Автономная некоммерческая организация
профессионального образования
«Московский областной колледж информации и технологий»

Специальность
09.02.07 Информационные системы
и программирование

ДОПУСК К ЗАЩИТЕ:
Приказ № _____
от «____» _____202__г.

ДИПЛОМНАЯ РАБОТА

Тема: «Разработка социальной сети»_____

Обучающийся Макаров Денис Владимирович_____/ _____ /
Ф. И. О. подпись

Руководитель: Моринов Артём Романович_____/_____ /
Ф. И. О. подпись

Консультант: _____/_____/
Ф. И. О. подпись

Дата представления работы: 05 июня 2023 г.

Подольск 2023 г

Автономная некоммерческая организация
профессионального образования
«Московский областной колледж информации и технологий»

ЗАДАНИЕ
на выполнение дипломной работы

Обучающийся Макаров Денис Владимирович
фамилия, имя, отчество

форма обучения очная, группа ИСиПо-9/19
очная/заочная/очно-заочная

специальность 09.02.07 «Информационные системы и программирование»
наименование

1. Тема «Разработка социальной сети»

2. Дата выдачи темы 15 декабря 2022 г.

3. Календарный график выполнения Введение, 1. Глава Теоретическая часть 01 апреля 2023
2. Глава Разработка социальной сети 20 мая 2023
3. Заключение, оформление работы 30 мая 2023

4. Содержание пояснительной записки Введение, 1 Глава Теоретическая часть, 2 Глава Разработка социальной сети, Заключение, Список использованных источников, Приложения

5. Срок представления обучающимся законченной ДР:
05 июня 2023 г.

Руководитель преподаватель ПЦК ИД Моринов Артём Романович
Ф.И.О., ученая степень, должность, место работы, подпись

Обучающийся / Макаров Денис Владимирович
(подпись) (ФИО)

Календарный план выполнения дипломной работы

№ п/п	Название раздела работы	Срок выполнения	Отметка о выполнении
1.	Введение, 1. Глава Теоретическая часть	01 апреля 2023	
2.	Глава, 2 Разработка социальной сети	20 мая 2023	
3.	Прохождение нормоконтроля	30 мая 2023	

Обучающийся _____ Макаров Денис Владимирович__
(подпись) (ФИО)

Руководитель _____ Моринов Артём Романович_____
(подпись) (ФИО)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Социальные сети стали неотъемлемой частью нашей жизни. Сегодня люди используют социальные сети для общения, поиска работы, продвижения своих бизнесов и многих других целей. С развитием технологий и доступностью интернета социальные сети продолжают расти и развиваться, создавая новые возможности для пользователей. Создание социальной сети – это актуальная тема в настоящее время, которая может привести к созданию новых возможностей для пользователей интернета. [9]

В настоящее время социальные сети являются неотъемлемой частью нашей жизни. Более 4,2 миллиардов человек используют социальные сети для общения, поиска работы, продвижения своих бизнесов и многих других целей. Они предоставляют уникальные возможности для пользователей интернета, позволяя им создавать и поддерживать связи, обмениваться информацией и мнениями, получать новости и развлечения.

Существует множество социальных сетей, таких как Facebook, Instagram, Twitter, LinkedIn, TikTok и многие другие, каждая из которых предлагает свои уникальные функции и возможности для пользователей. Однако, несмотря на множество существующих социальных сетей, все еще существует потребность в создании новых социальных сетей с более уникальными возможностями.

Например, многие социальные сети сегодня предлагают функцию постов, где пользователи могут публиковать фотографии и текстовые сообщения. Однако, возможности этих постов ограничены и не всегда могут удовлетворить потребности пользователей. Создание социальной сети с более разнообразными функциями, такими как возможность изменения цветовой темы сайта, просмотр профилей других пользователей, посты, сделанные только ими, может привести к созданию новых возможностей для пользователей интернета.

Цель данной дипломной работы заключается в разработке и создании новой социальной сети, которая будет предлагать уникальные функции и возможности для пользователей.

Для достижения этой цели будут поставлены следующие Задачи:

- Разработка дизайна и функционала социальной сети, включая возможности для создания постов, отметок "Нравится", написание комментариев, добавление и удаление пользователей в друзья и возможность изменения цветовой темы сайта.
- Создание механизмов обработки данных, таких как хранение и управление информацией о пользователях, постах, комментариях, друзьях, их взаимодействиях и т. д.
- Определение и разработка системы безопасности социальной сети для защиты данных пользователей и предотвращения возможных атак и нарушений.
- Проведение тестирования и отладки созданной социальной сети для обеспечения ее корректной работы и удобства использования для пользователей.

Для достижения поставленных целей и задач в работе будут использованы следующие методы исследования:

- Анализ существующих социальных сетей и их функционала, для выявления преимуществ и недостатков, а также определения уникальных возможностей, которые можно реализовать в создаваемой социальной сети.
- Проектирование дизайна и функционала социальной сети на основе собранной информации и с учетом потребностей пользователей.
- Разработка и тестирование программного обеспечения, включая базы данных и веб-интерфейсы, для обеспечения корректной работы социальной сети.

- Использование методов тестирования и отладки, таких как тестирование единиц кода, функциональное тестирование и интеграционное тестирование, для обеспечения качественной работы созданной социальной сети.
- Применение методов монетизации, таких как контекстная реклама, продажа дополнительных функций, для обеспечения финансовой устойчивости созданной социальной сети.

Результаты данной работы могут быть использованы для создания новой социальной сети с более уникальными функциями и возможностями, которые не предоставляются в существующих социальных сетях. Создание такой социальной сети может привести к новым возможностям для пользователей интернета, а также к развитию индустрии социальных сетей в целом. Кроме того, разработанные методы и технологии могут быть применены в других проектах разработки программного обеспечения, особенно в области веб-разработки и баз данных. [12, с. 20]

Таким образом, создание новой социальной сети является актуальной и перспективной задачей, которая может привести к новым возможностям для пользователей интернета, а также к развитию индустрии социальных сетей. Это может привести к появлению новых форм взаимодействия между людьми и расширению их возможностей в области обмена информацией, поиска новых знакомств и общения с друзьями. Кроме того, создание новой социальной сети может быть выгодным для бизнеса, так как может привести к появлению новых возможностей для рекламы и продвижения товаров и услуг.

ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Обзор социальных сетей

1.1.1 История развития социальных сетей

Социальная сеть — это интерактивный многопользовательский веб-сайт, который предоставляет возможность общения между пользователями с помощью различных функций и инструментов. Важно отметить, что контент на сайте создается и наполняется самими участниками сети. В Интернете существует множество социальных сетей, таких как Facebook, Instagram, Twitter, LinkedIn и многие другие. [Рис – 1.1]

Интернет — это мировая компьютерная сеть, которая позволяет пользователям обмениваться информацией, общаться и работать удаленно. Он стал самым массовым и оперативным источником информации, где можно найти информацию по любой теме в любой области человеческой деятельности. Интернет также предоставляет возможность для творчества и самовыражения.

В настоящее время социальные сети широко используются и с каждым днём становятся более популярны. Термин "социальная сеть" был введен социологом Джеймсом Барнсом в 1954 году и стал широкоупотребительным на Западе во второй половине XX века. Изначально социальные сети были связаны с исследованием социальных связей и отношений, но со временем они начали включать не только людей, но и другие объекты, имеющие социальные связи, такие как города, страны, фирмы и сайты. [Рис – 1.2]

С развитием технологий Web 2.0 социальные сети получили основу в виде порталов и веб-сервисов. Через такие сайты можно увидеть цепочку знакомств, связывающих вас с незнакомыми людьми.

Первой социальной сетью в Интернете считается Classmates.com, созданная в 1995 году. Затем последовали LinkedIn, MySpace и Facebook,

которые стали основными игроками в этой области. LinkedIn была создана для деловых контактов, а MySpace и Facebook делали акцент на самовыражении. Социальные сети стали местом, где люди могут создать свою виртуальную личность и делиться своим творчеством с другими пользователями.

Существуют также социальные сети, которые помогают искать объекты интересов, такие как веб-сайты или музыка. Обычно на сайте социальной сети пользователи указывают информацию о себе, по которой их могут найти другие участники. Сети могут быть открытыми или закрытыми, и они часто имеют систему друзей и групп.

Социальные сети в Интернете растут с невероятной скоростью и привлекают большую аудиторию. Они стали четвертой по популярности онлайн-категорией, опережая даже электронную почту. Их использование растет быстрее других секторов Интернета. В России социальные сети появились относительно недавно, но некоторые из них стали очень популярными. Вопрос о том, являются ли социальные сети благом или злом, вызывает дискуссии и требует изучения социологами.

Форумы и блоги являются распространенными формами онлайн-общения. С развитием этих форм общения возникли социальные сети, где участники связаны не только средой общения, но и социальными связями между собой. Термин "социальная сеть" был введен социологом Джеймсом Барнсом в 1954 году. Социальные сети в Интернете эффективно привлекают посетителей на сайты, обеспечивают обратную связь и являются средством генерации ценного интернет-контента.

Социальная сеть Classmates.com, открытая в 1995 году, считается одной из первых. Официальным началом "бума" социальных сетей считаются 2003–2004 годы, когда были запущены LinkedIn, MySpace и Facebook. Все социальные сети требуют регистрации пользователей, использующих уникальные учетные записи с логином, паролем и адресом электронной почты. Работа в сети осуществляется через сеансы, где происходит идентификация пользователя. Кроме учетных данных, пользователь может настраивать свой

профиль, включая внешний вид, дополнительную информацию о себе, интересы и контакты.

1.1.2 Возможности и функции соцсетей

Для большинства пользователей соцсети – это площадки для общения, развлечения и обмена новостями, но на этом их довольно широкие возможности и преимущества не ограничиваются. Далее в статье рассмотрим, что можно делать с помощью социальных сетей. [Рис – 1.3]

1) Общаться. Соцсети – удобная площадка для:

- общения с друзьями, коллегами, родственниками и т. д.;
- поиска людей, с которыми когда-то оборвалась связь;
- новых знакомств для дружбы и романтических отношений.

Поддерживать общение можно с помощью личных сообщений или в сообществах, в которых собраны люди по интересам или другим критериям. Например, группа любителей кошек или группа жителей одного города.

2) Обмениваться новостями. Это касается не только постов о событиях в жизни друзей, но и о классических новостях: что происходит в мире, в стране, в обществе, в природе. В сетях новости распространяются очень быстро и охватывают большую аудиторию пользователей, чем привычные СМИ.

3) Создавать виртуальное пространство и продвигать бизнес. Создавать виртуальное пространство и продвигать бизнес. Компании используют социальные сети наряду с частными лицами. Практически каждая фирма сегодня имеет страницы в соцсетях, благодаря которым формируется имидж предприятия, широко распространяется информация о новых продуктах и даже совершаются продажи. Тем не менее, компаниям важно следить за тем, какой контент публикуется, иначе неизбирательный постинг на сайтах может повлечь за собой негативные последствия для бренда.

4) Учиться. Учиться. Соцсети отлично подходят для обучения. Здесь можно черпать полезную информацию по интересам (например, кулинарные рецепты, лайфхаки, советы по уходу за ребенком), читать отдельные статьи авторитетных авторов на определенные темы, быть подписчиком узкоспециализированного блога. Сети позволяют даже организовывать полноценные курсы, вебинары и мастер-классы, в т. ч. проходящие в режиме реального времени.

5) Хранить файлы. Хранить файлы. В аккаунте пользователя в соцсети можно собрать документы в различных форматах, фотографии, изображения, инфографики, видеоролики, презентации. Файлы можно скачать, а также делиться ими с друзьями на своей странице, отправлять в личных сообщениях, публиковать в группах, чтобы с ними ознакомилась большая аудитория сайта.

6) Искать партнеров, сотрудников или работу. Профессиональные сети предназначены для развития деловых связей, обмена опытом, поиска работников и работодателей.

7) Наблюдать за жизнью других людей. Кто-то женился, побывал в отпуске на экзотическом острове, прыгнул с парашютом или завел новые отношения – о таких событиях из жизни знакомых люди чаще всего узнают благодаря соцсетям.

Кроме этого, в соцсетях можно оставлять и получать отзывы, обмениваться полезными данными и контентом, делать закладки, писать авторские заметки, создавать фотоальбомы, давать рекламу, выкладывать свое творчество для независимой оценки и многое другое. [Рис – 1.4]

Все это поможет пользователям решать следующие задачи:

- объединяться в узкоспециализированные сообщества для обсуждения общих тем, обмена мнениями, рекомендациями, опытом;
- искать и выбирать товары, услуги, компании, бренды, учебные заведения, курсы, руководствуясь комментариями пользователей и оценками основных экспертов;

- получать актуальную информацию о событиях от информационных агентств, следить за происходящим, в т. ч. в прямом эфире на сайте;
- знакомиться с историей различных народов и стран, особенностями их культуры, виртуально путешествовать, а также искать и планировать интересные туристические маршруты;
- организовывать совместную с другими пользователями работу над исследованиями, редактировать файлы с общим доступом.

Соцсети способны существенно облегчить жизнь людям благодаря более простой коммуникации. Даже глубокие интроверты имеют возможность реализовать себя, не покидая зоны комфорта.

1.1.3 Виды и типы соцсетей

Существуют разные классификации соцсетей, в нашей статье рассмотрим самые распространенные их них.

В зависимости от назначения и типа контента различают несколько видов социальных сетей. Разберем основные из них в таблице.

Таблица 1.1

Виды социальных сетей

Вид соцсети	Возможности	Примеры соцсетей
Для общения	Переписка с друзьями, общение, поиск людей, знакомства	ВКонтакте, Одноклассники, Facebook, Google+
Для обсуждений, дискуссий, вопросов	Ведение диалогов, обмен знаниями, мнениями, опытом	Reddit, Quora, Digg
Для распространения контента	Публикация фото, видео, музыки, книг, ведение блогов	YouTube, Flickr, Instagram

Продолжение таблицы 1.1

Для интересов	Поиск единомышленников по увлечениям, профессиональной деятельности, бизнесу	Goodreads, Last.fm
Для обзоров и отзывов	Поиск и выбор товаров и услуг для заказа, отзывы других людей, мнения профессионалов	Отзовик, Яндекс Маркет, TripAdvisor, Uber
Для авторских заметок	Публикация записей, создание постов, ведение блогов	Livejournal (Живой Журнал), Twitter, Tumblr

В формате Веб 2.0 выделяют следующие типы сетей:

Таблица 1.2

Типы социальных сетей

Тип соцсети	Возможности	Примеры
Социальные закладки (social bookmarking)	Предоставление другим пользователям списка ресурсов или закладок, поиск людей с общими данными и интересами	Pinterest, Delicious
Социальные каталоги (social cataloging)	Работа с базами данных научных статей и цитат	Academic Search Premier, Academic University, Connotea, LexisNexis, CiteULike
Социальные библиотеки	Публикация в свободном доступе ссылок на книги, файлы, аудиозаписи	IMDb.com, discogs.com
Социальные медиахранилища	Хранение медиафайлов совместно большим числом пользователей	RuTracker
Корпоративные	Сопровождение деятельности компании, решение его задач	Обычно организуются на базе существующих универсальных соцсетей

Продолжение таблицы 1.2

Сервисы для совместной работы с документами	Размещение файлов для определенного круга пользователей, которые могут в любое время читать и редактировать их на сервисе	Google Docs
Геосоциальные сети	Знакомства и общение пользователей с учетом их географического положения	AlterGeo
Многопользовательские сетевые игры (Massively Multiplayer Online Games)	Имитация виртуального мира с пользовательскими уровнями, системами подсчета баллов, выигравшими и проигравшими	World of Warcraft
Многоязычные соцсети	Общение между людьми, которые говорят на разных языках, с помощью специального ПО – онлайн-переводчиков, интегрированных в сервис	Dudu
Гендерные и возрастные соцсети	Общение пользователей конкретной возрастной группы и гендерной принадлежности	Соцсети для детей, девушек, женщин, мужчин, людей старшего поколения
Профессиональные	Общение специалистов, фрилансеров, обмен информацией и опытом, развитие уникальных деловых связей, предложение работы и поиск вакансий	Мой Круг, LinkedIn, Профессионалы.ру, MarketingPeople, Доктор на работе, Viadeo

Социальные сети могут иметь несколько форматов:

- веб сайт;
- онлайн сервис;
- мобильное приложение;
- специализированная платформа.

Также существуют сети, которые представлены одновременно в нескольких форматах для большей универсальности и удобства пользователей.

1.1.4 Популярные социальные сети

В Интернете есть множество социальных сетей, каждая с уникальными функциями и возможностями. Следующий список представляет популярных социальных сетей в России и СНГ. [Рис – 1.5]

ВКонтакте — это одна из первых и самых популярных социальных сетей в России. Создана в 2006 году Павлом Дуровым, продана компании Mail.ru Group и стала одной из самых посещаемых социальных платформ в мире. [18]

Facebook — это мировой лидер среди социальных сетей, основанный в 2004 году Марком Цукербергом. В России русскоязычная версия Facebook продолжает быть популярной, несмотря на санкционные меры Роскомнадзора.

Telegram — это отечественный мессенджер, созданный Павлом Дуровым. Telegram известен своей высокой конфиденциальностью и широким функционалом для коммерческой рекламы. [6, с. 13]

WhatsApp — это мобильный мессенджер, принадлежащий Facebook Inc. Он использует телефонную книгу для добавления контактов и обеспечивает обмен сообщениями. [25]

Instagram — это популярная социальная сеть, блокированная в России с марта 2022 года. Она предназначена для публикации фотографий и коротких видеороликов, особенно популярна среди молодежи.

Одноклассники — это одна из старейших российских социальных сетей, привлекающая взрослую аудиторию старше 25 лет. Она хорошо подходит для развития бизнеса и рекламы товаров, интересных этой аудитории. [23]

TikTok — это относительно новая, но очень популярная социальная сеть среди подростков и молодежи. Она специализируется на публикации коротких видеороликов и предлагает пользователям возможность заработка. [7, с. 33]

LinkedIn — это профессиональная социальная сеть, нацеленная на деловые отношения и поиск работы. Она предоставляет возможность специалистам находить вакансии и работодателям находить квалифицированных сотрудников. [28]

Дзен — это российская м платформа, разработанная компанией Яндекс. Дзен представляет собой платформу для публикации и потребления контента. Пользователи могут создавать свои блоги, публиковать статьи, фотографии и видео, а также подписываться на интересующие их темы и авторов. [24]

Эти социальные сети предоставляют различные возможности и охватывают разные аудитории. Они играют важную роль в обмене информацией, коммуникации и развлечении. В зависимости от ваших интересов и целей, вы можете выбрать подходящую социальную сеть или использовать несколько платформ одновременно.

1.2 Принципы проектирования социальных сетей

В настоящее время социальные сети являются одним из наиболее популярных видов веб-сервисов. Они объединяют людей со всего мира, позволяют им общаться, делиться информацией и взаимодействовать друг с другом. В этой главе мы рассмотрим принципы проектирования социальных сетей, которые помогут создать успешный и популярный веб-сервис.

Анализ конкурентов

Одним из важных этапов создания социальной сети является анализ конкурентов. Это позволит определить их преимущества и недостатки, а также выявить возможности для создания уникального продукта.

Для начала следует изучить существующие социальные сети, которые находятся в топе популярности. Необходимо определить, какие функции и возможности есть в каждой социальной сети, как они работают и как они привлекают пользователей.

Также следует изучить конкурентов в выбранной нише. Если целевая аудитория социальной сети — это профессионалы в определенной области, то следует изучить социальные сети, которые ориентированы на эту аудиторию. Если это социальная сеть для любителей спорта, то следует изучить аналогичные социальные сети.

Основные пункты, которые следует изучить при анализе конкурентов:

- Целевая аудитория. Необходимо определить, какие пользователи используют конкурирующие социальные сети. Какие возрастные группы и интересы они имеют.
- Функционал. Необходимо изучить, какие функции и возможности есть в конкурирующих социальных сетях. Какие функции наиболее популярны среди пользователей.
- Дизайн и интерфейс. Необходимо изучить дизайн и интерфейс конкурирующих социальных сетей. Какие цветовые решения и элементы дизайна наиболее популярны.

Кроме того, следует изучить, какие проблемы есть у конкурирующих социальных сетей и какие недостатки можно исправить в своей социальной сети. Например, многие пользователи жалуются на алгоритмы отбора контента в Instagram, которые не всегда отображают интересные для пользователя посты в его ленте. Это может стать возможностью для создания уникального функционала в своей социальной сети.

Анализ пользователей

Для успешной разработки социальной сети необходимо понимать, кто будет ее использовать и какие потребности у них есть. Для этого следует провести анализ пользователей, который поможет определить их интересы, привычки, цели и предпочтения.

Для начала необходимо определить целевую аудиторию социальной сети. Это могут быть, например, молодые люди в возрасте от 18 до 24 лет, люди с определенным уровнем дохода, профессионалы в определенной области или люди с определенными.

Анализ пользователей также поможет определить, какой должен быть дизайн и интерфейс социальной сети. Например, если целевая аудитория в основном состоит из молодых людей, то дизайн должен быть более современным и ярким, а интерфейс должен быть максимально интуитивно понятным и удобным в использовании.

Определение функционала

Одним из наиболее важных аспектов проектирования социальной сети является определение функционала. Функционал должен отвечать потребностям пользователей, быть удобным и понятным. Ниже рассмотрим основные функции, которые должны присутствовать в социальной сети.

1. Регистрация и создание профиля

Каждый пользователь должен иметь возможность зарегистрироваться на сайте и создать свой профиль. Профиль должен содержать базовую информацию о пользователе, такую как имя, фамилия, дата рождения, местоположение и фотографию профиля.

2. Добавление друзей

Одной из основных функций социальной сети является возможность добавлять друзей. Это позволяет пользователям находить старых друзей, находить новых и общаться с ними.

3. Посты и комментарии

Пользователи должны иметь возможность создавать посты и комментировать их. Посты могут содержать текст, фотографии и видео.

Комментарии позволяют пользователям общаться между собой и выражать свое мнение по поводу различных тем.

4. Лайки

Пользователи должны иметь возможность ставить лайки и делать репосты постов, которые им нравятся. Это помогает пользователям находить популярные посты и узнавать, что нравится их друзьям.

Выбор дизайна [26, с. 21]

Дизайн социальной сети является одним из самых важных аспектов ее проектирования. Он влияет на то, насколько пользователи будут удобно использовать платформу, и какие эмоции она вызовет у них. При выборе дизайна социальной сети следует учитывать следующие принципы:

1. Стиль

Важно подобрать стиль, который будет соответствовать теме социальной сети и ожиданиям пользователей. Например, если это профессиональная социальная сеть для бизнеса, то стиль должен быть строгим и корпоративным. Если же это социальная сеть для молодежи, то стиль может быть более ярким и экспрессивным.

2. Цветовая гамма

Цветовая гамма должна быть приятной для глаз пользователей и соответствовать стилю социальной сети. Например, если это социальная сеть для женщин, то можно использовать более яркие цвета, такие как розовый или фиолетовый. Если же это социальная сеть для бизнеса, то лучше использовать более строгие и спокойные цвета, такие как серый или синий.

3. Интерфейс

Интерфейс должен быть интуитивно понятным и простым в использовании. Пользователи не должны тратить много времени на изучение функций социальной сети, чтобы начать ее использовать. Важно, чтобы функции были легко доступны и понятны.

4. Адаптивность

Социальная сеть должна быть адаптивной к различным устройствам, таким как мобильные телефоны, планшеты и компьютеры. Это позволит пользователям использовать платформу в любом месте и в любое время.

Пример: Instagram

Instagram — это социальная сеть, которая известна своим простым и удобным дизайном. Она имеет яркую и приятную для глаз цветовую гамму, а также простой и интуитивно понятный интерфейс. Большое внимание уделяется дизайну профилей пользователей, которые имеют четкую структуру и ясное отображение постов. Более того, Instagram является адаптивной к различным устройствам, что позволяет пользователям использовать ее на мобильных телефонах и планшетах, а также на компьютерах.

1.3 Обзор технологии MERN и ее особенности

MERN [13, с. 20] – это стек технологий, который включает в себя MongoDB, Express.js, React и Node.js. MERN позволяет создавать современные веб-приложения с использованием современных технологий и инструментов.

MongoDB [14] – это документоориентированная база данных, которая позволяет хранить данные в формате JSON-документов. MongoDB [27] обеспечивает высокую производительность и масштабируемость при работе с большими объемами данных. [3. с. 23]

Express.js – это фреймворк для разработки серверных приложений на Node.js. Он обеспечивает удобный доступ к базе данных, обработку запросов и ответов, а также маршрутизацию и обработку ошибок.

React – это библиотека для разработки пользовательских интерфейсов на JavaScript. Она позволяет создавать динамические и интерактивные приложения, которые могут быстро реагировать на действия пользователя. [1, с. 21]

Node.js – это платформа для разработки серверных приложений на JavaScript. Она позволяет использовать JavaScript как язык программирования

для back-end разработки, что упрощает разработку и обеспечивает высокую производительность. [2, с. 10]

Основными преимуществами MERN являются высокая производительность, гибкость и удобство разработки, а также возможность использовать один язык программирования (JavaScript) как для клиентской, так и для серверной части приложения. Кроме того, MERN обеспечивает высокую масштабируемость приложения, что позволяет легко масштабировать его при увеличении количества пользователей или объема данных. [Рис – 1.6]

Для работы с MERN необходимы знания и опыт в разработке на JavaScript, а также в работе с базами данных и фреймворками для back-end разработки. Некоторые из основных инструментов, которые используются при разработке приложений на MERN, включают в себя:

Mongoose – библиотека для работы с MongoDB, которая облегчает взаимодействие с базой данных и упрощает создание моделей данных.

Redux – библиотека для управления состоянием приложения, которая позволяет эффективно управлять данными внутри приложения.

Axios – библиотека для работы с HTTP-запросами, которая позволяет легко отправлять и получать данные с сервера.

Passport – библиотека для аутентификации и авторизации пользователей, которая позволяет создавать безопасные приложения.

Nginx – веб-сервер и обратный прокси-сервер, который обеспечивает высокую производительность и безопасность при работе с приложением.

Существуют также другие стеки технологий для back-end разработки, которые могут быть использованы для создания веб-приложений. Например, MEAN, который включает в себя Angular вместо React [11], или LAMP, который включает в себя Linux, Apache, MySQL и PHP. Выбор стека технологий зависит от конкретных потребностей проекта и предпочтений разработчика.

1.4 Обзор инструментов разработки

Инструменты разработки — это программные средства, которые помогают разработчикам создавать, тестировать, отлаживать и поддерживать программное обеспечение. Это может включать интегрированные среды разработки (IDE), редакторы кода, компиляторы, отладчики, системы контроля версий, средства автоматической сборки и другие программы, которые облегчают процесс разработки и повышают продуктивность разработчиков.

VS Code - Универсальный редактор для создания социальной сети [29]

Visual Studio Code (VS Code) — это бесплатный и мощный редактор кода, разработанный компанией Microsoft. Он предоставляет широкий набор функций, инструментов и расширений, которые облегчают процесс разработки программного обеспечения. В отличие от полноценных интегрированных сред разработки (IDE), таких как Eclipse или Visual Studio, VS Code сконцентрирован на предоставлении минимального и легковесного интерфейса, при этом сохраняя мощные возможности для разработчиков.

VS Code поддерживает широкий спектр языков программирования и платформ, включая JavaScript, Python, C++, Java и многие другие. Он предоставляет множество функций, таких как подсветка синтаксиса, автодополнение, отладка кода, систему контроля версий, интеграцию с системами сборки и многое другое. Эти возможности делают VS Code удобным инструментом для разработчиков, независимо от их предпочтений и потребностей.

Почему выбран VS Code для проекта создания социальной сети?

При выборе редактора для проекта создания социальной сети было принято решение в пользу использования Visual Studio Code. Вот несколько причин, почему этот редактор был выбран:

- Множество расширений: VS Code имеет огромное сообщество разработчиков, которые создают и поддерживают расширения для

различных языков программирования и технологий. Эти расширения значительно упрощают разработку социальной сети, предоставляя инструменты для работы с фронтендом, бэкендом, базами данных и другими компонентами проекта.

- Интеграция с Git: Создание социальной сети требует эффективной работы с системой контроля версий, такой как Git. VS Code обладает мощными функциями интеграции с Git, позволяющими легко отслеживать изменения в коде, создавать и коммитить ветки, сливать изменения и решать конфликты. Это особенно важно при разработке социальной сети, где необходимо управлять и отслеживать множество файлов и изменений.
- Отладка и проверка кода: В процессе разработки социальной сети важно иметь возможность отлаживать код и проверять его на наличие ошибок. VS Code предоставляет инструменты для отладки кода, включая точки останова, просмотр значений переменных и шаги выполнения. Кроме того, с помощью интегрированных инструментов статического анализа и проверки синтаксиса, можно обнаруживать потенциальные проблемы и улучшать качество кода.
- Расширяемость и настраиваемость: VS Code предлагает широкие возможности по настройке редактора под индивидуальные потребности разработчика. Его функциональность можно расширять с помощью расширений, которые позволяют добавлять новые возможности и инструменты. Также можно настроить внешний вид редактора, цветовые схемы, шрифты и другие параметры для комфортной работы.
- Активное сообщество: VS Code является одним из самых популярных редакторов кода и имеет огромное активное сообщество разработчиков. Благодаря этому можно легко найти помощь, решения проблем и обменяться опытом с другими

разработчиками социальных сетей. Множество ресурсов, видеоуроков и расширений созданы сообществом и делают VS Code более доступным и удобным для работы над проектом.

GitHub Desktop - Удобный инструмент для управления версиями в проекте создания социальной сети [30]

GitHub Desktop — это графический интерфейс для работы с Git, распределенной системой управления версиями. Git является широко используемым инструментом для контроля версий и управления изменениями в проектах разработки программного обеспечения. GitHub Desktop облегчает работу с Git, предоставляя простой и интуитивно понятный интерфейс для выполнения распространенных операций Git, таких как создание репозитория, клонирование, коммиты, пуши, пулы и слияния. [4, с. 19]

В проекте создания социальной сети, где разработка происходит в коллективе и вносятся постоянные изменения в код, управление версиями является важным аспектом. GitHub Desktop предоставляет возможность эффективно контролировать изменения, отслеживать внесенные правки и взаимодействовать с другими участниками проекта. Он облегчает совместную работу, предотвращает конфликты слияний и позволяет откатиться к предыдущим версиям кода при необходимости.

Почему выбран GitHub Desktop для проекта создания социальной сети?

При выборе инструмента для управления версиями в проекте создания социальной сети было принято решение в пользу использования GitHub Desktop. Вот несколько причин, почему этот инструмент был выбран:

- Интеграция с GitHub: GitHub является одной из наиболее популярных платформ для хостинга Git-репозиториев и совместной разработки. GitHub Desktop плотно интегрирован с GitHub, что облегчает работу с репозиториями, пул-реквестами и управлением ветками. Это особенно полезно для совместной работы над проектом создания социальной сети, где участники команды могут легко синхронизировать свои изменения,

отслеживать прогресс и комментировать код через интерфейс GitHub.

- Простота использования: GitHub Desktop предоставляет интуитивно понятный и простой в использовании интерфейс, что делает его доступным даже для новичков. Он позволяет выполнять основные операции Git с помощью нескольких кликов, без необходимости запоминать сложные команды и синтаксис. Это особенно полезно в проекте создания социальной сети, где участники команды могут быть неспециалистами в области управления версиями.
- Визуализация изменений: GitHub Desktop предоставляет визуализацию изменений, позволяя увидеть, какие файлы были изменены, добавлены или удалены в проекте. Это упрощает отслеживание изменений и анализ внесенных правок. В проекте создания социальной сети, где существует множество файлов и компонентов, визуализация изменений помогает в понимании того, какие части проекта были модифицированы и как они взаимодействуют между собой.
- Инструменты для совместной работы: GitHub Desktop облегчает совместную работу над проектом создания социальной сети. Он предоставляет функциональность создания веток, выполнения слияний и пул-реквестов, что упрощает совместное внесение изменений и ревью кода. Это позволяет команде эффективно сотрудничать, обсуждать изменения и вносить предложения по улучшению проекта.
- Комментарии и отзывы: GitHub Desktop позволяет пользователям оставлять комментарии и отзывы к изменениям в коде. Это полезный инструмент для обратной связи в проекте создания социальной сети. Участники команды могут комментировать код,

предлагать исправления или задавать вопросы, что способствует обмену знаниями и улучшению проекта.

ГЛАВА 2. РАЗРАБОТКА СОЦИАЛЬНОЙ СЕТИ

2.1 Подготовка окружения разработки

В данной главе будет рассмотрена подробная подготовка окружения разработки для создания социальной сети. Мы ознакомимся с необходимыми инструментами и библиотеками, которые позволят нам эффективно разрабатывать и тестировать наше приложение.

1. Установка и настройка Visual Studio Code (VS Code):
    - Перейдите на официальный сайт VS Code (https://code.visualstudio.com/) и скачайте установочный файл.
    - Запустите установку и следуйте инструкциям.
    - После установки откройте VS Code и установите необходимые расширения, такие как "ESLint" и "Prettier", которые помогут поддерживать код в чистоте и согласованности.
    - Настройте предпочтения редактора в соответствии с вашими предпочтениями, например, выберите тему оформления, установите отступы и т. д.
2. Установка и настройка GitHub Desktop:
    - Перейдите на официальный сайт GitHub Desktop (https://desktop.github.com/) и скачайте установочный файл.
    - Запустите установку и следуйте инструкциям.
    - После установки откройте GitHub Desktop и войдите в свою учетную запись GitHub.
    - Создайте новый репозиторий для вашего проекта и настройте его связь с локальной папкой проекта.
3. Установка необходимых библиотек на сервере:

- Откройте командную строку в корневой папке проекта и выполните команду npm init -y, чтобы инициализировать новый проект Node.js.

Установите следующие библиотеки, выполнив команду npm install bcrypt body-parser cors dotenv express gridfs-stream helmet jsonwebtoken mongoose morgan multer multer-gridfs-storage:

- bcrypt - библиотека для хеширования паролей.
- body-parser - промежуточное ПО для обработки данных запроса в формате JSON.
- cors - промежуточное ПО для обеспечения безопасности и настройки правил доступа к серверу из разных доменов.
- dotenv - библиотека для загрузки переменных среды из файла .env.
- express - фреймворк для создания веб-приложений на Node.js.
- gridfs-stream - модуль для работы с GridFS, специфическим хранилищем файлов в MongoDB.
- helmet - промежуточное ПО для установки различных заголовков безопасности HTTP.
- jsonwebtoken - библиотека для генерации и проверки JSON Web Tokens (JWT).
- mongoose - библиотека для удобной работы с MongoDB, предоставляющая ORM-подобный интерфейс.
- morgan - промежуточное ПО для регистрации HTTP-запросов и ответов в логах сервера.
- multer - библиотека для обработки мультипартных (multipart) форм данных, таких как загрузка файлов.
- multer-gridfs-storage - модуль для сохранения файлов в GridFS с использованием Multer.

4. Установка необходимых библиотек на клиенте:

- Откройте командную строку в папке клиентской части проекта и выполните команду npm init -y, чтобы инициализировать новый проект React.

Установите следующие библиотеки, выполнив команду npm install @emotion/react @emotion/styled @mui/icons-material @mui/material @reduxjs/toolkit dotenv formik react react-dom react-dropzone react-redux react-router-dom react-scripts redux-persist yup:

- @emotion/react и @emotion/styled - библиотеки для работы с эмоциональным CSS в React.
- @mui/icons-material и @mui/material - библиотека компонентов Material-UI для создания пользовательского интерфейса. [16]
- @reduxjs/toolkit - набор инструментов для упрощения работы с Redux.
- dotenv - библиотека для загрузки переменных среды из файла .env.
- formik - библиотека для управления формами в React.
- react и react-dom - библиотеки React для разработки пользовательского интерфейса.
- react-dropzone - компонент для загрузки файлов методом перетаскивания в React.
- react-redux - библиотека для связывания Redux с React.
- react-router-dom - библиотека для реализации маршрутизации в React.
- react-scripts - набор скриптов для разработки и сборки React-приложения.
- redux-persist - библиотека для сохранения состояния Redux в локальном хранилище.
- yup - библиотека для валидации данных в формах.

2.2 Проектирование базы данных

Проектирование данных (Data Design) — это процесс создания структуры и организации данных в базе данных. Оно включает в себя определение сущностей (таблиц), атрибутов (столбцов), связей между сущностями и другие характеристики данных, такие как ограничения целостности. Целью проектирования данных является создание эффективной и надежной базы данных, которая соответствует требованиям и потребностям организации. [8, с. 21]

Моделирование данных (Data Modeling) — это процесс создания абстрактной модели данных, которая описывает структуру данных, их взаимосвязи и правила, регулирующие эти данные. Моделирование данных помогает разработчикам баз данных понять требования к данным и создать подходящую структуру для их хранения и обработки. Одним из распространенных методов моделирования данных является использование сущность-связной модели (Entity-Relationship Model, ER-модель). [10, с. 12]

Свойства данных (Data Properties) — это характеристики данных, которые определяют их качество, точность, целостность, доступность и другие аспекты. Свойства данных включают в себя ограничения целостности, правила проверки, типы данных и другие метаданные, которые определяют, как данные могут быть использованы и обрабатываются в базе данных.

UML (Unified Modeling Language) — это унифицированный язык моделирования, который широко используется для моделирования различных аспектов систем, включая моделирование данных. UML предоставляет нотацию и набор диаграмм для визуального представления структуры и поведения системы. С помощью UML можно создавать диаграммы классов, диаграммы взаимодействия, диаграммы состояний и другие, которые могут быть использованы для моделирования структуры данных и их взаимосвязей. [5, с. 36]

ER-диаграмма (Entity-Relationship diagram) — это тип диаграммы, используемый для моделирования структуры данных в базе данных с помощью ER-модели. ER-модель представляет сущности (entities) в системе, их атрибуты (attributes) и связи (relationships) между сущностями. ER-д - диаграммы помогают визуализировать структуру данных и связи между сущностями. На ER-диаграмме сущности представлены в виде прямоугольников, атрибуты - в виде овалов, а связи между сущностями - в виде стрелок или линий. ER-диаграммы позволяют легко понять структуру базы данных, их взаимосвязи и ограничения целостности данных.

Связь между UML и ER-диаграммами заключается в том, что оба являются инструментами для визуального моделирования системы. UML общеупотребимый язык моделирования, который может быть использован для моделирования различных аспектов систем, включая моделирование данных. В рамках UML можно использовать диаграммы классов, чтобы моделировать сущности и их атрибуты, а также диаграммы взаимодействия для представления связей между объектами или сущностями. ER-диаграммы специализированы на моделировании структуры данных и связей между сущностями, они используются при проектировании баз данных.

В данном проекте всего 4 таблицы из них 2 таблицы: Post и User являются основными, а Friend и Images дополнительными. Friend в данном случае это динамически изменяемый объект, а Images это хранилище для изображений таких как фото профиля или фото поста. [Рис – 2.1]

Таблица 2.1

Моделирование таблицы постов

Post	
id	String
userId	String Ref
firstName	String
lastName	String

Продолжение таблицы 2.1

location	String
description	String
userPicurePath	String Ref
PicturePath	String Ref
likes	Object<String Ref>
comments	Array<String>

Таблица 2.2

Моделирование таблицы пользователей

User	
id	String
firstName	String
lastName	String
friends	Array<Object>
email	String
password	String
picturePath	String Ref
location	String
occupation	String

Таблица 2.3

Моделирование таблицы друзей

Friend (Sub Docs)	
id	String
firstName	String
lastName	String
picturePath	String Ref
occupation	String
location	String

2.3 Реализация серверной части

2.3.1 Создание и настройка сервера на Node.js

В данной главе мы создадим и настроим сервер на Node.js [15] для разработки нашей социальной сети. Мы будем использовать фреймворк Express.js, базу данных MongoDB [14] с помощью Mongoose, а также несколько дополнительных пакетов для обработки запросов, загрузки файлов и обеспечения безопасности.

Шаг 1: Установка зависимостей

Откройте командную строку и выполните следующую команду: npm install express body-parser mongoose cors dotenv multer helmet morgan path

Шаг 2: Подготовка файловой структуры

Перед тем, как начать разрабатывать сервер, создадим необходимую файловую структуру. [Рис – 2.2]

Шаг 3: Импортирование библиотек и модулей [Рис – 2.3]

- import express from "express"; - Мы импортируем модуль express, который позволяет нам создавать сервер.
- import bodyParser from "body-parser"; - Модуль body-parser используется для разбора тела запросов.
- import mongoose from "mongoose"; - Мы используем модуль mongoose для работы с MongoDB.
- import cors from "cors"; - Модуль cors позволяет настроить политику CORS (Cross-Origin Resource Sharing) для нашего сервера.
- import dotenv from "dotenv"; - Модуль dotenv позволяет нам загрузить переменные окружения из файла .env.
- import multer from "multer"; - Модуль multer используется для загрузки файлов на сервер.

- import helmet from "helmet"; - Модуль helmet помогает защитить наш сервер от некоторых уязвимостей с помощью различных HTTP-заголовков.
- import morgan from "morgan"; - Модуль morgan используется для логирования HTTP-запросов.
- import path from "path"; - Модуль path предоставляет утилиты для работы с путями к файлам и директориям.
- import { fileURLToPath } from "url"; - Модуль url предоставляет утилиты для работы с URL.
- import authRoutes from "./routes/auth.js"; - Мы импортируем маршруты, связанные с аутентификацией.
- import userRoutes from "./routes/users.js"; - Мы импортируем маршруты, связанные с пользователями.
- import postRoutes from "./routes/posts.js"; - Мы импортируем маршруты, связанные с постами.
- import { register } from "./controllers/auth.js"; - Мы импортируем контроллер для регистрации пользователей.
- import { createPost } from "./controllers/posts.js"; - Мы импортируем контроллер для создания постов.
- import { verifyToken } from "./middleware/auth.js"; - Мы импортируем промежуточное ПО для проверки токена аутентификации.

Шаг 4: Настройка сервера

Теперь откроем файл index.js и начнем настраивать сервер.

Конфигурация сервера [Рис – 2.4]

- Загружаем переменные.env с помощью dotenv.config().
- Создаем приложения Express с помощью app = express().
- Используем express.json() для разбора JSON-тела запросов.
- Применяем промежуточное ПО helmet()

- Устанавливаем политику CORS в заголовках.
- Используем morgan("common") для записи логов запросов.
- Разрешаем передачу JSON с максимальным размером 30 МБ.
- Разрешаем передачу данных URL с длиной размером 30 МБ.
- Разрешаем кросс-доменные запросы с помощью cors().
- Устанавливаем путь для статических файлов.

Хранилище

В этом коде мы настраиваем хранение файлов на сервере с использованием multer для обработки загрузки файлов. Определяется папка (destination), куда будут сохраняться загруженные файлы. файлы. [Рис – 2.5]

Маршрутизация [Рис – 2.6]

Маршрут "/auth/register":

- Метод: POST
- Путь: "/auth/register"
- Функция загрузки файла: upload.single("picture")
- Функция обработки регистрации пользователя: register()

Маршрут "/posts":

- Метод: POST
- Путь: "/posts"
- Функция проверки токена аутентификации: verifyToken
- Функция загрузки файла: upload.single("picture")
- Функция создания нового поста: createPost()

Подключаем маршруты

Мы подключаем маршруты, определенные в отдельных файлах, следующим образом: [Рис – 2.7]

1. Маршруты для пути "/auth":

- Путь: "/auth"
- Файл маршрутов: "auth.js"
- Код: app.use("/auth", authRoutes)

2. Маршруты для пути "/users":

- Путь: "/users"
- Файл маршрутов: "users.js"
- Код: app.use("/users", userRoutes)

3. Маршруты для пути "/posts":

- Путь: "/posts"
- Файл маршрутов: "posts.js"
- Код: app.use("/posts", postRoutes)

Настройка базы данных [Рис – 2.8]

Настраиваем подключение к MongoDB с использованием Mongoose:

1. Устанавливаем порт сервера:

- Переменная: PORT
- Значение: process.env.PORT || 6001
- Если переменная не задана, используется порт по 6001.

2. Устанавливаем соединение с базой данных MongoDB:

- Метод: mongoose.connect()
- Параметры: process.env.MONGO_URL, { useNewUrlParser: true, useUnifiedTopology: true }
- Используемый URL берется из process.env.MONGO_URL.
- Передаем опции для использования парсера URL.

3. Запускаем сервер:

- Метод: app.listen(PORT)
- Сервер начинает прослушивать входящие запросы.
- Если подключение прошло успешно, выводится сообщение.
- В случае ошибки при подключении выводится сообщение.

2.3.2 Создание моделей для базы данных

Описание модели пользователей [Рис – 2.9]

Давайте разберем каждое поле и его свойства, используемые в модели:

1. firstName (имя пользователя):
   - Тип: String (строка).
   - Обязательное поле (required: true).
   - Минимальная длина имени: 2 символа (min: 2).
   - Максимальная длина имени: 50 символов (max: 50).
2. lastName (фамилия пользователя):
   - Тип: String (строка).
   - Обязательное поле (required: true).
   - Минимальная длина фамилии: 2 символа (min: 2).
   - Максимальная длина фамилии: 50 символов (max: 50).
3. email (электронная почта пользователя):
   - Тип: String (строка).
   - Обязательное поле (required: true).
   - Максимальная длина электронной почты: 50 символов (max: 50).
   - Уникальное значение (unique: true), чтобы каждый пользователь имел уникальный адрес электронной почты.
4. password (пароль пользователя):
   - Тип: String (строка).
   - Обязательное поле (required: true).
   - Минимальная длина пароля: 5 символов (min: 5).
5. picturePath (путь к изображению профиля пользователя):
   - Тип: String (строка).
   - Значение по умолчанию: "" (пустая строка).
   - Позволяет хранить путь к изображению профиля пользователя.
6. friends (список друзей пользователя):
   - Тип: Array (массив).
   - Значение по умолчанию: [] (пустой массив).
   - Позволяет хранить список друзей пользователя.

7. location (местоположение пользователя):

- Тип: String (строка).
- Позволяет хранить информацию о местоположении пользователя.

8 occupation (занятие пользователя):

- Тип: String (строка).
- Позволяет хранить информацию о занятии пользователя, таком как профессия, род деятельности и т. д.

Помимо полей, также определили дополнительные опции для модели:

- { timestamps: true } — это опция, которая добавляет два поля к каждому документу в коллекции: createdAt (дата и время создания документа) и updatedAt (дата и время последнего обновления документа). Эти поля автоматически обновляются при создании и обновлении документа, соответственно.

Описание модели постов [Рис – 2.10]

Описание модели постов включает следующие поля:

1. userId (идентификатор пользователя):

- Тип: String (строка).
- Обязательное поле, которое содержит идентификатор пользователя, создавшего пост. Используется для связи с моделью пользователей.

2. firstName (имя пользователя):

- Тип: String (строка).
- Обязательное поле, которое содержит имя пользователя, создавшего пост.

3. lastName (фамилия пользователя):

- Тип: String (строка).
- Обязательное поле, которое содержит фамилию пользователя, создавшего пост.

4. location (местоположение):

- Тип: String (строка).
- Поле, которое содержит информацию о местоположении, связанное с постом. Например, место, где был сделан пост.

5. description (описание):
    - Тип: String (строка).
    - Поле, которое содержит описание поста. Здесь пользователь может оставить текстовый комментарий, описывающий содержимое поста.
6. picturePath (путь к изображению):
    - Тип: String (строка).
    - Поле, которое содержит путь к изображению, связанному с постом. Здесь может храниться ссылка или путь к файлу с изображением.
7. userPicturePath (путь к изображению пользователя):
    - Тип: String (строка).
    - Поле, которое содержит путь к изображению профиля пользователя, создавшего пост. Здесь может храниться ссылка или путь к файлу с изображением.
8. likes (лайки):
    - Тип: Map (отображение).
    - Поле, которое содержит информацию о лайках, полученных постом. Здесь используется Map, где ключом является идентификатор пользователя, а значением - логическое значение (true/false), указывающее на наличие или отсутствие лайка от данного пользователя.
9. comments (комментарии):
    - Тип: Array (массив).
    - Поле, которое содержит массив комментариев, связанных с постом. Каждый элемент массива представляет отдельный

комментарий, который может содержать текст, информацию об авторе комментария и дату создания.

Дополнительные свойства модели:

- { timestamps: true }: это опция схемы, которая добавляет поля createdAt и updatedAt для автоматического отслеживания времени создания и последнего обновления поста. Эти поля автоматически обновляются при сохранении и обновлении документа в базе данных.
- const Post = mongoose.model("Post", postSchema): здесь создается модель "Post" на основе определенной схемы postSchema. Модель позволяет взаимодействовать с коллекцией постов в базе данных MongoDB.
- export default Post: Модель постов экспортируется для использования в других частях приложения.

2.3.3 Создание API для регистрации и авторизации пользователей

В данной главе будет представлено создание API [19, с. 37] для регистрации и авторизации пользователей в социальной сети. Мы будем использовать следующий код Рис в качестве основы:

Регистрация пользователя [Рис – 2.11]

Функция register отвечает за регистрацию нового пользователя в социальной сети. При вызове функции происходят следующие действия:

- Получение данных пользователя из запроса (req.body), включая firstName, lastName, email, password, picturePath, friends, location, occupation.
- Разбор полученных данных.
- Генерация соли (salt) с помощью функции bcrypt.genSalt().
- Хеш пароля с использованием salt и функции bcrypt.hash).

- Создание нового экземпляра модели User
- Сохранение нового пользователя в базе данных
- Возврат статуса ответа 201 (Created) и JSON-объекта
- Если происходит ошибка, то возвращается статус ответа 500.

Авторизация пользователя [Рис – 2.12]

Функция login отвечает за аутентификаци. При вызове функции происходят следующие действия:

- Получение данных пользователя (email и password)
- Извлечение значений email и password из полученных данных.
- Поиск пользователя в базе данных
- Сравнение хеша пароля в базе данных, с введенным паролем
- Если пароль совпадает, создается JWT.
- Возвращается статус ответа 200.
- Если ошибка, возвращается статус ответа 500

2.3.4 Создание API для работы с постами

В данной главе мы создадим API [20, с. 29] для работы с постами в социальной сети. API [22] будет предоставлять функции создания, чтения, обновления и удаления постов, а также управления комментариями.

Импортируем модели Post и User.

Метод createPost: [Рис – 2.13]

- Создает новый пост на основе переданных данных.
- Сохраняет новый пост в базе данных.

Метод getFeedPosts: [Рис – 2.14]

- Возвращает список всех постов в ленте.

Метод getUserPosts: [Рис – 2.15]

- Возвращает список постов, опубликованных определенным пользователем.

Метод likePost: [Рис – 2.16]

- Обновляет информацию о посте, добавляя или удаляя лайк.
- Принимает идентификатор поста и идентификатор пользователя.

Метод postComment: [Рис – 2.17]

- Добавляет комментарий к посту.
- Принимает id поста, id пользователя и текст комментария.

Метод deleteComment: [Рис – 2.18]

- Удаляет комментарий к посту.
- Принимает id поста и текст комментария.

2.3.5 Создание API для работы с друзьями

В данной главе мы рассмотрим создание API [21, с. 21] для работы с друзьями в социальной сети, используя предоставленный код. Код содержит методы для чтения и обновления данных о пользователях и их друзьях.

Метод getUser: [Рис – 2.19]

- Получает информацию о пользователе по его идентификатору.
- Принимает запрос req и ответ res в качестве параметров.
- Использует модель User и метод findById.
- Если пользователь найден, отправляет данные с кодом 200.
- Если пользователь не найден, отправляет с код состояния 404.

Метод getUserFriends: [Рис – 2.20]

- Получает список друзей пользователя по его идентификатору.
- Принимает запрос req и ответ res.
- Находит пользователя по идентификатору с помощью метода findById.
- Использует метод map для создания массива промисов.
- Перебирает массив id друзей пользователя.
- Получает массив объектов friends.

- Форматирует данные.
- Отправляет массив друзей в формате JSON с кодом состояния 200.
- В случае ошибки отправляет с кодом состояния 404.

Метод addRemoveFriend: [Рис – 2.21]

- Добавляет или удаляет друга у пользователя.
- Получает идентификатор пользователя и идентификатор друга из параметров запроса.
- Находит объекты пользователя и друга.
- Проверяет, содержит ли пользователь id друга.
- Если содержит, удаляет id из массивов user.friends и friend.friends.
- Если не содержит, добавляет идентификаторы друг друга.
- Сохраняет данные пользователей с помощью метода save().
- Создает массив промисов для поиска данных о друзьях.
- Отправляет массив друзей в формате JSON с кодом состояния 200.
- В случае ошибки отправляет сообщение с кодом состояния 404.

2.3.6 ПО для проверки токена Аутентификации

В этом разделе мы рассмотрим реализацию ПО для проверки токена аутентификации в социальной сети. [Рис – 2.22]

Код выполняет следующие шаги:

Шаг 1: Получение токена из заголовка запроса

При выполнении запроса клиент должен дать токен аутентификации.

Шаг 2: Проверка формата токена и его обработка

Мы проверяем формат полученного токена, который должен быть в формате "Bearer <token_value>".

Шаг 3: Проверка и верификация токена

Мы используем библиотеку jsonwebtoken для проверки и верификации токена.

Шаг 4: Сохранение

После успешной проверки и верификации токена, мы сохраняем информацию о пользователе.

Затем мы передаем управление следующему обработчику, вызывая функцию "next()".

Шаг 5: Обработка ошибок

Если возникает ошибка при проверке или верификации токена, мы перехватываем исключение и отправляем ответ с кодом состояния 500 и объектом JSON, содержащим сообщение об ошибке.

2.4 Реализация клиентской части

2.4.1 Регистрация и Авторизация

Реализация клиентской части [17, с. 16] относится к процессу создания и разработки программного кода, который выполняется на стороне клиента, то есть на устройстве пользователя, таком как веб-браузер или мобильное приложение. Клиентская часть обычно отвечает за представление и взаимодействие пользователя с приложением.

Регистрация пользователя является процессом создания учетной записи или профиля на платформе, системе или сервисе. В данной главе мы рассмотрим реализацию аутентификации и авторизации в социальной сети. Для этого мы будем использовать формы регистрации и авторизации, а также проводить проверку и валидацию данных как на стороне клиента, так и на стороне сервера.

Регистрация [Рис – 2.23]

1. Форма регистрации: Мы начнем с создания формы регистрации, которая содержит необходимые поля, такие как имя пользователя, электронная почта, пароль и другие данные.

2. Валидация формы регистрации: Мы определили схему валидации "registerSchema", которая устанавливает правила для каждого поля. Например, поле "email" должно быть валидным адресом электронной почты, а поля

"firstName", "lastName", "password", "location" и "occupation" должны быть заполнены. Мы также задали начальные значения полей формы в объекте "initialValuesRegister".

3. Обработка формы регистрации: для обработки данных, отправляемых с формы регистрации, мы создаем объект FormData, в котором добавляем значения полей формы, включая изображение. Затем мы отправляем этот объект на сервер методом POST по адресу "http://localhost:3001/auth/register". После получения ответа от сервера в формате JSON, мы сбрасываем форму и переходим на страницу авторизации, если регистрация прошла успешно.

Авторизация [Рис – 2.24]

1. Форма авторизации: Мы создаем форму авторизации, которая содержит поля для ввода имени пользователя/электронной почты и пароля. Аналогично форме регистрации, мы определили схему валидации "loginSchema" и начальные значения полей в объекте "initialValuesLogin".

2. Обработка формы авторизации: для обработки данных, отправляемых с формы авторизации, мы отправляем данные на сервер методом POST по адресу "http://localhost:3001/auth/login". Мы устанавливаем заголовок "Content-Type" для указания типа данных (JSON). После получения ответа от сервера в формате JSON, мы сбрасываем форму, сохраняем информацию о пользователе и токене в хранилище Redux с помощью функции dispatch, а затем переходим на страницу Home.

2.4.2 Главная страница

На главной странице требуется отобразить ленту новостей, состоящую из постов пользователей. В этой главе мы рассмотрим получение данных о постах с сервера и их отображение на странице. [Рис – 2.25]

Получение данных о постах

Для получения списка постов мы используем две функции: getPosts и getUserPosts. Функция getPosts отправляет запрос на сервер для получения

всех постов, а функция getUserPosts отправляет запрос для получения постов, принадлежащих определенному пользователю. Обе функции вызываются при загрузке компонента PostsWidget с использованием хука useEffect.

Функции getPosts и getUserPosts используют fetch для отправки GET-запроса на сервер. Мы указываем URL-адрес сервера и добавляем заголовок Authorization с токеном пользователя для аутентификации. После получения ответа от сервера в формате JSON, мы используем функцию dispatch для сохранения полученных данных в хранилище Redux с помощью действия setPosts.

Отображение постов

Мы будем разрабатывать компонент "PostWidget", который будет отображать посты на странице нашей социальной сети. Мы будем рассматривать разметку и стилизацию постов, а также включение фотографий и текста в каждый пост.

Начнем с импорта необходимых компонентов и хуков. Далее определим компонент "PostWidget" и передадим ему необходимые пропсы. Внутри компонента "PostWidget" определим необходимые состояния и получим данные из хранилища Redux. Далее, определим функцию "patchLike", которая будет выполнять запрос на сервер для обновления данных о лайках поста. Затем определим функцию "handleComment", которая будет выполнять запрос на сервер для добавления комментария к посту.

Продолжим с разметки и стилизации компонента "PostWidget":

В данной разметке мы используем компоненты и стили из библиотеки Material-UI для создания интерфейса поста.

- WidgetWrapper - обертка для виджета поста.
- Friend - компонент, отображающий информацию о пользователе, размещающем пост (имя, фотография, местоположение).
- Typography - компонент для отображения описания поста.
- img - отображение фотографии поста, если она доступна.

- FlexBetween - контейнер, обеспечивающий выравнивание элементов по горизонтали и расстояние между ними.

Внутри компонента FlexBetween у нас две группы элементов:

Первая группа содержит иконку лайка (FavoriteOutlined или FavoriteBorderOutlined) и количество лайков (likeCount).

Вторая группа содержит иконку комментария (ChatBubbleOutlineOutlined) и количество комментариев (comments.length).

Если isComments равно true, то отображается блок с комментариями:

- Мы отображаем комментарии в обратном порядке (последний комментарий отображается первым) с помощью метода reverse().
- Для каждого комментария создается компонент Comment, который отображает информацию о пользователе, оставившем комментарий, и текст комментария.
- Между комментариями мы добавляем горизонтальный разделитель (Divider).
- В конце блока комментариев мы также добавляем горизонтальный разделитель.
- Внизу блока комментариев находится контейнер FlexBetween, который содержит поле ввода для написания комментария (InputBase) и кнопку "POST" (Button), которая выполняет функцию handleComment при клике.

Таким образом, компонент "PostWidget" отображает информацию о посте, включая имя пользователя, описание, фотографию (если есть), количество лайков и комментариев. При клике на иконку лайка происходит обновление данных о лайках поста, а при клике на иконку комментария показываются комментарии к посту. Компонент также позволяет пользователю добавлять комментарии к посту через поле ввода и кнопку "POST".

На главной странице социальной сети требуется отобразить ленту новостей, состоящую из постов пользователей. В этой главе мы рассмотрим получение данных о постах с сервера и их отображение на странице.

Получение данных о постах

Для получения списка постов мы используем две функции: getPosts и getUserPosts. Функция getPosts отправляет запрос на сервер для получения всех постов, а функция getUserPosts отправляет запрос для получения постов, принадлежащих определенному пользователю. Обе функции вызываются при загрузке компонента PostsWidget с использованием хука useEffect.

Функции getPosts и getUserPosts используют fetch для отправки GET-запроса на сервер. Мы указываем URL-адрес сервера и добавляем заголовок Authorization с токеном пользователя для аутентификации. После получения ответа от сервера в формате JSON, мы используем функцию dispatch для сохранения полученных данных в хранилище Redux с помощью действия setPosts.

Отображение постов

Мы будем разрабатывать компонент "PostWidget", который будет отображать посты на странице нашей социальной сети. Мы будем рассматривать разметку и стилизацию постов, а также включение фотографий и текста в каждый пост.

Начнем с импорта необходимых компонентов и хуков. Далее определим компонент "PostWidget" и передадим ему необходимые пропсы. Внутри компонента "PostWidget" определим необходимые состояния и получим данные из хранилища Redux. Далее, определим функцию "patchLike", которая будет выполнять запрос на сервер для обновления данных о лайках поста. Затем определим функцию "handleComment", которая будет выполнять запрос на сервер для добавления комментария к посту. Продолжим с разметки и стилизации компонента "PostWidget":

В данной разметке мы используем компоненты и стили из библиотеки Material-UI для создания интерфейса поста.

- WidgetWrapper - обертка для виджета поста.
- Friend - компонент, отображающий информацию о пользователе, размещающем пост (имя, фотография, местоположение).
- Typography - компонент для отображения описания поста.
- img - отображение фотографии поста, если она доступна.
- FlexBetween - контейнер, обеспечивающий выравнивание элементов по горизонтали и расстояние между ними.

Внутри компонента FlexBetween у нас две группы элемент ов:

- Лайки и комментарии. Для лайков мы используем компонент IconButton из Material-UI, который позволяет создать иконку с возможностью клика. Для отображения количества лайков мы используем компонент Typography. Для комментариев мы также используем компонент IconButton и компонент Typography.
- Дата и время публикации поста. Для отображения даты и времени мы используем компонент Typography.

Наконец, определим функцию renderPost, которая будет отображать каждый пост на странице. Функция получает объект поста в качестве аргумента и возвращает разметку компонента "PostWidget" с данными поста.

В итоге мы получили компонент "PostWidget", который может отображать посты пользователей на странице нашей социальной сети. Компонент использует данные из хранилища Redux и обновляет их при необходимости с помощью запросов к серверу. Мы также разработали разметку и стилизацию компонента, используя компоненты и стили из библиотеки Material-UI.

Создание формы для нового поста с возможностью загрузки фотографий

Для начала нам нужна форма, которая позволит пользователям заполнить информацию о своих постах и, при необходимости, загрузить фотографии. Рассмотрим, как это можно сделать.

Компонент PostForm содержит текстовое поле для описания поста и поле ввода типа "file" для загрузки фотографии. При изменении значения в поле описания мы обновляем состояние переменной description с помощью функции setDescription. Аналогично, при выборе фотографии мы обновляем состояние переменной picture с помощью функции setPicture. Когда пользователь нажимает кнопку "Опубликовать", вызывается обработчик handleSubmit.

Отправка данных на сервер для создания нового поста

Теперь рассмотрим, как отправить данные на сервер для создания нового поста, используя форму, которую мы только что создали. После заполнения формы с описанием поста и выбора фотографии мы можем отправить эти данные на сервер для создания нового поста.

Добавляем обработчик события handleSubmit, который вызывается при нажатии кнопки "Опубликовать". Внутри обработчика мы создаем экземпляр FormData, добавляем описание поста и выбранную фотографию. Затем мы используем функцию fetch для отправки данных на сервер.

Важно отметить, что мы указываем метод "POST" и URL ${process.env.REACT_APP_SERVER_URL}/posts для создания нового поста на сервере. Мы также включаем заголовок Authorization с токеном доступа пользователя, чтобы сервер мог аутентифицировать запрос.

После отправки запроса мы проверяем статус ответа. Если статус response.ok, значит пост был успешно создан на сервере. Вы можете добавить дополнительные действия в этой ветви кода, если это необходимо. Если при отправке запроса возникла ошибка или статус ответа не response.ok, мы можем обработать ошибку или вывести сообщение об ошибке в консоль для отладки. Мы добавили вывод сообщений в консоль в случае успешного создания поста или ошибки. Вы можете заменить console.log и console.error на соответствующие действия, необходимые в вашем приложении.

2.4.3 Список друзей

Отображение списка друзей:

Мы рассмотрим процесс отображения списка друзей пользователя на странице. Для этого мы используем запрос к серверу для получения данных о друзьях и их обработку на стороне клиента. [Рис – 2.26]

Запрос к серверу для получения списка друзей пользователя:

Для получения списка друзей пользователя мы отправляем GET-запрос на сервер с помощью функции getFriends. Мы используем функцию fetch, передавая ей URL сервера и данные аутентификации в заголовке запроса. URL запроса формируется с использованием переменной окружения REACT_APP_SERVER_URL и идентификатора текущего пользователя.

Обработка данных на стороне клиента:

Полученные данные обрабатываются на стороне клиента. Мы используем функцию dispatch из хранилища Redux для обновления состояния приложения и сохранения списка друзей. Функция setFriends является action creator в Redux, создающим action с типом "SET_FRIENDS" и данными о друзьях.

Отображение друзей:

После получения списка друзей мы отображаем их на странице, стилизуя каждого друга и включая информацию о нем.

Разметка и стилизация списка друзей на странице:

Для создания разметки списка друзей на странице мы используем компоненты Material-UI, такие как Box и Typography. Компонент WidgetWrapper используется для обертки списка друзей и добавления стилей виджета. Мы добавляем заголовок "Friend List" с помощью компонента Typography и создаем контейнер с помощью компонента Box.

Включение информации о каждом друге (фото, имя пользователя и т. д.):

Для каждого друга в списке мы создаем компонент Friend, передавая ему необходимую информацию, такую как фото, имя пользователя и

дополнительные данные. Мы используем функцию map для прохода по каждому другу и создания компонента Friend с нужными данными.

Таким образом, мы получаем список друзей пользователя, отображаем его на странице с помощью компонента FriendListWidget. Вы можете настроить стили и дизайн компонента Friend в соответствии с вашими потребностями и дизайном приложения. Этот код поможет вам создать социальную сеть и отобразить список друзей на странице.

Взаимодействие с друзьями:

Добавление в друзья:

Добавление в друзья:

Пользователи нашей социальной сети могут добавлять друг друга в друзья, чтобы установить связь и следить за активностью.

1.1. Добавление через кнопку или поиск:

Мы предоставляем два варианта для добавления друга: через кнопку на профиле или через поиск.

1.2. Отправка запроса на сервер:

При добавлении друга мы отправляем PATCH запрос на сервер для выполнения соответствующего действия.

1.3. Обновление списка друзей:

После подтверждения запроса и успешного добавления друга, обновляем список друзей текущего пользователя.

Удаление из друзей:

2. Удаление из друзей:

Пользователи могут удалить друга из списка друзей, разрывая связь между собой.

2.1. Удаление из списка:

Мы предоставляем возможность удалить друга из списка друзей.

2.2. Отправка запроса на сервер:

При удалении друга отправляем PATCH запрос на сервер для удаления.

2.3. Обновление списка друзей:

После успешного удаления друга обновляем список друзей.

2.4.4 Настройки темы

В этой главе мы рассмотрим настройку темы и цветовой схемы для выпускной работы по созданию социальной сети. Пользователям будет предоставлен список доступных тем и цветовых схем, а также разобран соответствующий код для настройки темы.

При создании социальной сети важно предоставить пользователям возможность выбрать удобную или приятную им тему или цветовую схему. Мы разработали список доступных вариантов для использования в нашей социальной сети.

Список доступных тем/цветовых схем: Темная тема: темный фон и яркие акцентные цвета. [Рис – 2.27] Светлая тема: светлый фон и нейтральные цвета. [Рис – 2.28] Пользователи смогут выбрать одну из этих тем/цветовых схем в настройках профиля или на главной странице социальной сети.

Код настройки темы:

Функция themeSettings принимает аргумент mode, который определяет выбранную тему (светлую или темную). В зависимости от выбранной темы функция возвращает объект с настройками темы.

Внутри объекта palette определяются цветовые значения для различных элементов интерфейса. В зависимости от выбранной темы (mode) применяются соответствующие палитры.

Объект primary содержит основные цвета для акцентных элементов, включая темный, основной и светлый оттенки.

Объект neutral содержит значения для нейтральных цветов, включая темные, основной, средний основной, средний и светлый оттенки.

Объект background определяет цветовые значения для фона, включая стандартный и альтернативный фон.

Объект typography определяет настройки шрифтов, включая семейство шрифтов и базовый размер.

Когда пользователь выбирает определенную тему или цветовую схему, функция themeSettings вызывается, возвращая объект с соответствующими настройками для выбранной темы.

Теперь у вас есть готовая глава "Настройки темы" для вашей выпускной работы по созданию социальной сети, включающая список доступных тем/цветовых схем и разбор кода для настройки темы.

ЗАКЛЮЧЕНИЕ

В данной главе будет представлено заключение дипломной работы на тему "Создание социальной сети". Будут описаны основные результаты исследования, оценена эффективность социальной сети, предложены рекомендации по ее улучшению, сделаны выводы по исследованию, рассмотрено практическое применение результатов исследования, а также представлено резюме работы.

Основные результаты исследования

В ходе выполнения работы была разработана и реализована социальная сеть с базовым функционалом, включающим регистрацию и авторизацию пользователей, создание постов (фото + текст), отметки "Нравится", комментарии, добавление и удаление друзей, а также возможность изменения цветовой темы сайта. Были учтены основные требования и потребности пользователей социальных сетей.

Оценка эффективности социальной сети

Проведена оценка эффективности разработанной социальной сети на основе следующих критериев:

Удобство использования: социальная сеть предоставляет интуитивно понятный интерфейс и удобные функции, что способствует удовлетворению пользовательских потребностей.

Функциональность: разработанный функционал социальной сети позволяет пользователям выполнять основные действия, такие как создание постов, ставка "Нравится", комментирование, управление друзьями и изменение цветовой темы.

Производительность: социальная сеть обеспечивает достаточную скорость работы и отзывчивость интерфейса, обрабатывая запросы пользователей без заметных задержек.

Рекомендации по улучшению социальной сети

На основе проведенной оценки эффективности исследования, предлагаются следующие рекомендации по улучшению социальной сети:

Расширение функционала: добавить возможность загрузки видео, создания групп, отправки сообщений между пользователями и другие расширенные функции, чтобы обеспечить более полный и разнообразный пользовательский опыт.

Улучшение интерфейса: провести дизайн-исследование для оптимизации пользователь ского интерфейса, улучшения навигации и общей визуальной привлекательности социальной сети. Внимание следует уделять читабельности текста, расположению элементов, цветовой схеме и общей эстетике пользовательского интерфейса.

Усиление безопасности: реализовать дополнительные меры безопасности, такие как проверка подлинности изображений и контента, фильтрация нежелательных комментариев и предотвращение возможности злоупотребления функционалом социальной сети.

Улучшение производительности: провести оптимизацию кода, базы данных и серверной инфраструктуры для обеспечения быстрой и надежной работы социальной сети даже при большом количестве пользователей и активности.

Выводы по исследованию

В результате исследования и разработки социальной сети можно сделать следующие выводы:

Социальная сеть предоставляет базовый функционал, необходимый для общения и взаимодействия пользователей. В ходе работы были реализованы основные возможности, такие как создание постов, ставка "Нравится", комментирование, управление друзьями и изменение цветовой темы сайта.

Оценка эффективности социальной сети показала, что она обладает удобным интерфейсом, функциональностью и достаточной производительностью, что способствует удовлетворению пользовательских потребностей.

Рекомендации по улучшению социальной сети включают расширение функционала, улучшение интерфейса и безопасности, а также оптимизацию производительности.

Практическое применение результатов исследования

Разработанная социальная сеть может быть использована в реальных условиях для обеспечения коммуникации и обмена информацией между пользователями. Она может быть применена как внутренняя социальная сеть для организаций или сообществ, так и в качестве платформы для общения между людьми с общими интересами или целями.

Резюме работы

В дипломной работе была успешно разработана и реализована социальная сеть с базовым функционалом, позволяющим пользователям создавать посты, ставить отметки "Нравится", писать комментарии, добавлять и удалять друзей, а также изменять цветовую тему сайта. Был проведен анализ эффективности социальной сети, который показал, что она обладает удобным интерфейсом, функциональностью и достаточной производительностью, удовлетворяя потребности пользователей.

Одним из ключевых результатов исследования является разработка функциональности, позволяющей пользователям создавать и делиться постами в виде фотографий с текстом, а также взаимодействовать с контентом других пользователей путем ставки "Нравится" и комментирования. Кроме того, реализована возможность добавления и удаления друзей, что способствует расширению социальных связей между пользователями.

Важным аспектом работы является возможность изменения цветовой темы сайта. Эта функция позволяет пользователям настроить внешний вид социальной сети в соответствии с их предпочтениями, создавая более комфортную и индивидуальную пользовательскую среду.

Оценка эффективности социальной сети показала, что разработанный функционал соответствует ожиданиям пользователей. Удобный интерфейс,

понятная навигация и возможность взаимодействия с другими пользователями способствуют позитивному пользовательскому опыту.

Однако, существуют аспекты, требующие улучшения. В частности, рекомендуется расширить функционал социальной сети путем добавления возможности загрузки видео, создания групп и отправки сообщений между пользователями. Такие дополнительные функции позволят пользователям более глубоко взаимодействовать и обмениваться контентом.

Для дальнейшего улучшения социальной сети также рекомендуется провести дизайн-исследование с целью оптимизации пользовательского интерфейса. Важно обратить внимание на читабельность текста, удобство расположения элементов и привлекательность визуального оформления. Такие улучшения помогут привлечь больше пользователей и повысить удовлетворенность ими.

Также следует обратить внимание на безопасность социальной сети. Рекомендуется внедрить дополнительные меры

безопасности, чтобы защитить пользователей от возможных угроз и злоупотреблений. Это может включать проверку подлинности изображений и контента, фильтрацию нежелательных комментариев, а также механизмы предотвращения злоумышленников от злоупотребления функционалом социальной сети.

Для обеспечения высокой производительности социальной сети рекомендуется провести оптимизацию кода, базы данных и серверной инфраструктуры. Это поможет обеспечить быструю и надежную работу даже при большом количестве пользователей и активности на платформе.

Разработанная социальная сеть имеет практическое применение как внутриорганизационная платформа для общения и взаимодействия между сотрудниками, так и в качестве публичной платформы для обмена информацией и социального взаимодействия между людьми с общими интересами. Она может быть использована в различных сферах, таких как бизнес, образование, развлечения и многих других.

В заключение дипломная работа по созданию социальной сети была успешно выполнена. Результаты исследования показали, что разработанная социальная сеть обладает удобным интерфейсом, функциональностью и достаточной производительностью, удовлетворяя потребности пользователей. Однако, с целью улучшения социальной сети, рекомендуется расширить функционал, улучшить интерфейс и безопасность, а также оптимизировать производительность.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Стоян Стефанов: React.js. Быстрый старт: Издательство: Питер, 2017 г. 304 с.

2. Кантелон, Янг, Мек: Node.js в действии: Издательство: Питер, 2018 г. 432 с.

3. Кайл Бэнкер: MongoDB в действии: Издательство: ДМК-Пресс, 2017 г. 394 с.

4. Чакон, Страуб: Git для профессионального программиста: Издательство: Питер, 2019 г. 496 с.

5. Хассан Гома: UML Проектирование систем реального времени, распределенных и параллельных приложений: Издательство: ДМК-Пресс, 2016 г. 700 с.

6. Алексей Шабаршин: Элементарный Telegram. Все, что нужно знать о самом перспективном мессенджере страны: Издательство: Бомбора, 2023 г. 224 с.

7. Мэтью Бреннан: TikTok. Фабрика внимания. История взлета: Издательство: АСТ, 2021 г. 288 с.

8. Дмитрий Осипов: Технологии проектирования баз данных: Издательство: ДМК-Пресс, 2019 г. 498 с.

9. Главные понятия социальных сетей - http://itua.info/internet/15459.html

10. Владимир Волк: Базы данных. Проектирование, программирование, управление и администрирование. Учебник для СПО: Издательство: Лань, 2022 г. 340 с.

11. Первая в мире социальная сеть - http://ingvarr.net.ru/publ/161–1–0–9906

12. Официальная документация React – https://ru.react.js.org/docs/getting-started.html

13. Сэмми Пьюривал: Основы разработки веб-приложений: Издательство: Питер, 2015 г. 272 с.

14. Vasan Subramanian: Pro MERN Stack: 2019 г. 552 с.

15. Онлайн-руководство по MongoDB - https://metanit.com/nosql/mongodb/

16. Официальная документация Node.js - https://nodejs.org/en/docs

17. Документация MATERIAL-UI - https://v4.mui.com/ru/

18. Аквино, Ганди: Front-end. Клиентская разработка для профессионалов. Node.js, ES6, REST: Издательство: Питер, 2017 г. 512 с.

19. Социальная сеть «В Контакте» - https://ru.wikipedia.org/wiki/ВКонтакте

20. Джей Гивакс: Паттерны проектирования API: Издательство: Питер, 2023 г. 512 с.

21. Арно Лоре: Проектирование веб-API: Издательство: ДМК-Пресс, 2020 г. 440 с.

22. Меджуи, Уайлд, Митра: Непрерывное развитие API. Правильные решения в изменчивом технологическом ландшафте: Издательство: Питер, 2023 г. 368 с.

23. API - https://ru.wikipedia.org/wiki/API

24. Социальная сеть «В Контакте» - https://ru.wikipedia.org/wiki/Одноклассники_(социальная_сеть)

24. Дзен (контентная платформа) - https://ru.wikipedia.org/wiki/Дзен_(контентная_платформа)

25. WhatsApp - https://ru.wikipedia.org/wiki/WhatsApp

26. Трэйси Осборн: Веб-дизайн для недизайнеров: Издательство: Питер, 2022 г. 286 с.

27. Официальный сайт MongoDB - https://www.mongodb.com

28. LinkedIn - https://ru.wikipedia.org/wiki/LinkedIn

29. Visual Studio Code- https://code.visualstudio.com

30. GitHub - https://github.com

ПРИЛОЖЕНИЯ

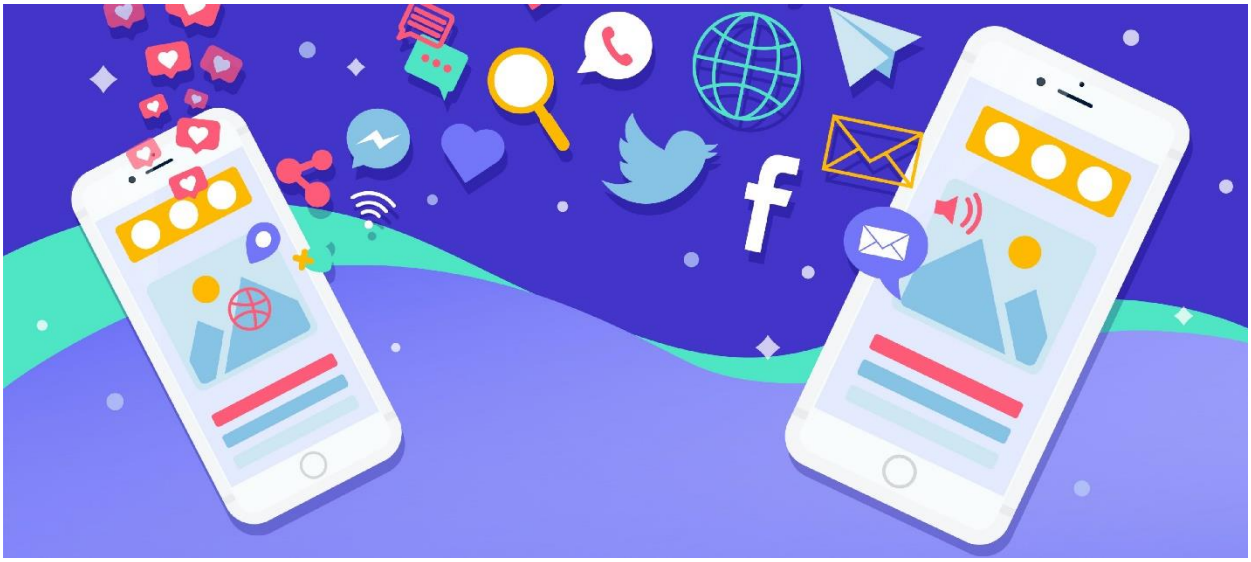


Рис 1.1 – Соцсети

Рис 1.2 – История

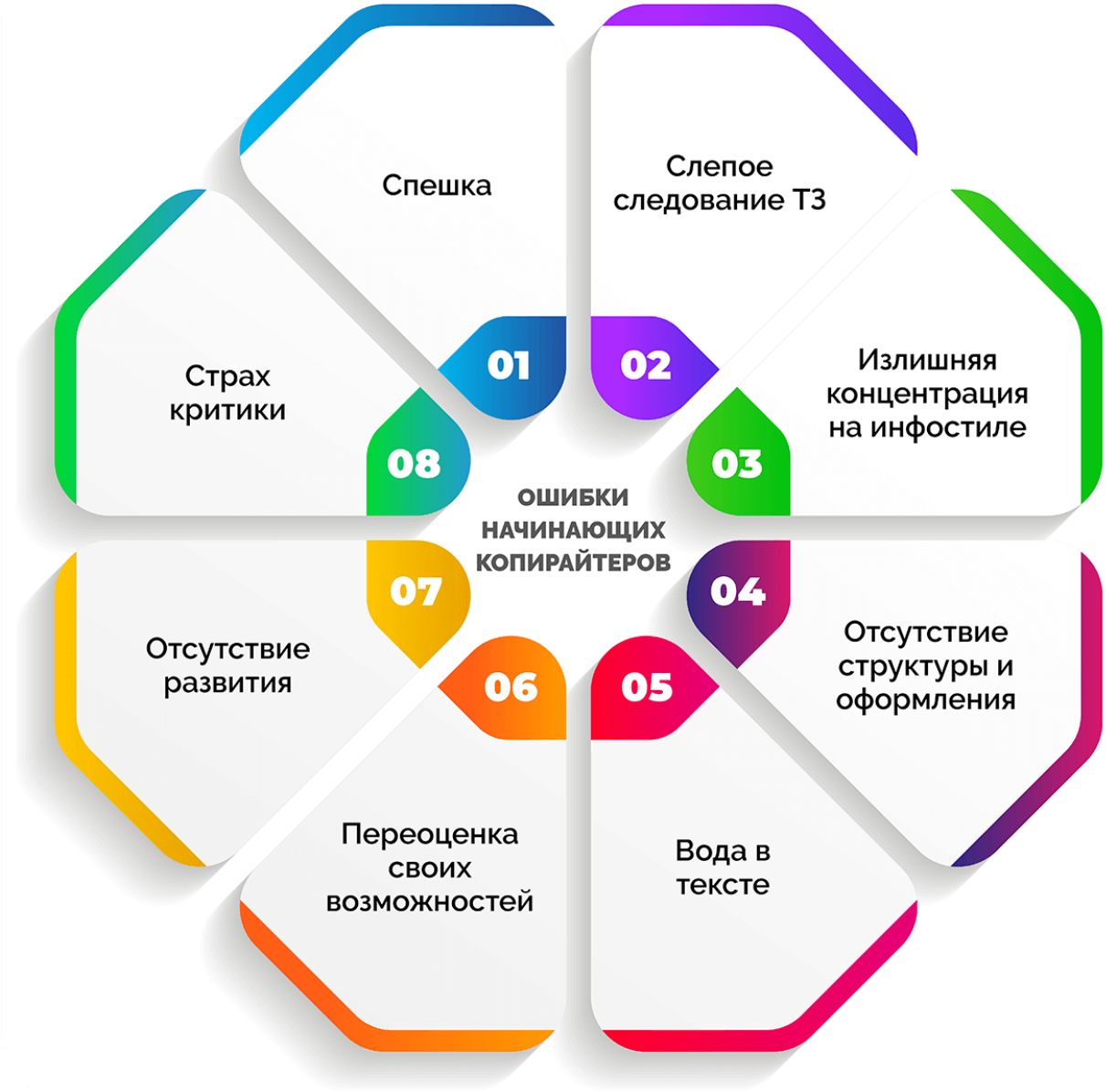


Рис 1.3 – Что можно делать

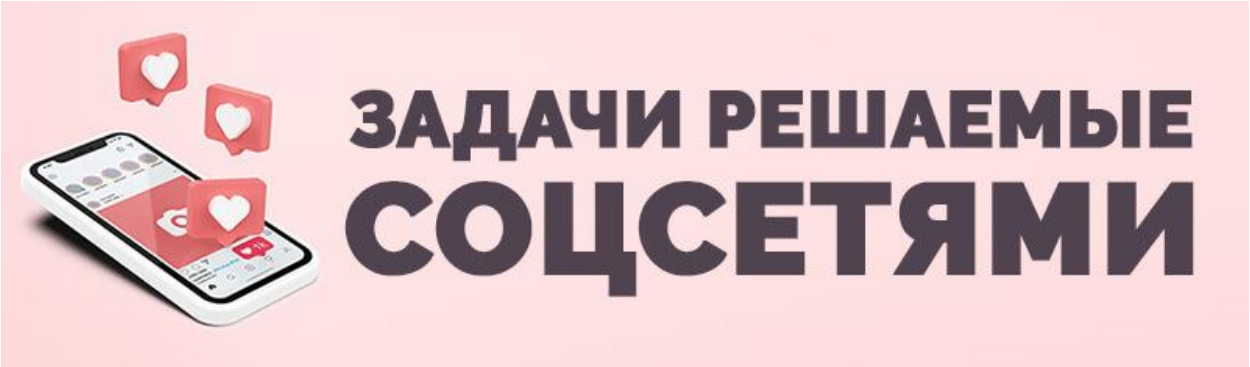


Рис 1.4 – Задачи

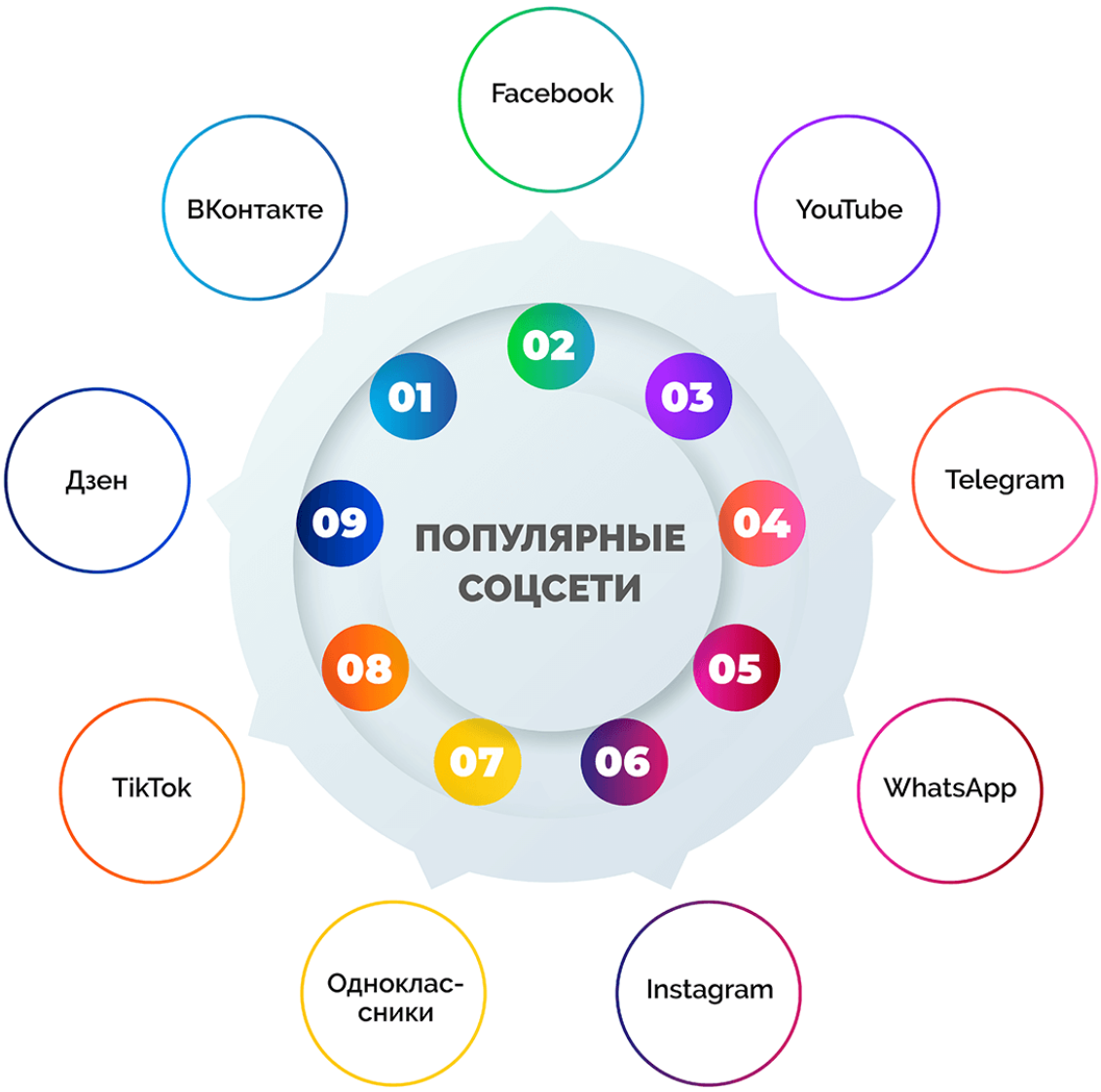


Рис 1.5 – Популярные социальные сети

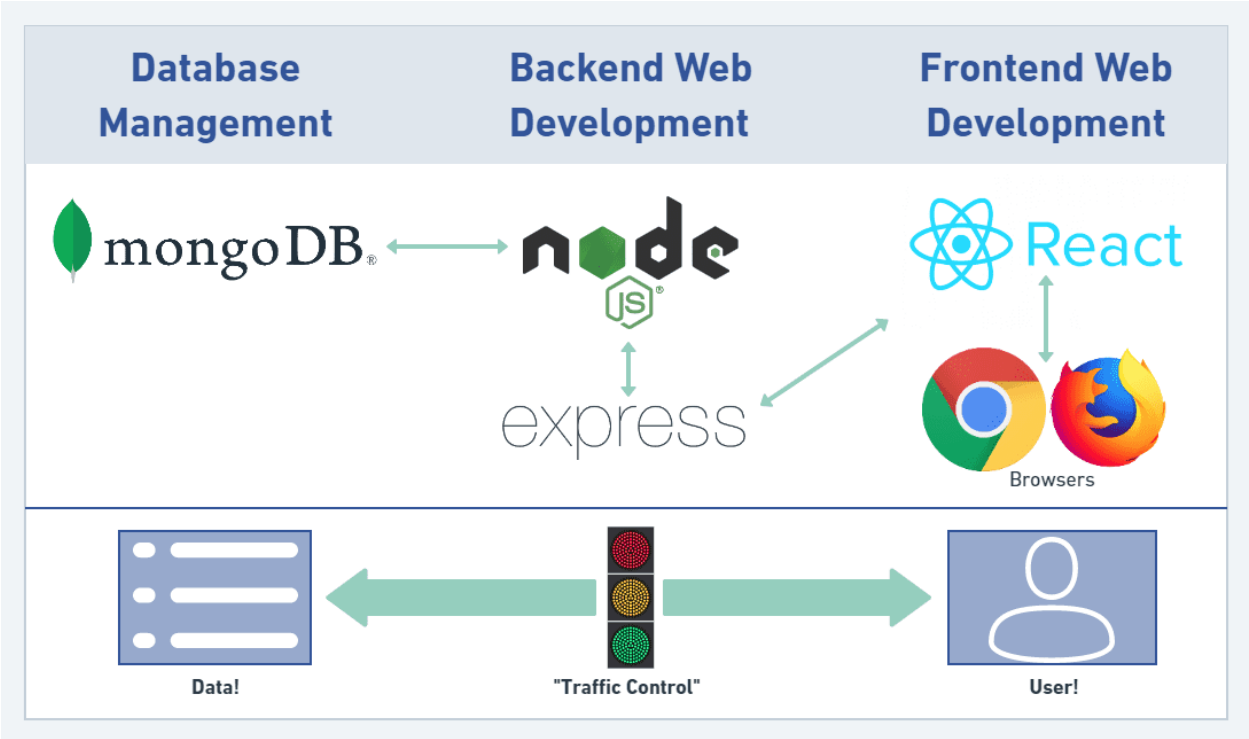


Рис 1.6 – Описание раюоты технологии MERN

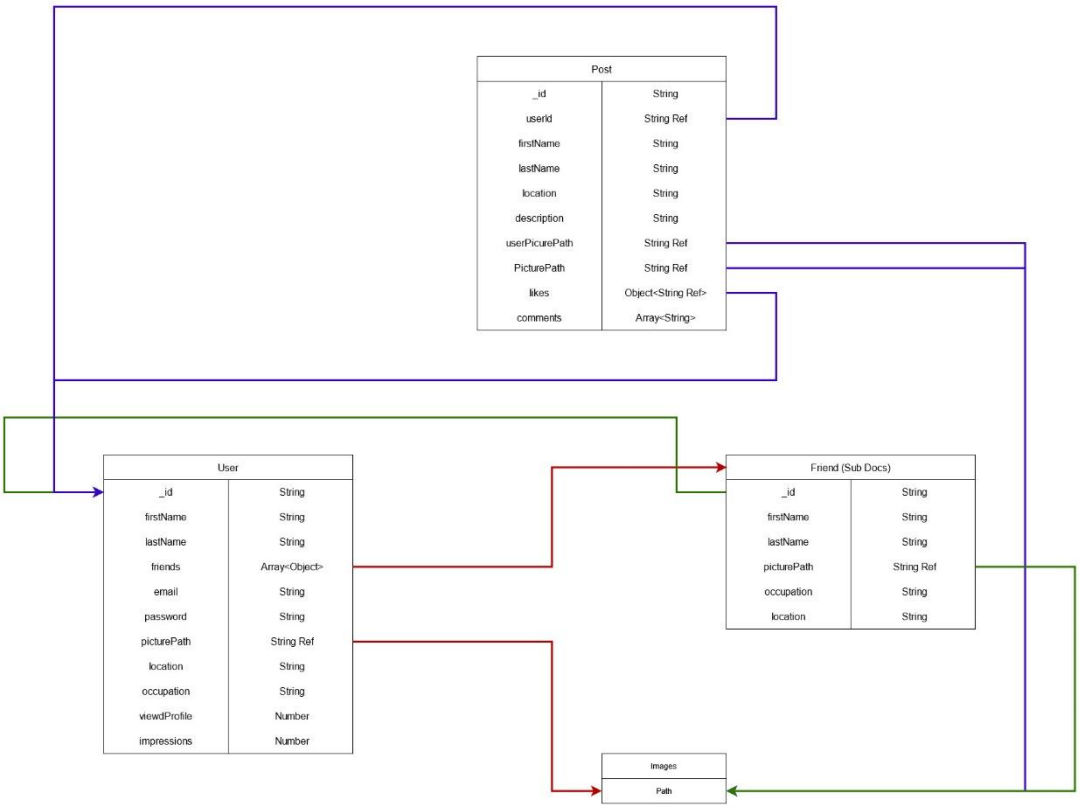


Рис 2.1 – Схема данных

Рис 2.2 – Файловая структура сервера

server > JS index.js > ...

```
1   import express from "express";
2   import bodyParser from "body-parser";
3   import mongoose from "mongoose";
4   import cors from "cors";
5   import dotenv from "dotenv";
6   import multer from "multer";
7   import helmet from "helmet";
8   import morgan from "morgan";
9   import path from "path";
10  import { fileURLToPath } from "url";
11  import authRoutes from "./routes/auth.js";
12  import userRoutes from "./routes/users.js";
13  import postRoutes from "./routes/posts.js";
14  import { register } from "./controllers/auth.js";
15  import { createPost } from "./controllers/posts.js";
16  import { verifyToken } from "./middleware/auth.js";
17  import User from "./models/User.js";
18  import Post from "./models/Post.js";
19  import { users, posts } from "./data/index.js";
```

Рис 2.3 – Импортирование библиотек и модулей

```
/* CONFIGURATIONS */
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
dotenv.config();
const app = express();
app.use(express.json());
app.use(helmet());
app.use(helmet.crossOriginResourcePolicy({ policy: "cross-origin" }));
app.use(morgan("common"));
app.use(bodyParser.json({ limit: "30mb", extended: true }));
app.use(bodyParser.urlencoded({ limit: "30mb", extended: true }));
app.use(cors());
app.use("/assets", express.static(path.join(__dirname, "public/assets")));
```

Рис 2.4 – Базовые конфигурации сервера

```
/* FILE STORAGE */
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "public/assets");
  },
  filename: function (req, file, cb) {
    cb(null, file.originalname);
  },
});
const upload = multer({ storage });
```

Рис 2.5 – Хранение файлов

```
/* ROUTES WITH FILES */
app.post("/auth/register", upload.single("picture"), register);
app.post("/posts", verifyToken, upload.single("picture"), createPost);
```

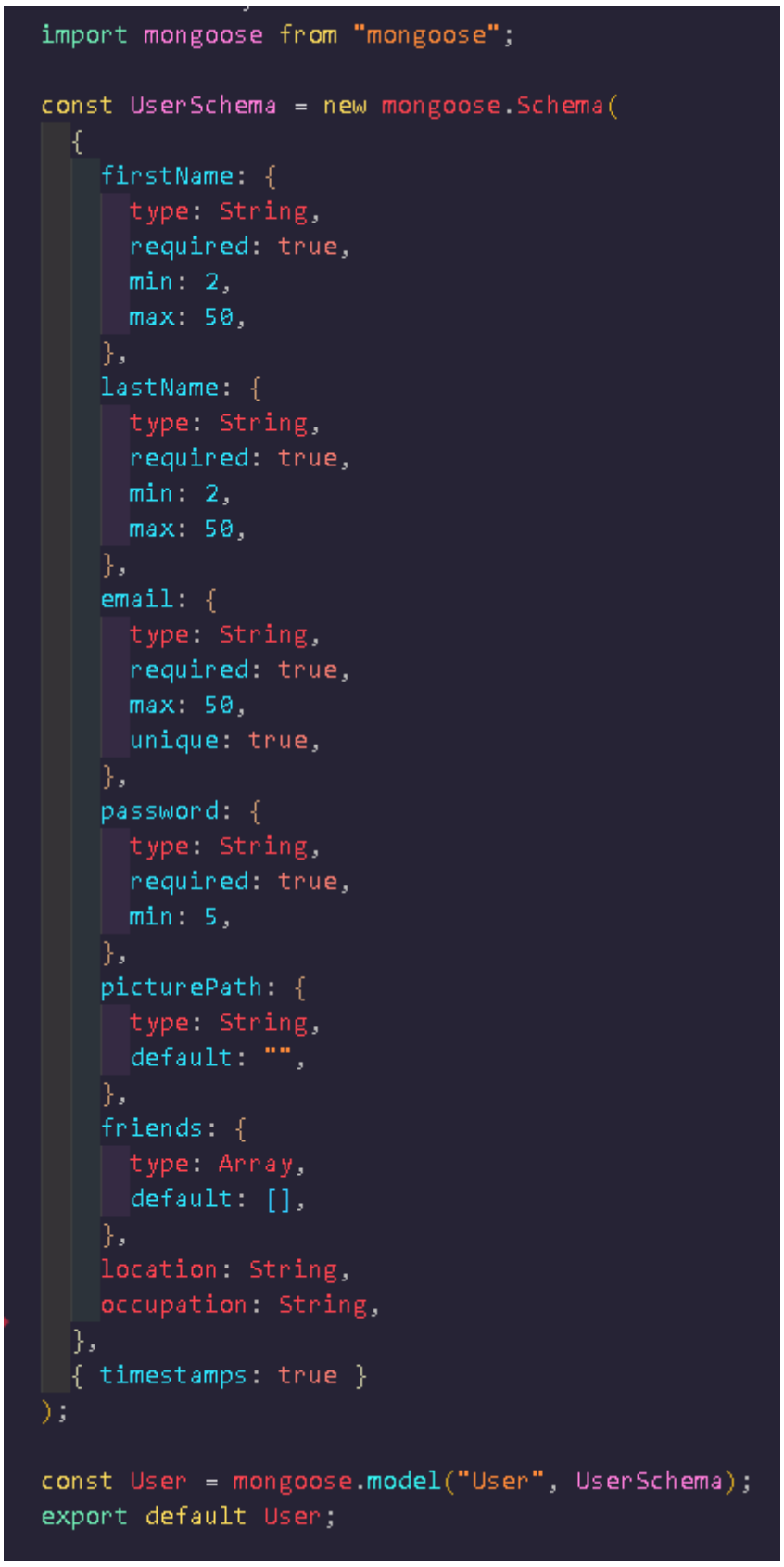
Рис 2.6 – Маршруты

```
/* ROUTES */
app.use("/auth", authRoutes);
app.use("/users", userRoutes);
app.use("/posts", postRoutes);
```

Рис 2.7 – Подключение маршрутов

```
/* MONGOOSE SETUP */
const PORT = process.env.PORT || 6001;
mongoose
  .connect(process.env.MONGO_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => {
    app.listen(PORT, () => console.log(`Server Port: ${PORT}`));
  })
  .catch((error) => console.log(`${error} did not connect`));
```

Рис 2.8 – Подключение к базе данных



```
import mongoose from "mongoose";

const UserSchema = new mongoose.Schema(
  {
    firstName: {
      type: String,
      required: true,
      min: 2,
      max: 50,
    },
    lastName: {
      type: String,
      required: true,
      min: 2,
      max: 50,
    },
    email: {
      type: String,
      required: true,
      max: 50,
      unique: true,
    },
    password: {
      type: String,
      required: true,
      min: 5,
    },
    picturePath: {
      type: String,
      default: "",
    },
    friends: {
      type: Array,
      default: [],
    },
    location: String,
    occupation: String,
  },
  { timestamps: true }
);

const User = mongoose.model("User", UserSchema);
export default User;
```

Рис 2.9 – Модель пользователей

```
import mongoose from "mongoose";

const postSchema = mongoose.Schema(
  {
    userId: {
      type: String,
      required: true,
    },
    firstName: {
      type: String,
      required: true,
    },
    lastName: {
      type: String,
      required: true,
    },
    location: String,
    description: String,
    picturePath: String,
    userPicturePath: String,
    likes: {
      type: Map,
      of: Boolean,
    },
    comments: {
      type: Array,
      default: [],
    },
  },
  { timestamps: true }
);

const Post = mongoose.model("Post", postSchema);

export default Post;
```

Рис 2.10 – Модель постов

```
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import User from "../models/User.js";

/* REGISTER USER */
export const register = async (req, res) => {
  try {
    const {
      firstName,
      lastName,
      email,
      password,
      picturePath,
      friends,
      location,
      occupation,
    } = req.body;

    const salt = await bcrypt.genSalt();
    const passwordHash = await bcrypt.hash(password, salt);

    const newUser = new User({
      firstName,
      lastName,
      email,
      password: passwordHash,
      picturePath,
      friends,
      location,
      occupation,
    });
    const savedUser = await newUser.save();
    res.status(201).json(savedUser);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

Рис 2.11 – Функция регистации

```
/* LOGGING IN */
export const login = async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email: email });
    if (!user) return res.status(400).json({ msg: "User does not exist. " });
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ msg: "Invalid credentials. " });
    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET);
    delete user.password;
    res.status(200).json({ token, user });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

Рис 2.12 – Функция авторизации

```
import Post from "../models/Post.js";
import User from "../models/User.js";

/* CREATE */
export const createPost = async (req, res) => {
  try {
    const { userId, description, picturePath } = req.body;
    const user = await User.findById(userId);
    const newPost = new Post({
      userId,
      firstName: user.firstName,
      lastName: user.lastName,
      location: user.location,
      description,
      userPicturePath: user.picturePath,
      picturePath,
      likes: {},
      comments: [],
    });
    await newPost.save();

    const post = await Post.find();
    res.status(201).json(post);
  } catch (err) {
    res.status(409).json({ message: err.message });
  }
};
```

Рис 2.13 – Создание поста

```
/* READ */
export const getFeedPosts = async (req, res) => {
  try {
    const post = await Post.find();
    res.status(200).json(post);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};
```

Рис 2.14 – Получение списка постов

```
export const getUserPosts = async (req, res) => {
  try {
    const { userId } = req.params;
    const post = await Post.find({ userId });
    res.status(200).json(post);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};
```

Рис 2.15 – Получение списка постов пользователя

```
/* UPDATE */
export const likePost = async(req,res) => {
  try {
      const {id} = req.params;
      const {userId} = req.body;
      const post = await Post.findById(id);
      const isLiked = post.likes.get(userId);
      if(isLiked){
          post.likes.delete(userId);
      }else{
          post.likes.set(userId , true);
      }
      const updatedPost = await Post.findByIdAndUpdate(id , {likes:post.likes},{new:true});
      res.status(200).json(updatedPost);
  } catch (err) {
      res.status(404).json({error:err.message});
  }
}
```

Рис 2.16 – Обновление поста

```
export const postComment = async(req,res) => {
  try {
      const {postId, userId} = req.params;
      const {comment} = req.body;
      const newComment = {
          userId,
          comment
      };
      const post = await Post.findById(postId);
      post.comments.push(newComment);
      const updatedPost = await Post.findByIdAndUpdate(postId , {comments:post.comments},{new:true});
      res.status(200).json(updatedPost);
  } catch (err) {
      res.status(404).json({error:err.message});
  }
}
```

Рис 2.17 – Добавление комментария к посту

```
export const deleteComment = async(req,res) => {
  try {
      const {postId} = req.params;
      const {comment} = req.body;
      const post = await Post.findById(postId);
      // console.log(post);
      const updatedPost = post.comments.filter((item,i)=>{
          return item.comment !== comment;
      });
      const updatedNewPost = await Post.findByIdAndUpdate(postId , {comments:updatedPost},{new:true});
      res.status(200).json(updatedNewPost);
  } catch (err) {
      res.status(404).json({error:err.message});
  }
}
```

Рис 2.18 – Удаление комментария

```
import User from "../models/User.js";

/* READ */
export const getUser = async (req, res) => {
  try {
    const { id } = req.params;
    const user = await User.findById(id);
    res.status(200).json(user);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};
```

Рис 2.19 – получение информации о пользователе

```
export const getUserFriends = async (req, res) => {
  try {
    const { id } = req.params;
    const user = await User.findById(id);

    const friends = await Promise.all(
      user.friends.map((id) => User.findById(id))
    );
    const formattedFriends = friends.map(
      ({ _id, firstName, lastName, occupation, location, picturePath }) => {
        return { _id, firstName, lastName, occupation, location, picturePath };
      }
    );
    res.status(200).json(formattedFriends);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};
```

Рис 2.20 – получение списка друзей пользователя

```
/* UPDATE */
export const addRemoveFriend = async (req, res) => {
  try {
    const { id, friendId } = req.params;
    const user = await User.findById(id);
    const friend = await User.findById(friendId);

    if (user.friends.includes(friendId)) {
      user.friends = user.friends.filter((id) => id !== friendId);
      friend.friends = friend.friends.filter((id) => id !== id);
    } else {
      user.friends.push(friendId);
      friend.friends.push(id);
    }
    await user.save();
    await friend.save();

    const friends = await Promise.all(
      user.friends.map((id) => User.findById(id))
    );
    const formattedFriends = friends.map(
      ({ _id, firstName, lastName, occupation, location, picturePath }) => {
        return { _id, firstName, lastName, occupation, location, picturePath };
      }
    );

    res.status(200).json(formattedFriends);
  } catch (err) {
    res.status(404).json({ message: err.message });
  }
};
```

Рис 2.21 – добавление или удаление друга из списка друзей

```
import jwt from "jsonwebtoken";

export const verifyToken = async (req, res, next) => {
  try {
    let token = req.header("Authorization");

    if (!token) {
      return res.status(403).send("Access Denied");
    }

    if (token.startsWith("Bearer ")) {
      token = token.slice(7, token.length).trimLeft();
    }

    const verified = jwt.verify(token, process.env.JWT_SECRET);
    req.user = verified;
    next();
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

Рис 2.22 – ПО для проверки токена Аутентификации

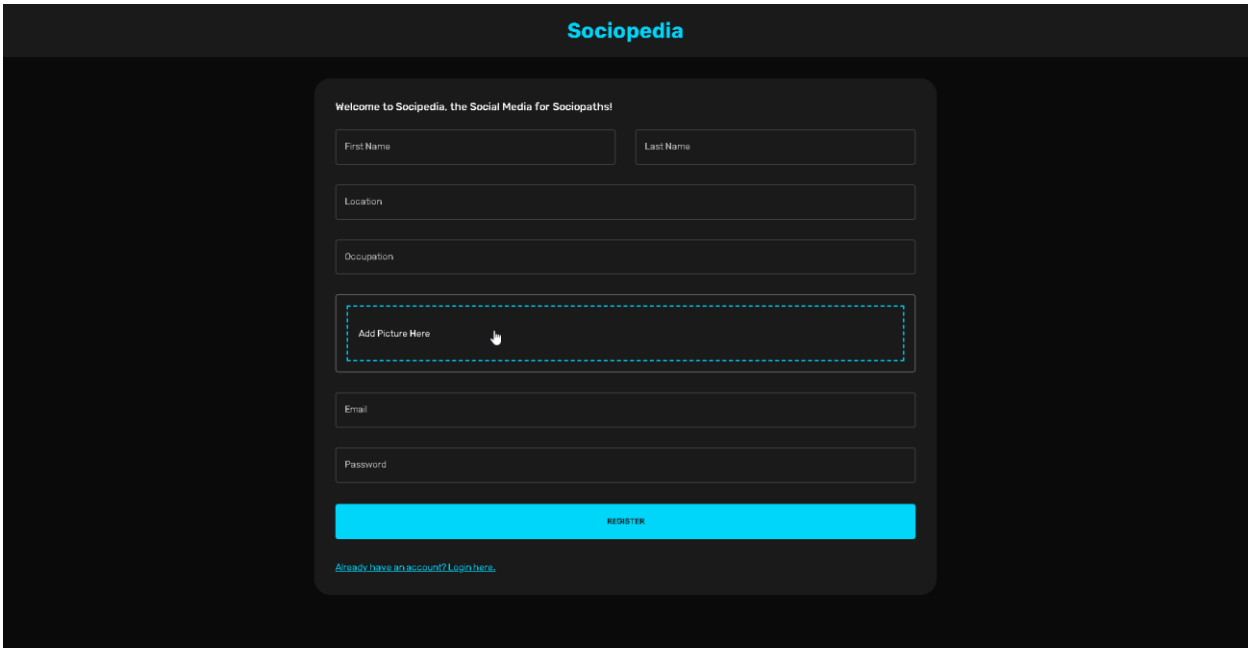


Рис 2.23 – это Регистрация

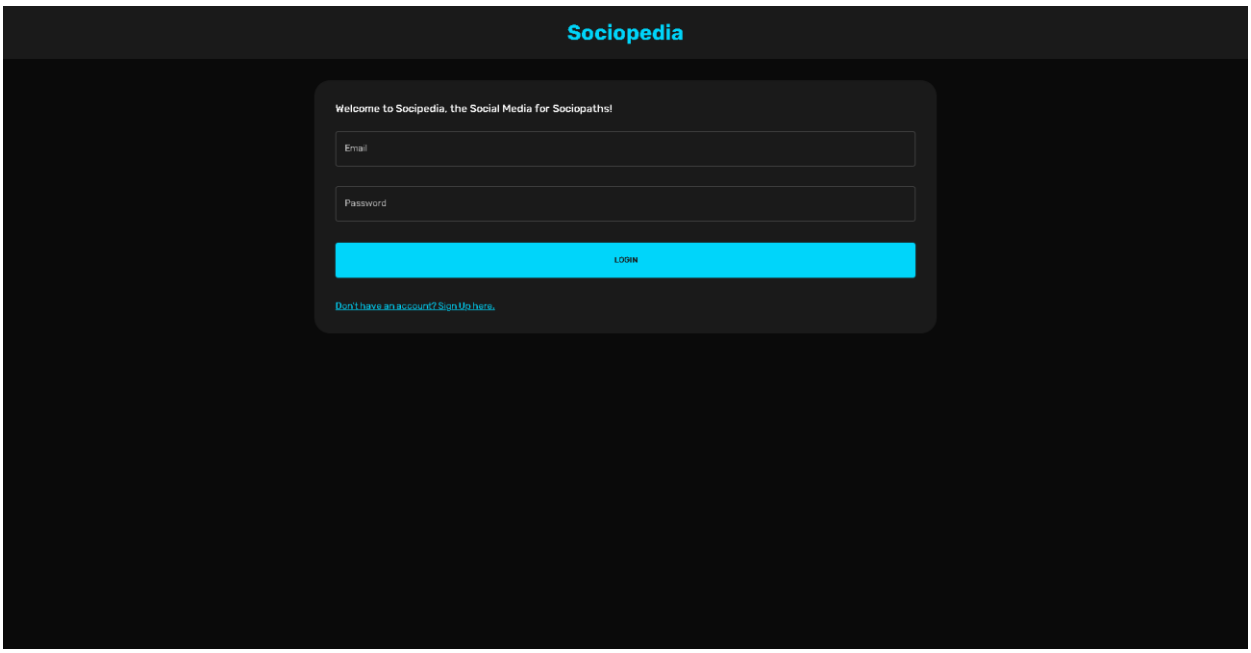


Рис 2.24 – Авторизация

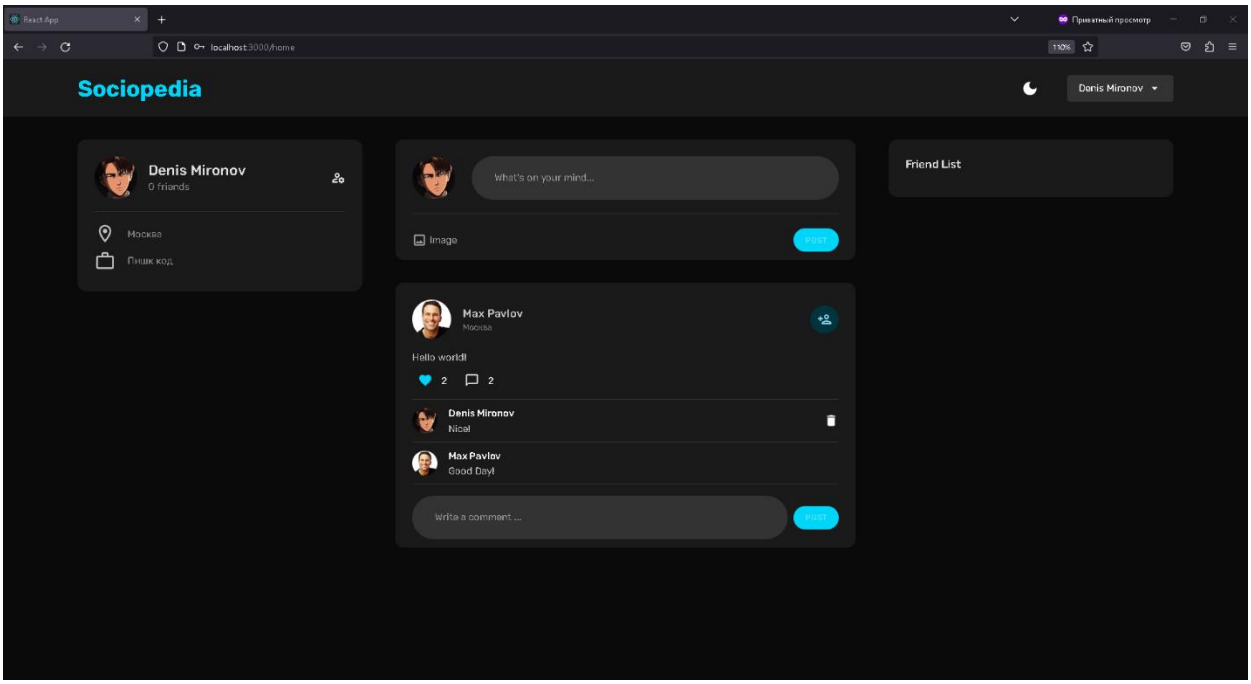


Рис 2.25 – Главная страница

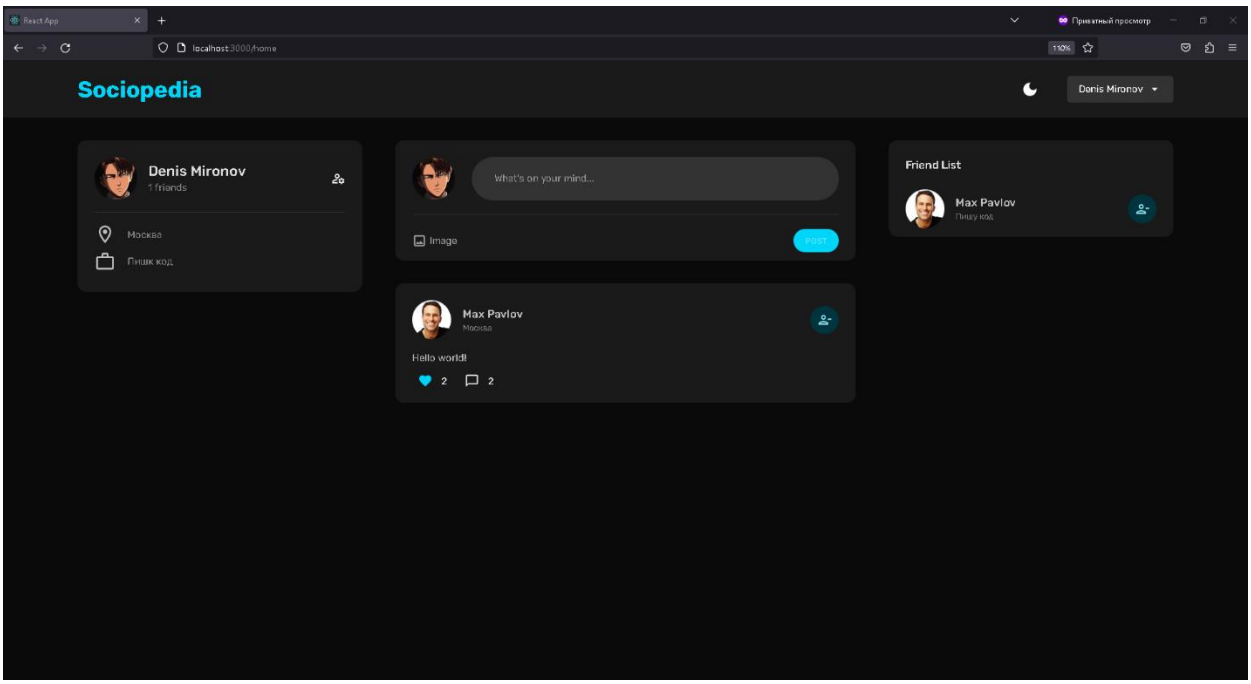


Рис 2.26 – Отображение списка друзей

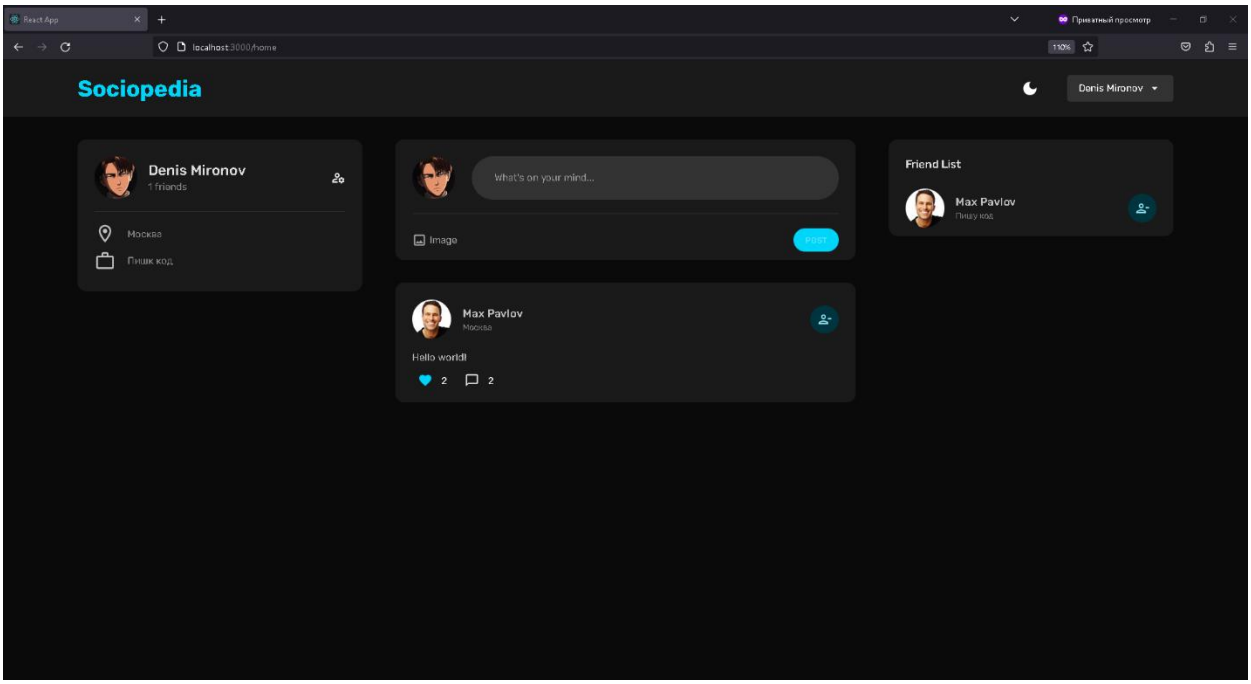


Рис 2.27 – Тёмная тема

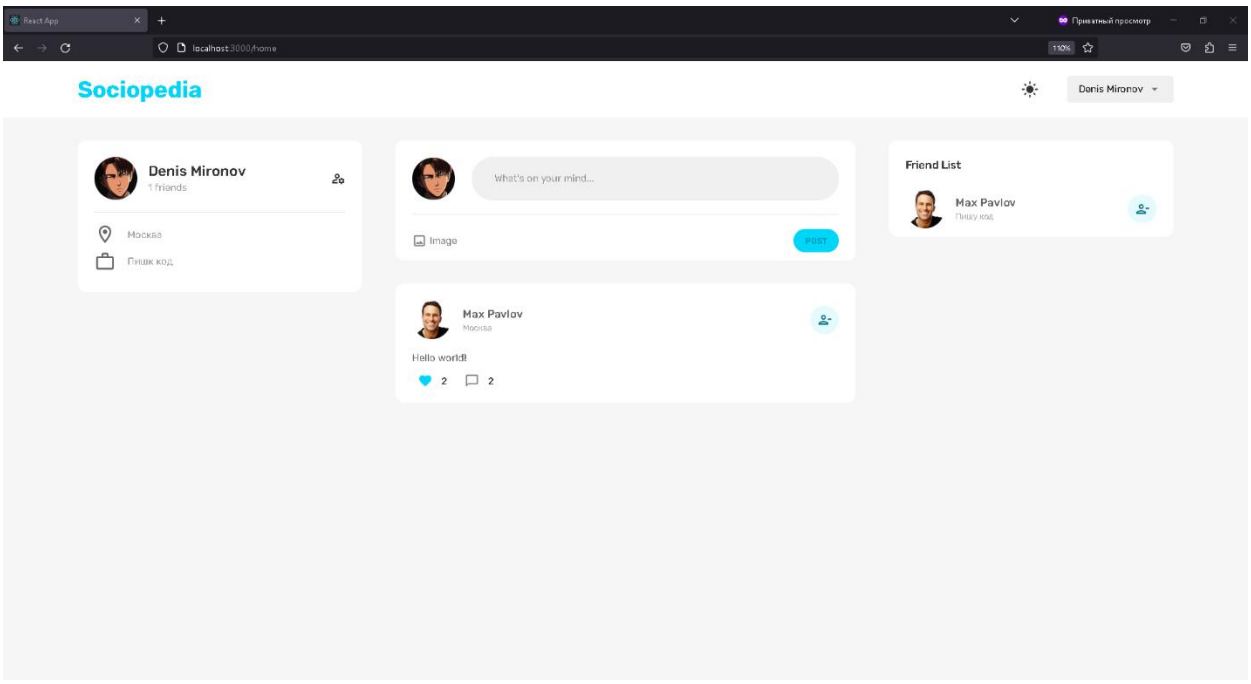


Рис 2.28 – Светлая тема

Автономная некоммерческая организация
профессионального образования
«Московский областной колледж информации и технологий»

ПОСЛЕДНИЙ ЛИСТ ДИПЛОМНОЙ РАБОТЫ

Работа выполнена мной совершенно самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

Обучающийся	Макаров Д. В	_____	05.06.2023 г.
	Ф.И.О.	подпись	дата