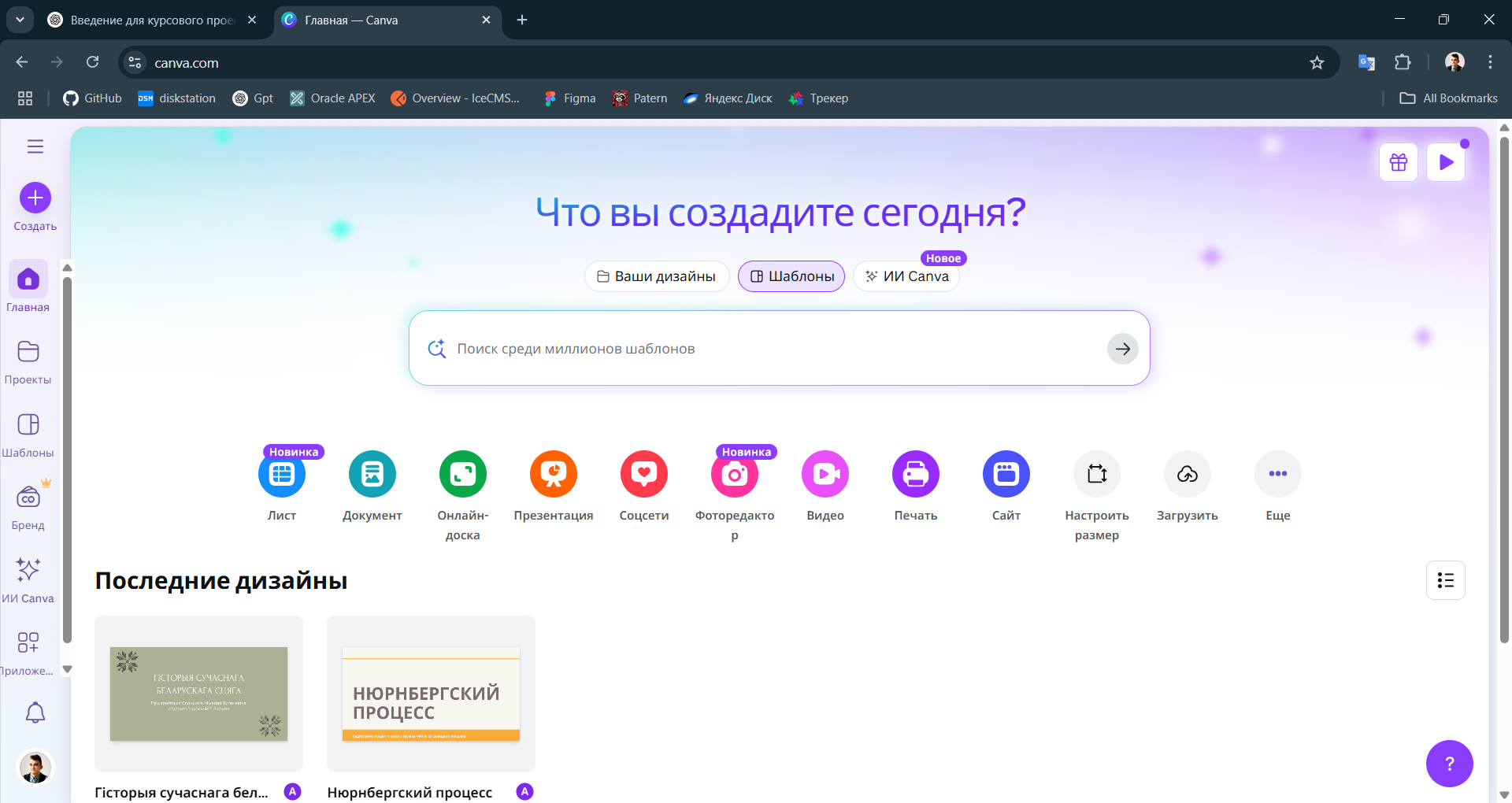
Администратор должен иметь все возможности пользователя и доступ к странице с пользователями, где он может: давать другим пользователям права администратора или изымать их, а также блокировать или разблокировать других пользователей.

1. Аналитический разбор аналогов

В данном разделе рассмотрены существующие web-приложения для онлайн-дизайна и создания баннеров.

1. Онлайн-редактор Canva

На рисунке 1.1 представлен интерфейс платформы Canva — популярного онлайн-сервиса для создания графического контента [1].



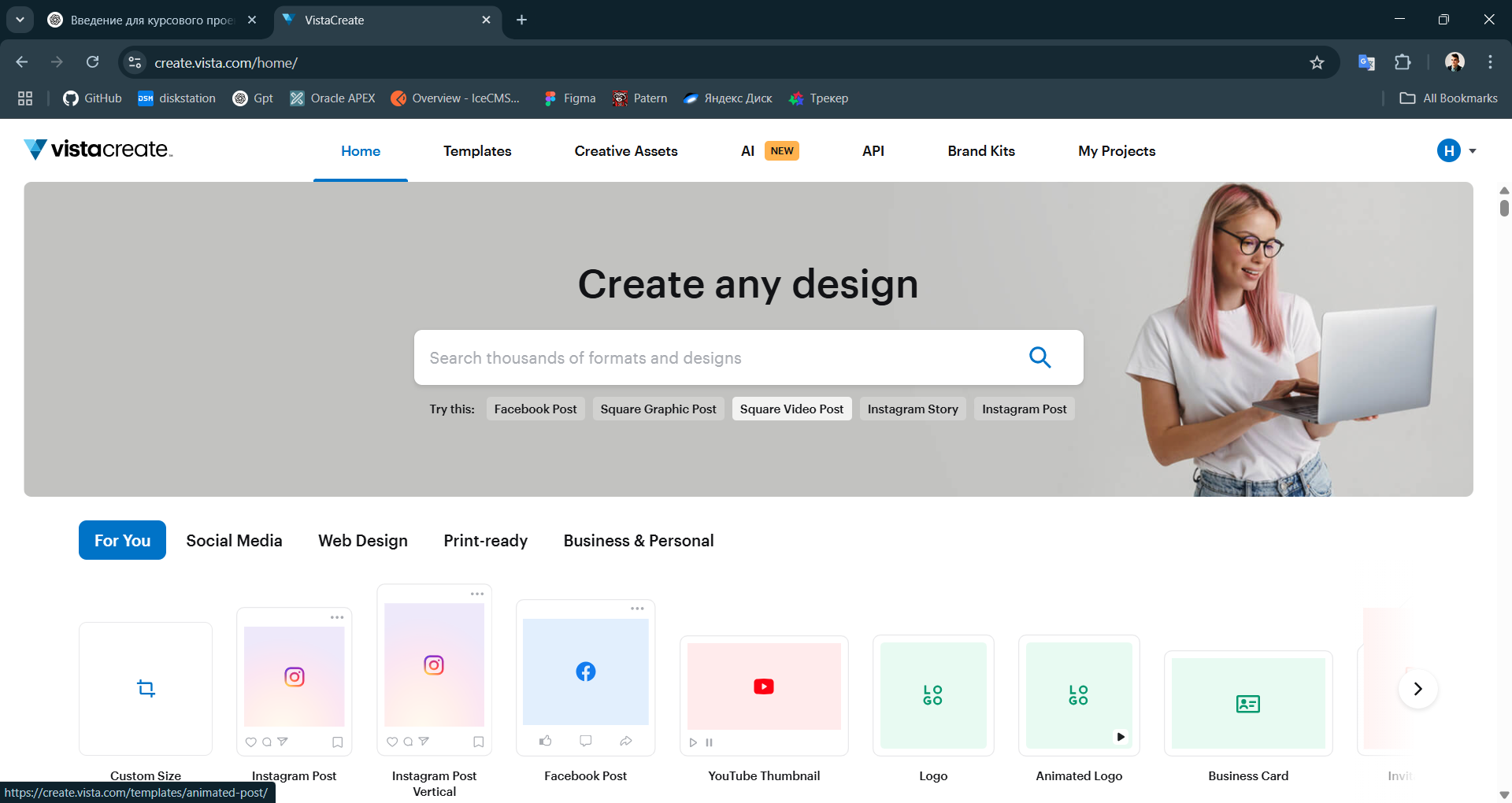
*Рисунок 1.1 – Интерфейс Canva*

Преимущества Canva:  
– широкий выбор шаблонов и графических элементов;  
– интуитивно понятный интерфейс, адаптированный как для новичков, так и для профессионалов;  
– возможность командной работы и совместного редактирования;  
– экспорт в различных форматах.

Недостатки Canva:  
– многие функции доступны только в платной версии;  
– ограниченная гибкость при создании уникального дизайна в бесплатной версии;  
– высокая нагрузка на слабых устройствах.

1. Онлайн-редактор Crello (VistaCreate)

На рисунке 1.2 представлен интерфейс редактора Crello — сервиса, аналогичного Canva [2].



*Рисунок 1.2 – Интерфейс Crello*

**Преимущества Crello:**  
– удобные инструменты редактирования и большой выбор шаблонов;  
– встроенные анимации и видео для создания динамичного контента;

– возможность экспорта без потери качества.

**Недостатки Crello:**  
– ограниченные возможности настройки шаблонов по сравнению с профессиональными редакторами;  
– наличие платных функций;  
– менее развитый пользовательский опыт по сравнению с Canva.

1. Выводы по разделу

Анализ современных онлайн-редакторов, включая популярные решения вроде Canva и Crello, выявил ряд характерных особенностей данных платформ. Несмотря на впечатляющий набор готовых шаблонов и интуитивно понятный интерфейс, эти сервисы имеют существенные ограничения, особенно заметные для профессиональных пользователей. Основные недостатки заключаются в жёсткой привязке расширенного функционала к платным подпискам, что значительно сужает возможности бесплатного использования, а также в ограниченной гибкости настроек, не позволяющей осуществлять глубокую кастомизацию элементов дизайна. Эти факторы создают существенные барьеры для пользователей, которым требуется полный контроль над процессом создания и редактирования графических материалов.

Разработка специализированного веб-приложения «Конструктор баннеров» направлена на устранение выявленных недостатков существующих решений. Ключевым преимуществом собственной разработки станет ориентация на конкретные потребности целевой аудитории, что позволит реализовать именно те функции, которые действительно необходимы пользователям в их повседневной работе.

Важным аспектом разрабатываемого решения станет его адаптивность и способность учитывать специфические требования различных групп пользователей. В отличие от универсальных редакторов, предлагающих стандартизированный набор инструментов, «Конструктор баннеров» будет предоставлять возможность тонкой настройки рабочих процессов и интерфейса под индивидуальные предпочтения. Это позволит создать по-настоящему персонализированную среду для работы с графикой, где каждый пользователь сможет организовать пространство в соответствии со своими задачами и привычками. Такой подход не только повысит эффективность работы, но и обеспечит конкурентное преимущество перед существующими массовыми решениями.

1. Проектирование web-приложения
2. Функциональность web-приложения

Функциональность web-приложения представлена на диаграмме вариантов использования. Диаграмма вариантов использования представлена на рисунке 2.1.

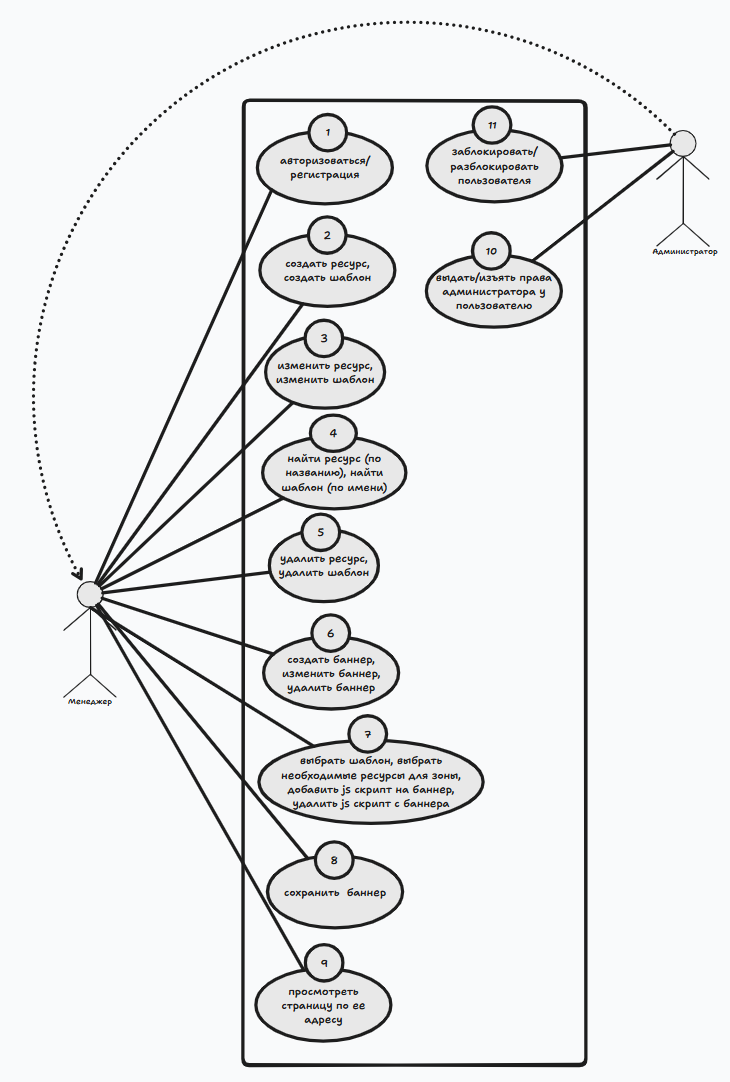


Рисунок 2.1 – Диаграмма вариантов использования

Диаграмма вариантов использования наглядно демонстрирует доступные функции для каждой роли и способы их взаимодействия с приложением.

Список ролей и их описание представлены в таблице 2.1.

Таблица 2.1 – Роли в *web*-приложения

|  |  |
| --- | --- |
| Роль | Назначение |
| Менеджер | Имеет возможность зарегистрироваться или авторизоваться, создавать ресурсы, шаблоны и страницы, публиковать их и получать html страницы. |
| Администратор | Все возможности менеджеров и так же возможности выдавать/изымать права администратора и блокировать, и разблокировать аккаунты. |

Описание функциональности приложения, доступной для роли менеджера представлено в таблице 2.2.

Таблица 2.2 – Функциональности менеджера

|  |  |  |
| --- | --- | --- |
| Номер | Функциональности | Пояснение |
| 1 | Регистрация/Авторизация | Есть возможность зарегистрироваться или авторизоваться с использованием электронной почты и пароля или с помощи OAuth-аутентификация. |
| 2 | Создать ресурс, создать шаблон | Есть возможность просматривать списки ресурсов и шаблонов и создавать новые. |
| 3 | Изменить ресурс, изменить шаблон | Есть возможность изменять уже созданные ресурсы и шаблоны |
| 4 | Найти ресурс (по названию), найти шаблон (по имени) | Можно искать ресурсы и шаблоны по их имени, а так же сортировать их. |
| 5 | Удалить ресурс, удалить шаблон | Есть возможность удалять уже созданные ресурсы и шаблоны. |

Таблица 2.2 – Функциональности менеджера

|  |  |  |
| --- | --- | --- |
| 6 | Создать баннер, изменить баннер, удалить баннер | Есть возможность создания и изменения баннера с использованием ресурсов и шаблонов, а также удалять уже созданные. |
| 7 | Выбрать шаблон, выбрать необходимые ресурсы для зоны, добавить js скрипт на баннер, удалить js скрипт с баннера | При создании нового баннера выбирается шаблон и так же есть возможность выбирать ресурсы и скрипты. |
| 8 | Сохранить баннер | Есть возможность сохранять страницу и получать результат в виде url, html и png файла. |
| 9 | Просмотреть страницу по ее адресу | Есть возможность использовать Url для просмотра сгенерированной страницы используя pageId который задается при создании страинцы |

Описание функциональности приложения, доступной для роли администратора представлено в таблице 2.3.

Таблица 2.3 – Функциональности администратора

|  |  |  |
| --- | --- | --- |
| Номер | Функциональности | Пояснение |
| 10 | Выдать/изъять права администратора у пользователя | Есть возможность выдавать пользователям права администратора или изымать если пользователь уже имеет их. |
| 11 | Заблокировать/разблокировать пользователя | Есть возможность заблокировать пользователя, тем самым закрыв ему доступ к приложению, или разблокировать, вновь открыв доступ к приложению. |

В рамках проектирования web-приложения, функциональность была детализирована на основе диаграммы вариантов использования. Созданные роли позволяют выдавать права администратора определённым пользователям, так же имеется супер администратор, которого нельзя заблокировать и у которого нельзя изъять права администратора. Каждый новый пользователь получает автоматически права менеджера и является незаблокированным.

1. Структура базы данных

Для хранения данных о пользователях и курсах, была выбрана база данных *MongoDB 7.15.0*[3]. Логическая схема базы данных представлена на рисунке 2.2.

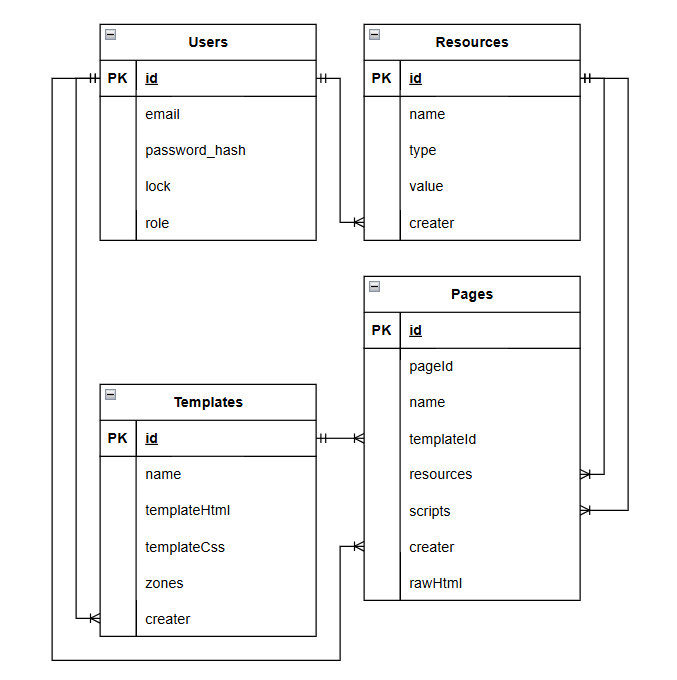


Рисунок 2.2 – Логическая схема базы данных

Схема базы данных разрабатываемого приложения состоит из 4 коллекций, состоящих из различных документов и объектов. Были созданы следующие коллекции: *Users, Resources, Templates, Pages*. Наименования и описание всех сущностей базы данных приведены в таблице 2.4.

Таблица 2.4 – Коллекции базы данных

|  |  |
| --- | --- |
| Название Коллекции | Назначение коллекции |
| Users | Хранение данных о пользователе. |
| Resources | Хранение данных о ресурсе. |
| Templates | Хранение данных о шаблоне. |
| Pages | Хранение данных о странице. |

Далее подробнее про каждую из коллекций базы данных.

Коллекция *Users* хранит данные о пользователях. Состоит из объектов, представленных в таблица 2.5.

Таблица 2.5 – Объекты коллекции User

|  |  |  |
| --- | --- | --- |
| Объект | Тип данных | Описание |
| \_id | string | Уникальный идентификатор пользователя. |
| email | string | Электронная почта пользователя. |
| passwordHash | string | Хеш Пароля пользователя. |
| lock | boolean | Флаг блокировки пользователя. |
| role | string | Роль пользователя в системе. Доступные значения: «User» — для менеджера, и «Admin» — для администратора. |

Коллекция Resources хранит данные о ресурсах. Состоит из объектов, представленных в таблица 2.6.

Таблица 2.6 – Объекты коллекции Resources

|  |  |  |
| --- | --- | --- |
| Объект | Тип данных | Описание |
| \_id | string | Уникальный идентификатор ресурса. |
| name | string | Название ресурса. |
| type | string | Указание чем является ресурс. Доступны значения: «*Text*» — простой текст или *http/css*, «*Image*» — изображение, «*js*» — *js* скрипт. |
| value | string | Содержимое ресурса. Если *text* или *js* то сам текст, если *image* то ссылка на картинку в Яндекс диске. |
| creater | string | Идентификатор создателя ресурса. |

Коллекция Templates хранит данные о шаблонах. Состоит из объектов, представленных в таблица 2.7.

Таблица 2.7 – Объекты коллекции Templates

|  |  |  |
| --- | --- | --- |
| Объект | Тип данных | Описание |
| \_id | string | Уникальный идентификатор шаблона. |
| name | string | Название шаблона. |
| templateHtml | string | Html код шаблона. |
| templateCss | string | Css код шаблона. |

Таблица 2.7 – Объекты коллекции Templates

|  |  |  |
| --- | --- | --- |
| Zones | array | Массив строк, каждая строка – название одной зоны на шаблоне. |
| creater | string | Идентификатор создателя шаблона. |

Коллекция Pages хранит данные о страницах. Состоит из объектов, представленных в таблица 2.8.

Таблица 2.8 – Объекты коллекции Pages

|  |  |  |
| --- | --- | --- |
| Объект | Тип данных | Описание |
| \_id | string | Уникальный идентификатор страницы. |
| pageId | string | Уникальный идентификатор страницы по которому ее можно просматривать. |
| name | string | Название страницы. |
| temaplteId | string | Уникальный идентификатор используемого шаблона. |
| scripts | array | Массив строк, каждая строка – уникальный идентификатор ресурса с типом *js*. |
| resources | array | Массив объектов, хранящий 2 строки. Первая строка - имя зоны из шаблона, вторая строка - уникальный идентификатор ресурса с типом *text* или *image* |
| rawHtml | string | Текст cгенерированной страницы. |
| creater | string | Идентификатор создателя страницы. |

Каждая коллекция имеет четко определенные поля, отражающие определенные аспекты работы платформы для онлайн-курсов. Определены типы данных объектов и связи между коллекциями и объектами.

1. Архитектура web-приложения

Архитектура web-приложения представлена на рисунке 2.3.

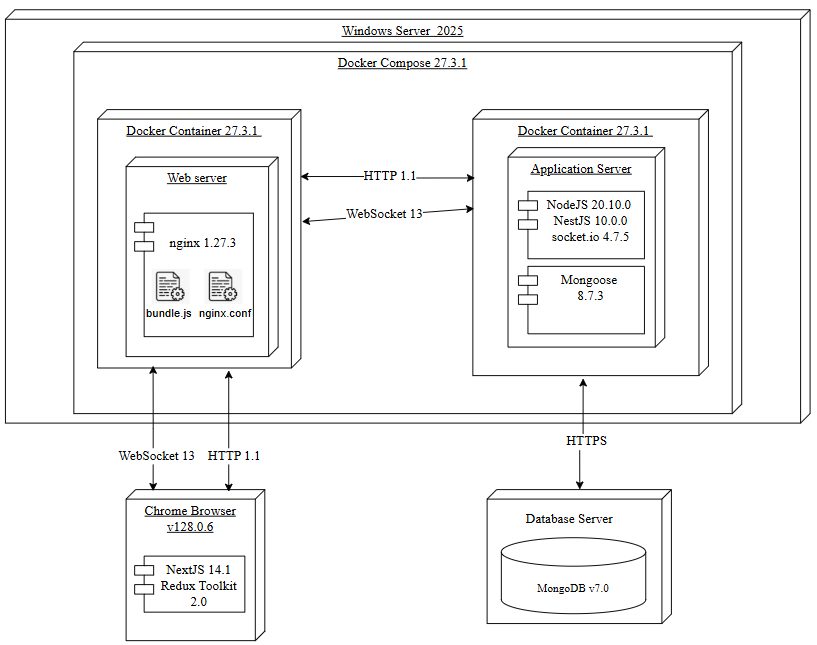


Рисунок 2.3 – Архитектура web-приложения

Приложение разворачивается с помощью docker-compose в 4 контейнерах docker. Пользователь взаимодействует с web-приложением через web-браузер.

Пояснение назначения каждого элемента web-приложения представлено в таблице 2.9.

Таблица 2.9 – Объекты коллекции QuestionContent

|  |  |
| --- | --- |
| Элемент | Назначение |
| Web Server (nginx) | Предоставляет доступ к статическим ресурсам фронтенд-части web-приложения. |
| Database Server | Используется для хранения и предоставления доступа к данным, которые необходимы для работы web-приложения. |
| Application Server | Обрабатывать запросы пользователя, запрашивать данные из базы данных |
| Chrome Browser | Отображать фронтенд-часть web-приложения, отправлять запросы пользователя, отображать ответы сервера. |

Таким образом были рассмотрены все ключевые элементы архитектуры web-приложения.

1. Выводы по разделу

В ходе анализа функциональных возможностей веб-приложения была детально изучена система ролей, включающая два ключевых уровня доступа: менеджера и администратора. Менеджеры обладают базовым набором прав, позволяющим осуществлять регистрацию и аутентификацию пользователей, а также выполнять полный цикл операций CRUD (создание, чтение, обновление и удаление) с различными ресурсами системы, включая управление шаблонами и страницами контента. Администраторы, являясь расширенной версией менеджеров, наделены дополнительными привилегиями: возможностью назначать и отзывать административные права у других пользователей, а также блокировать и разблокировать учетные записи менеджеров, что обеспечивает контроль за безопасностью и управлением доступом в системе.

Архитектурная часть исследования включала глубокий анализ логической структуры базы данных приложения, которая организована в виде четырех взаимосвязанных коллекций. Каждая коллекция обладает строго типизированной структурой полей, точно отражающей соответствующие бизнес-процессы платформы. Поля документов спроектированы с учетом всех необходимых аспектов хранения и обработки информации, что гарантирует целостность данных и эффективность выполнения запросов. Связи между коллекциями реализованы через систему ссылок и вложенных документов, обеспечивая оптимальный баланс между нормализацией данных и производительностью.

1. Реализация web-приложения
2. Программная платформа Node.js

Для серверной части проекта была выбрана платформа Node.js, основанная на событийно-ориентированной архитектуре и однопоточной модели выполнения. В сочетании с фреймворками Nest.js [4] и Next.js [5] она обеспечивает удобную разработку backend- и frontend-частей приложения.

Nest.js предоставляет мощные инструменты для создания API, маршрутизации и обработки запросов, а также поддерживает модульную структуру приложения. Next.js, в свою очередь, упрощает разработку пользовательского интерфейса и взаимодействие с серверной частью.

Использование этих технологий позволяет создавать производительные и масштабируемые веб-приложения.

1. NoSQL база данных MongoDB

Для работы web-приложения используется база данных MongoDB. Она является NoSQL базой данных, которая хранит данные в формате документов BSON (Binary JSON) [6]. Поддерживает репликацию для обеспечения отказоустойчивости и высокую производительность при обработке больших объемов данных.

1. Средство моделирования объектного документа

В проекте используется Средство моделирования объектного документа Mongoose[]. Оно предоставляет удобный способ работы с MongoDB, обеспечивая автоматическое управление схемами и типизированные модели данных. Поддерживает сложные связи между документами, облегчает выполнение операций CRUD. С помощью Mongoose определены схемы всех коллекций базы данных.

Схема User, описывающая коллекцию UserSchema представлена на листинге 3.1

|  |
| --- |
| export const UserSchema = new Schema<User>(    {      \_id: { type: String },      email: { type: String, required: true },      passwordHash: { type: String, required: true },      lock: { type: Boolean, required: true },      role: { type: String, required: true },    },    { timestamps: true },  ); |

Листинг 3.1 – Схема User

Схема UserSchema описывает структуру коллекции User объектами, представленными в таблице 3.1.

Таблица 3.1 – Объекты схемы User

|  |  |  |
| --- | --- | --- |
| Объект | Тип данных | Описание |
| \_id | string | Уникальный идентификатор. Добавляется Mongoose автоматически. Выполняет роль уникального идентификатора пользователя. |
| email | string | Электронная почта пользователя.  Имеет ограничения required — обязательный для заполнения. |
| passwordHash | string | Хеш Пароля пользователя.  Имеет ограничение required — обязательный для заполнения. |
| lock | boolean | Флаг блокировки пользователя.  Имеет ограничение required — обязательный для заполнения. |
| role | string | Роль пользователя в системе.  Имеет ограничения required — обязательный для заполнения. |

Схема ResourceSchema, описывающая коллекцию Resource представлена на листинге 3.2

|  |
| --- |
| export const ResourceSchema = new Schema<Resource>(    {      \_id: { type: String },      name: { type: String, required: true },      type: { type: String, required: true },      value: { type: String, required: true },      creater: { type: String, required: true },    },    { timestamps: true },  ); |

Листинг 3.2 – Схема ResourceSchema

Схема ResourceSchema описывает структуру коллекции Resource объектами, представленными в таблице 3.2.

Таблица 3.2 – Объекты схемы Resource

|  |  |  |
| --- | --- | --- |
| Объект | Тип данных | Описание |
| \_id | string | Уникальный идентификатор. Добавляется Mongoose автоматически. |
| name | string | Название ресурса.  Имеет ограничение required — обязательный для заполнения. |
| type | string | Тип ресурса.  Имеет ограничение required — обязательный для заполнения. |
| value | string | Данные, которые ресурс хранит.  Имеет ограничение required — обязательный для заполнения. |
| creater | string | Уникальный идентификатор пользователя, создавшего этот ресурс.  Имеет ограничение required — обязательный для заполнения. |

Схема TemplateSchema, описывающая коллекцию Template представлена на листинге 3.3

|  |
| --- |
| export const TemplateSchema = new Schema<Template>(    {      \_id: { type: String },      name: { type: String, required: true },      templateHtml: { type: String, required: true },      templateCss: { type: String, require: true },      zones: { type: [String], require: true },      creater: { type: String, required: true },    },    { timestamps: true },  ); |

Листинг 3.3 – Схема Template

Схема TemplateSchema описывает структуру коллекции Template объектами, представленными в таблице 3.3.

Таблица 3.3 – Объекты схемы Template

|  |  |  |
| --- | --- | --- |
| Объект | Тип данных | Описание |
| \_id | string | Уникальный идентификатор. Добавляется Mongoose автоматически. |
| name | string | Название шаблона.  Имеет ограничение required — обязательный для заполнения. |
| templateHtml | string | Html код шаблона.  Имеет ограничение required — обязательный для заполнения. |

Таблица 3.3 – Объекты схемы Template

|  |  |  |
| --- | --- | --- |
| templateCss | string | CSS код шаблона.  Имеет ограничение required — обязательный для заполнения. |
| zones | string[] | Имена зон на шаблоне.  Имеет ограничение required — обязательный для заполнения. |
| creater | string | Уникальный идентификатор пользователя, создавшего этот шаблон.  Имеет ограничение required — обязательный для заполнения. |

Схема PageSchema, описывающая коллекцию Page представлена на листинге 3.4

|  |
| --- |
| export const PageSchema = new Schema<Page>(    {      \_id: { type: String },      pageId: { type: String, required: true, unique: true, index: true },      name: { type: String, required: true },      templateId: { type: String, required: true },      scripts: { type: [String], require: true },      resources: { type: Map, of: String, require: true },      rawHtml: { type: String, required: true },      creater: { type: String, required: true },    },    { timestamps: true },  ); |

Листинг 3.4 – PageSchema

Схема PageSchema описывает структуру коллекции Page объектами, представленными в таблице 3.4.

Таблица 3.4 – Объекты схемы Page

|  |  |  |
| --- | --- | --- |
| Объект | Тип данных | Описание |
| \_id | string | Уникальный идентификатор. Добавляется Mongoose автоматически. |
| pageId | string | Уникальное имя, по которому можно просмотреть страницу.  Имеет ограничение required — обязательный для заполнения. |
| name | string | Уникальное имя страницы.  Имеет ограничение required — обязательный для заполнения. |
| templateId | string | Уникальный идентификатор используемого шаблона.  Имеет ограничение required — обязательный для заполнения. |
| scripts | string[] | Список уникальных идентификаторов ресурсов с типом js.  Имеет ограничение required — обязательный для заполнения. |
| resources | map | Список объектов, состоящий из 2-ух строк, первая строка – название зоны, вторая – уникальный идентификатор ресурса с типом text или image.  Имеет ограничение required — обязательный для заполнения. |
| rawHtml | string | Html код сгенерированной страницы.  Имеет ограничение required — обязательный для заполнения. |
| creater | string | Уникальный идентификатор пользователя, создавшего эту страницу.  Имеет ограничение required — обязательный для заполнения. |

1. Программные библиотеки

В процессе разработки серверной части web-приложения для обеспечения её функциональности и повышения эффективности работы системы были использованы программные библиотеки, представленные в таблице 3.5.

Таблица 3.5 – Программные библиотеки серверной части

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Назначение |
| @nestjs/jwt | 11.0.0 | JWT-аутентификация |
| @nestjs/platform-express | 11.1.0 | HTTP-сервер на Express |
| bcrypt | 5.1.1 | Хеширование паролей |
| mongoose | 8.12.2 | ODM для MongoDB |
| passport | 0.7.0 | Аутентификация через google |
| passport-jwt | 4.0.1 | JWT-стратегия для Passport |
| passport-google-oauth20 | 2.0.0 | Google OAuth 2.0 аутентификация |

В процессе разработки клиентской части web-приложения были задействованы программные библиотеки, представленные в таблице 3.6.

Таблица 3.6 – Программные библиотеки клиентской части

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Назначение |
| lucide-react | 0.487.0 | Иконки (альтернатива Font Awesome, Feather Icons) |
| radix-ui (@radix-ui/react-\*) | 2.1.6 | Готовые доступные (a11y) UI-компоненты (Select, Label, Slot) |
| tailwind-merge | 3.2.0 | Утилита для слияния Tailwind-классов без конфликтов |
| tw-animate-css | 1.3.0 | CSS-анимации для Tailwind |
| class-variance-authority | 0.7.1 | Управление вариативными классами (удобно для6 UI-китов) |
| clsx | 2.1.1 | Динамическое объединение классов (аналог classnames |

Программные библиотеки позволяют упростить реализацию web-приложения.

1. Структура серверной части

Основные компоненты структуры серверной части в NestJS основаны на модульной архитектуре, которая обеспечивает гибкость и масштабируемость приложения. В основе NestJS лежат модули, которые группируют связанные функциональные элементы, такие как:

Контроллеры — отвечают за обработку HTTP-запросов и маршрутизацию, определяя, какие методы будут вызваны для обработки конкретных маршрутов.

Сервисы — используются для реализации бизнес-логики и предоставления данных контроллерам.

Middleware — промежуточные обработчики, используемые для обеспечения безопасности.

Модули — инкапсулирует тесно связанные функциональные элементы.

В таблице 3.7 приведён список директорий проекта разработки серверной части web-приложения и назначение файлов, хранящихся в этих директориях.

Таблица 3.7 – Директории серверной части web-приложения

|  |  |
| --- | --- |
| Директория | Назначение |
| auth | Содержит файлы для регистрации и авторизации. |
| page\_module | Содержит файлы для обработки запросов, связанных со страницами. |
| resource\_module | Содержит файлы для обработки запросов, связанных с ресурсами. |
| template\_module | Содержит файлы для обработки запросов, связанных с шаблонами. |
| user\_module | Содержит файлы для обработки запросов, связанных с пользователями. |

Таблица соответствия маршрутов контроллерам и функциям в исходном коде представлена в таблице 3.8.

Таблица 3.8 – Контроллеры

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Маршрут | Контроллер | Метод Контроллера |
| Post | /auth/register | auth.controller | register |
| Post | /auth/login | auth.controller | login |
| Get | /auth/google | auth.controller | googleAuth |
| Get | /auth/redirect | auth.controller | googleAuthRedirect |
| Get | /p/:pageId | page.view.controller | getRawHtml |
| Get | /page | pages.controller | findAll |
| Get | /page/:pageId | pages.controller | findOne |
| Post | /page | pages.controller | create |
| Put | /page/:id | pages.controller | update |
| Delete | /page/:id | pages.controller | delete |
| Get | /resource | resources.controller | findAll |
| Get | /resource/:id | resources.controller | findOne |
| Post | /resource | resources.controller | create |
| Put | /resource/:id | resources.controller | update |
| Delete | /resource/:id | resources.controller | delete |
| Get | /template | template.controller | findAll |
| Get | /template/:id | template.controller | findOne |
| Post | /template | template.controller | create |
| Put | /template/:id | template.controller | update |
| Delete | /template/:id | template.controller | delete |
| Get | /user | user.controller | findAll |
| Get | /user/:id | user.controller | findOne |
| Post | /user | user.controller | create |
| Put | /user/:id | user.controller | update |
| Delete | /user/:id | user.controller | delete |

При передаче данных между клиентом и сервером используется формат JSON (JavaScript Object Notation).

1. Реализация функций для менеджера
2. Регистрация

Для менеджера доступна регистрация, которая позволяет ему создать учетную запись в системе. Этот процесс реализован в методе register контроллера AuthController. Реализация метода представлена на листинге 3.5.

|  |
| --- |
| @Post('register')  async register(@Body() dto: CreateUpdateUserDto) {  return await this.authService.register(dto);  } |

Листинг 3.5 – Реализация метода register

Метод принимает HTTP-запрос с данными типа CreateUpdateUserDto. Вызывает метод register сервиса authService, который проверяет корректность данных, записывает их в базу данных, генерирует токены и возвращает их. После возврата токенов, метод register записывается в localStorage записывает.

1. Авторизация

Менеджер может аутентифицироваться для получения доступа к ресурсов, шаблонам и страницам. Этот процесс реализован в методе login контроллера AuthController. Реализация метода представлена на листинге 3.6.

|  |
| --- |
| @Post('login')  async login(@Body() dto: { email: string; password: string }) {    const user = await this.authService.validateUser(dto.email, dto.password);    if (!user) {      throw new UnauthorizedException('Invalid credentials');    }    return this.authService.login(user);  } |

Листинг 3.6 – Реализация метода login

Он принимает HTTP-запрос с данными типа { email: string; password: string }. Вызывает метод login сервиса authService, который проверяет корректность данных, генерирует токены и возвращает их. После возврата токенов, метод login записывает в localStore и передает access-токен в теле ответа. В результате выполнения, клиенту возвращается CreateUpdateUserDto.

1. Просмотр ресурсов

Менеджер может просматривать список ресурсов. Этот процесс реализован в методе findAll контроллера ResourceController. Реализация метода представлена на листинге 3.7.

|  |
| --- |
| @Get()  findAll(): Promise<ResourceDto[]> {    return this.resourcesService.findAll();  } |

Листинг 3.7 – Реализация метода findAll

Он ничего не принимает. Вызывает метод findAll сервиса ResourceService, который возвращает список типа ResourceDto[] который извлекается из базы данных.

1. Создание ресурса

Менеджер может создавать новый ресурс. Этот процесс реализован в методе create контроллера ResourceController. Реализация метода представлена на листинге 3.8.

|  |
| --- |
| @Post()  create(@Body() resourceDto: CreateUpdateResourceDto):Promise<ResourceDto> {  return this.resourcesService.create(resourceDto);  } |

Листинг 3.8 – Реализация метода create

Он принимает данные типа CreateUpdateUserDto. Вызывает метод create сервиса ResourceService, который возвращает объект типа ResourceDto который извлекается из базы данных.

1. Обновление ресурса

Менеджер может обновлять созданный ресурс. Этот процесс реализован в методе update контроллера ResourceController. Реализация метода представлена на листинге 3.9.

|  |
| --- |
| @Put(':id')    async update(      @Param('id') id: string,      @Body() resourceDto: CreateUpdateResourceDto,    ): Promise<ResourceDto> {      return this.resourcesService.update(id, resourceDto);  } |

Листинг 3.9 – Реализация метода update

Он принимает id как параметр и данные типа CreateUpdateUserDto как json объект. Вызывает метод update сервиса ResourceService, который возвращает объект типа ResourceDto который извлекается из базы данных.

1. Удаление ресурса

Менеджер может удалять созданный ресурс. Этот процесс реализован в методе delete контроллера ResourceController. Реализация метода представлена на листинге 3.10.

|  |
| --- |
| @Delete(':id')    async delete(@Param('id') id: string): Promise<void> {      await this.resourcesService.delete(id);  } |

Листинг 3.10 – Реализация метода delete

Он принимает id как параметр. Вызывает метод delete сервиса ResourceService, который ничего не возвращает.

1. Просмотр шаблонов

Менеджер может просматривать список шаблонов. Этот процесс реализован в методе findAll контроллера TemplateController. Реализация метода представлена на листинге 3.11.

|  |
| --- |
| @Get()    async findAll(): Promise<TemplateDto[]> {      return this.templateService.findAll();  } |

Листинг 3.11 – Реализация метода findAll

Он ничего не принимает. Вызывает метод findAll сервиса TemplateService, который возвращает список типа TemplateDto[] который извлекается из базы данных.

1. Создание шаблона

Менеджер может создавать новый шаблон. Этот процесс реализован в методе create контроллера TemplateController. Реализация метода представлена на листинге 3.12.

|  |
| --- |
| @Post()  async create(@Body() templateDto: CreateUpdateTemplateDto): Promise<TemplateDto> {    return this.templateService.create(templateDto);  } |

Листинг 3.12 – Реализация метода create

Он принимает данные типа CreateUpdateTemplateDto. Вызывает метод create сервиса TemplateService, который возвращает объект типа TemplateDto который извлекается из базы данных.

1. Обновление шаблона

Менеджер может обновлять созданный шаблон. Этот процесс реализован в методе update контроллера TemplateController. Реализация метода представлена на листинге 3.13.

|  |
| --- |
| @Put(':id')    async update(      @Param('id') id: string,      @Body() templateDto: CreateUpdateTemplateDto,    ): Promise<TemplateDto> {      return this.templateService.update(id, templateDto);  } |

Листинг 3.13 – Реализация метода update

Он принимает id как параметр и данные типа CreateUpdateTemplateDto как json объект. Вызывает метод update сервиса TemplateService, который возвращает объект типа TemplateDto который извлекается из базы данных.

1. Удаление шаблона

Менеджер может удалять созданный шаблон. Этот процесс реализован в методе delete контроллера TemplateController. Реализация метода представлена на листинге 3.14.

|  |
| --- |
| @Delete(':id')    async delete(@Param('id') id: string): Promise<void> {      return this.templateService.delete(id);  } |

Листинг 3.14 – Реализация метода delete

Он принимает id как параметр. Вызывает метод delete сервиса TemplateService, который ничего не возвращает.

1. Просмотр страниц

Менеджер может просматривать список страниц. Этот процесс реализован в методе findAll контроллера PageController. Реализация метода представлена на листинге 3.15.

|  |
| --- |
| @Get()    async findAll(): Promise<PageDto[]> {      return this.pagesService.findAll();    } |

Листинг 3.15 – Реализация метода findAll

Он ничего не принимает. Вызывает метод findAll сервиса PageService, который возвращает список типа PageDto[] который извлекается из базы данных.

1. Создание страницы

Менеджер может создавать новую страницу. Этот процесс реализован в методе create контроллера PageController. Реализация метода представлена на листинге 3.16.

|  |
| --- |
| @Post()    async create(@Body() createPageDto: CreateUpdatePageDto): Promise<PageDto> {      return this.pagesService.create(createPageDto);    } |

Листинг 3.16 – Реализация метода create

Он принимает данные типа CreateUpdatePageDto. Вызывает метод create сервиса PageService, который возвращает объект типа PageDto который извлекается из базы данных.

1. Обновление страницы

Менеджер может обновлять созданную страницу. Этот процесс реализован в методе update контроллера PageController. Реализация метода представлена на листинге 3.17.

|  |
| --- |
| @Put(':id')    async update(@Param('id') id: string, @Body() updatePageDto: CreateUpdatePageDto): Promise<PageDto|null> {      return this.pagesService.update(id, updatePageDto);    } |

Листинг 3.17 – Реализация метода update

Он принимает id как параметр и данные типа CreateUpdatePageDto как json объект. Вызывает метод update сервиса PageService, который возвращает объект типа PageDto который извлекается из базы данных.

1. Удаление страницы

Менеджер может удалять созданную страницу. Этот процесс реализован в методе delete контроллера PageController. Реализация метода представлена на листинге 3.18.

|  |
| --- |
| @Delete(':id')    async delete(@Param('id') id: string): Promise<void> {      await this.pagesService.delete(id);    } |

Листинг 3.18 – Реализация метода delete

Он принимает id как параметр. Вызывает метод delete сервиса PageService, который ничего не возвращает.

1. Просмотр сгенерированной страницы

Любой пользователь может просмотреть сгенерированную страницу. Этот процесс реализован в методе getRawHtml контроллера PageViewController. Реализация метода представлена на листинге 3.19.

|  |
| --- |
| @Get()      async getRawHtml(@Param('pageId') pageId: string): Promise<string> {        return this.pagesService.findOne(pageId);    } |

Листинг 3.19 – Реализация метода getRawHtml

Он принимает id как параметр и выдает страницу для просмотра.

1. Структура клиентской части

Клиентская часть приложения реализована с использованием Next.js. Основная логика и элементы пользовательского интерфейса размещены в модуле frontend. Директории представлены в таблице 3.9.

Таблица 3.9 – Директории клиентской части web-приложения

|  |  |
| --- | --- |
| Директория | Назначение |
| pages | Содержит все файлы страниц. |

Таблица 3.9 – Директории клиентской части web-приложения

|  |  |
| --- | --- |
| modules | Содержит файлы с модулями для работы с пользовательскими типами. |
| services | Содержит сервис для работы с базой данных. |
| helpModules | Содержит вспомогательные модули для вывода результатов. |

Таблица соответствия маршрутов и компонентов страниц представлена в таблице 3.10.

Таблица 3.10 – Маршруты и компоненты страниц

|  |  |  |  |
| --- | --- | --- | --- |
| Компонеты страниц | Маршрут | Роли | Назначение компонентов |
| login | /login | Все пользователи | Страница, содержащая формы регистрации и аутентификации. |
| home | /home | User, admin | Стартовая страница, где можно увидеть количество созданных объектов и перейти на страницы со списками. |
| resource | /resource | User, admin | Страница, на которой можно просмотреть список всех ресурсов. |
| creater | /resource/creater | User, admin | Страница либо для создания, либо для обновления существующего ресурса. |
| template | /template | User, admin | Страница, на которой можно просмотреть список всех шаблонов. |
| builder | /template/builder | User, admin | Страница либо для создания, либо для обновления существующего шаблона. |
| page | /page | User, admin | Страница, на которой можно просмотреть список всех страниц. |
| editor | /page/editor | User, admin | Страница либо для создания, либо для обновления существующей страницы. |
| user | /user | Admin | Страница, на которой можно просмотреть список всех пользователей. |
| viewPage | /p | Все пользователи | Страница, где можно посмотреть сгенерированную страницу. |

Компоненты и страницы реализуют все необходимые функции для различных ролей. Код модуля генерации финальной страницы представлен в приложении В. Код для авторизации через google аккаунт представлен в приложении Г.

1. Выводы по разделу

Веб-приложение разработано с использованием современного технологического стека, обеспечивающего высокую производительность и удобство разработки. Серверная часть приложения построена на платформе Node.js с применением фреймворка Nest.js, который предоставляет структурированный подход к созданию масштабируемых и поддерживаемых серверных приложений. Для хранения данных выбрана NoSQL-база данных MongoDB, известная своей гибкостью и высокой скоростью работы с большими объемами данных. Взаимодействие с MongoDB осуществляется через библиотеку Mongoose, которая упрощает моделирование документов и обеспечивает удобный API для работы с данными.

Клиентская часть приложения разработана с использованием фреймворка Next.js, который позволяет создавать быстрые и интерактивные пользовательские интерфейсы. Next.js поддерживает компонентный подход, что значительно упрощает разработку и повторное использование кода. Этот фреймворк также предоставляет возможности серверного рендеринга и статической генерации страниц, что улучшает SEO и производительность приложения. В результате клиентская часть получается отзывчивой, удобной для пользователей и легко расширяемой.

1. Тестирования web-приложения
2. Функциональное тестирование

Для проверки корректности работы всех функций разработанного web-приложения было проведено ручное тестирование, описание и итоги которого представлены в таблице 4.1.

Таблица 4.1 – Описание тестирования функций web-приложения

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Функция | Описание тестирования | Итог |
| 1 | Регистрация/Авторизация |  | Успешно |
| 2 | Создать ресурс, создать шаблон |  | Успешно |
| 3 | Изменить ресурс, изменить шаблон |  | Успешно |
| 4 | Найти ресурс (по названию), найти шаблон (по имени) |  | У спешно |
| 5 | Удалить ресурс, удалить шаблон |  | Успешно |
| 6 | Создать баннер, изменить баннер, удалить баннер |  | Успешно |
| 7 | Выбрать шаблон, выбрать необходимые ресурсы для зоны, добавить js скрипт на баннер, удалить js скрипт с баннера |  | Успешно |
| 8 | Сохранить баннер |  | Успешно |
| 9 | Просмотреть страницу по ее адресу |  | Успешно |
| 10 | Выдать/изъять права администратора у пользователя |  | Успешно |
| 11 | Заблокировать/разблокировать пользователя |  | Успешно |