**ачМИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Механико-математический факультет**

**Кафедра веб-технологий и компьютерного моделирования**

**РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ КЛОНА СОЦСЕТИ-НОВОСТНОЙ ЛЕНТЫ FACEBOOK**

Курсовая работа

Логаш Полины Александровны

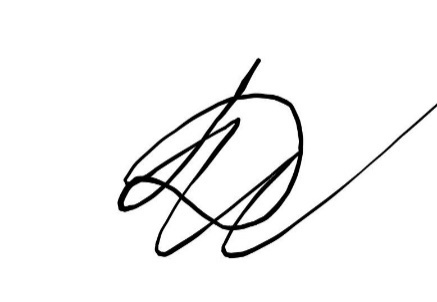
Студентки 3 курса

Специальности «Математика

и информационные технологии»

Научный руководитель:

Ассистент Д. В. Филимонов

9

Минск, 2023

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3

ГЛАВА 1. ОБЗОР ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ, ИСПОЛЬЗУЕМЫХ ПРИ СОЗДАНИЯ ВЕБ-ПРИЛОЖЕНИЯ 4

* 1. Преимущества JavaScript фреймворков 4
  2. Сравнение и выбор фреймворк для разработки клиентской

части приложения 5

* 1. Next.js 5
  2. OAuth 7
  3. Spring 8
  4. MySQL 11

ГЛАВА 2 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ВЕБ-РИЛОЖЕНИЯ 13

2.1. Разработка авторизации через Facebook посредством OAuth 13

2.2 Реализация страницы-новостной ленты 17

ГЛАВА 3. ЗАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ВЕБ-ПРИЛОЖЕНИЯ 20

3.1 Разработка структуры базы данных и передача данных

ЗАКЛЮЧЕНИЕ 23

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 24

ПРИЛОЖЕНИЕ. ЛИСТИНГИ ПРОГРАММ 25

**ВВЕДЕНИЕ**

В наше время всё больше появляется соцсетей. Они стали неотъемлемой частью жизни многих людей. Они позволяют нам оставаться на связи с друзьями и близкими, делиться новостями и фотографиями, находить новых друзей и интересные сообщества. Одним из главных атрибутов социальных сетей является новостная лента, которая позволяет пользователям получать информацию о том, что происходит в мире и в жизни их друзей.

Новостная лента соцсети – это список сообщений, который отображается на главной странице пользователя. Эти сообщения могут содержать фотографии, видео, текстовые посты, ссылки на статьи и другие материалы. Новости отображаются в порядке обновления, то есть самые свежие сообщения появляются вверху списка.

Мне показалось интересным сделать клон новостной ленты одной из самых известных социальных сетей Facebook.

Цель данной работы – создание клона новостной ленты соцсети Facebook.

Для реализации поставленной цели необходимо решить следующие задачи:

* изучить инструментальные средства для разработки клиентской части веб-приложений (выбор фреймворка и других инструментов);
* изучить инструментальные средства для обеспечения безопасности;
* разработать архитектуру клиентской части;
* разработать архитектуру серверной части;

**ГЛАВА 1**

**ОБЗОР ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ, ИСПОЛЬЗУЕМЫХ ПРИ СОЗДАНИЯ ВЕБ-ПРИЛОЖЕНИЯ**

* 1. **Преимущества JavaScript фреймворков**

Фреймворк – это программный продукт, который упрощает создание и поддержку технически сложных или нагруженных проектов. Фреймворк, как правило, содержит только базовые программные модули, а все специфичные для проекта компоненты реализуются разработчиками на их основе. Тем самым достигается не только высокая скорость разработки, но и большая производительность и надёжность решений.

JavaScript фреймворки предоставляют набор компонентов и функций, которые можно использовать для создания современных веб-приложений. Обычно они обрабатывают и ускоряют многие громоздкие задачи веб-разработки, такие как упрощение HTTP-вызовов на сервер или предоставление методов для взаимодействия и управления DOM (объектной моделью документа) веб-страницы и т.д.

Преимущества использования фреймворков по сравнению с использованием чистого JavaScript:

* *Эффективность*. Одним из преимуществ JavaScript фреймворков является наличие встроенных функций. Благодаря встроенным функциям, разработчик в итоге пишет меньше кода и быстрее разрабатывает приложения, по сравнению с разработчиком, создающим все с нуля. Это означает, что задачи, которые обычно требуют сотни и тысячи строк кода, теперь могут быть выполнены гораздо проще.
* *Совместная работа пользователей конкретного фреймворка*. Многие JavaScript фреймворки в значительной степени полагаются на свои сообщества и отзывы, которые эти сообщества предоставляют. Это является несомненным плюсом, потому что пользователи не только предоставляют обратную связь, связанную с ошибками, но также могут предлагать новые функции.
* *Безопасность*. Чем больше используется фреймворк, тем больше требуется безопасности. Это важная причина для выбора структуры, которая имеет большое и активное сообщество. Это же сообщество выступает в роли долгосрочных тестировщиков, поэтому они могут сообщать о проблемах безопасности и даже предоставлять решения проблем.
  1. **Сравнение и выбор фреймворк для разработки клиентской части приложения**

Рассмотрев самые популярные фреймворки JavaScript и изучив частоту использования различных библиотек по статистике скачивания из пакетного менеджера NPM (рис.1), среди трёх основных фреймворков – Angular, Vue и React – выбор пал на самый популярный – React.

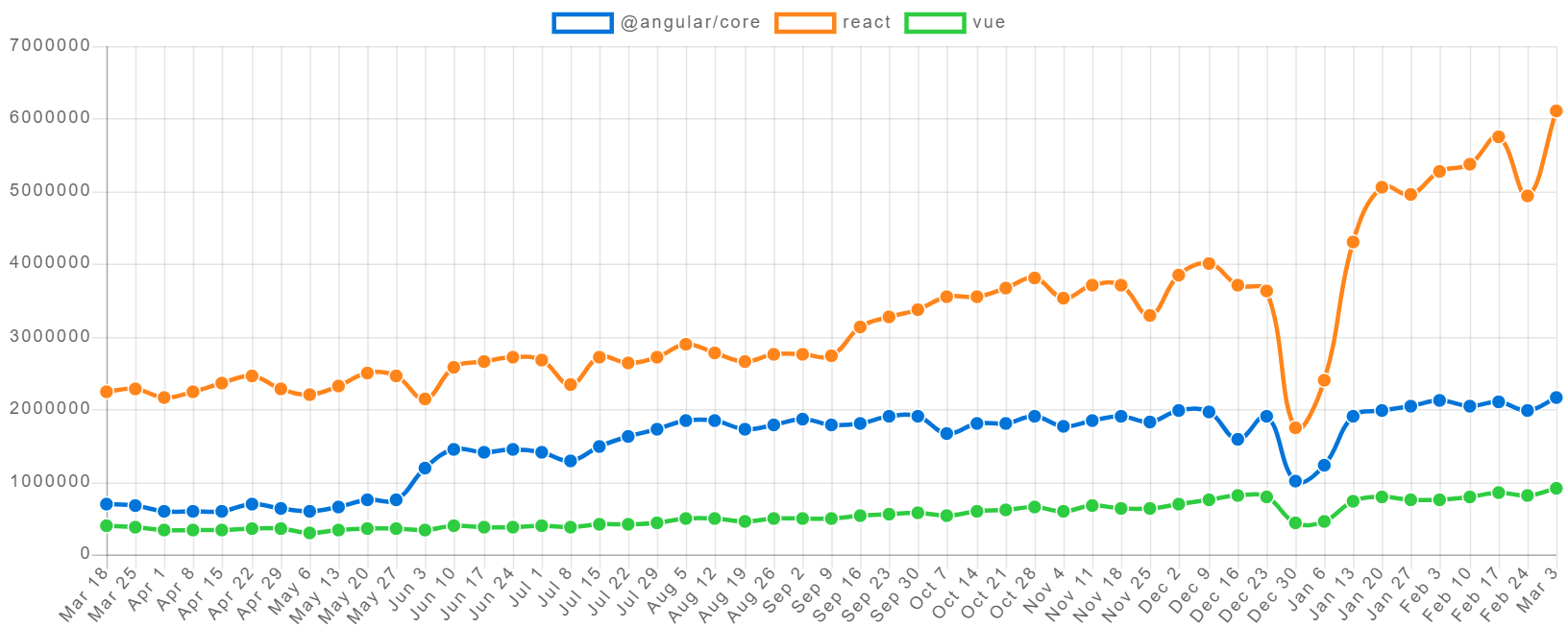


Рисунок 1. Количество скачиваний фреймворков Angular, Vue и React в NPM

* 1. **Next.js**

Сейчас React.js - это один из самых популярных фреймворков, с которым должен уметь работать хотя бы на примитивном уровне любой веб-разработчик. Появился он в 2013 году под покровительством команды Facebook. На базе React были созданы дополнительные фреймворки, призванные расширить функционал оригинала и упростить выполнение некоторых задач. **Одним из таких решений является Next.js.**

**Next.js — это фреймворк с открытым исходным кодом на основе и для работы с React, который создает приложения с рендерингом на стороне сервера.**

**Next.js используется для создания целевых страниц, SEO-дружественных веб-сайтов, магазинов электронной коммерции и всех видов веб-приложений, которым требуется быстрая и высокопроизводительная загрузка.**

Почему стоит использовать Next.js, а не другие фреймворки React?

* Простая настройка и развертывание.
* Его легко кодировать — по сравнению с самим React и другими фреймворками, работающими с React, Next.js требует меньше кода.
* Скорость. Приложения, созданные с помощью Next.js, работают быстро благодаря рендерингу на стороне сервера и генерации статики, обеспечивая более разумный способ обработки данных. Рендеринг на стороне сервера будет работать настолько быстро, насколько быстро сервер обрабатывает запросы. Встроенные функции оптимизации изображений также повышают производительность.
* Быстрый рендеринг — каждое изменение в файле сразу же становится видимым при обновлении страницы. Компонент рендерится на месте, что облегчает слежение за изменениями по мере их внесения.
* Встроенный CSS — с Next.js вы можете импортировать стили CSS из файла JavaScript, чтобы использовать их в строке для более быстрого рендеринга.
* Лучшая оптимизация изображений — изображения изменяются в размерах и передаются с использованием лучших, современных форматов, таких как WebP (при этом они открыты для новых форматов), а изображения настраиваются для адаптации к небольшим экранам просмотра.
* SEO — Для тех, кто стремится к улучшению SEO, заголовки и ключевые слова для каждой страницы легко создать.
* Совместимость с ESLint .
* Поддержка API — API сторонних разработчиков могут расширить возможности вашего проекта, а Next.js легко подключается к ним, помогая создавать собственные маршруты API.
  1. **OAuth**

OAuth 2.0 — протокол авторизации, позволяющий выдать одному сервису (приложению) права на доступ к ресурсам пользователя на другом сервисе. Протокол избавляет от необходимости доверять приложению логин и пароль, а также позволяет выдавать ограниченный набор прав, а не все сразу.

Данный протокол используют такие платформы, как Google, Facebook, Mail.Ru, Twitter и другие. А так как данное приложение

Плюсы использования протокола Auth:

* Обращение к ресурсам происходит по HTTP/HTTPS с указанием токена в заголовках. Это позволяет использовать OAuth практически в любых решения: мобильных и десктоп-приложениях, сайтах и даже в плагинах для браузеров.
* Возможность авторизации пользователя.
* Популярность — большинство компаний используют его в своих API.
* Простота реализации и большое количество литературы.
* Наличие готовых решений, которые можно изменять под свои нужды.
  1. **Spring**

При выборе языка и фреймворка для серверной части приложения я также была ориентация на статистику из фреймворков, которые больше всего нравятся разработчикам по исследованиям Stack Overflow (рис. 3). По этой статистике стразу за фреймворками для JavaScript следует Spring, поэтому выбор пал на него.

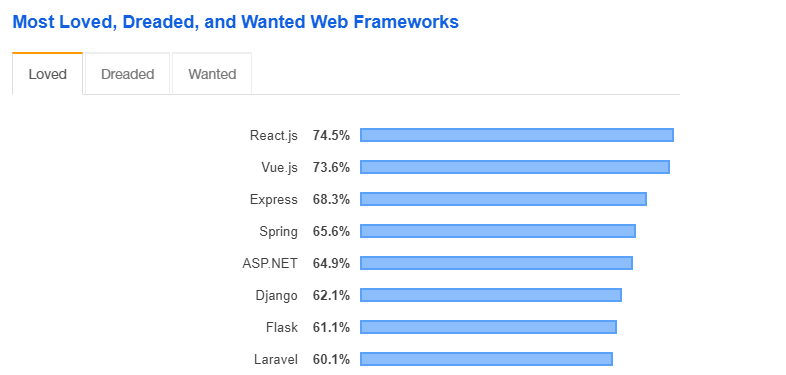


Рисунок 2. Процент излюбленности фреймворков разработчиками по исследованиям StackOverflow

Spring Framework – мощный фреймворк для разработки приложений. В более широком смысле Spring Framework – это хорошо разработанный инструмент, который поддерживает несколько веб-приложений, использующих Java в качестве языка программирования. Фреймворк был замечен на рынке благодаря модульности его базового функционала. Его можно разделить на разные модули, каждый из которых выполняет свою функцию.

Одной из важных особенностей Spring Framework является его легкость. Данный фреймворк очень легкий с точки зрения размера и функциональности благодаря реализации POJO, которая не требует наследования каких-либо классов и реализации каких-либо интерфейсов. Управление транзакциями используется для объединения нескольких API-интерфейсов управления транзакциями и координации транзакций объектов Java. Spring проектирует и управляет жизненным циклом и настройками объектов приложения. Внедрение зависимостей позволяет разрабатывать слабосвязанные приложения. Это значительно упрощает выполнение их модульного тестирования. Spring обеспечивает интеграцию фреймворков, решающих более сложные проблемы. Например, IBATIS, Hibernate, Toplink и т.д.

Spring Web MVC, также известный как Spring MVC, является веб-средой Spring. Spring MVC позволяет создавать различные приложения: от небольших веб-сайтов до сложных веб-сервисов. Структура шаблона Spring MVC состоит из модели, контроллера и представления

Модель содержит данные, которые требуется отобразить на веб-странице. Однако данные полностью независимы от HTML, это простые объекты Java, из которых состоит приложение.

Представлением будет HTML-шаблоном, который является каркасом HTML-страницы, написанной с помощью определенных библиотек шаблонов. Эти библиотеки позволяют включать заполнители в шаблоны, которые позволяют получить доступ к данным модели, например имени пользователя.

Контроллер будет аннотированным методом @Controller, который отвечает на HTTP-запрос и знает, как преобразовать HTTP-запрос в объекты Java, а также объекты Java в ответ HTML.

К достоинствам Spring MVC однозначно можно отнести саму концепцию ориентированности на запросы: когда разработчик четко понимает, какие запросы и куда отправляются со стороны клиента, это приводит к более рациональному дизайну приложения, к более оптимизированному и быстрому коду. Поддерживать приложение на Spring MVC легко, потому что все прозрачно. Нет зависимости от сторонних библиотек. Для создания визуальных компонентов и реализации определенной логики можно использовать библиотеки, известные всем веб-разработчикам, такие как JQuery, Twitter Bootstrap и т.д.

Spring Data JPA интегрируется с фреймворком Spring MVC, поэтому приложения, написанные на Spring Data JPA, имеют структуру шаблона MVC.

Spring Data JPA — это подпроект Spring Framework. Он предоставляет набор абстракций и утилит для работы с JPA (Java Persistence API) и Hibernate. Spring Data JPA сокращает объем шаблонного кода, необходимого для реализации уровней доступа к данным, и способствует внедрению передовых методов проектирования программного обеспечения.

Плюсы и преимущества Spring Data JPA:

* Упрощение работы с базами данных: Spring Data JPA предоставляет абстракцию для работы с базами данных, что упрощает разработку приложений и сокращает количество кода.
* Увеличение производительности: Spring Data JPA использует кэширование данных, что увеличивает производительность приложения.
* Поддержка различных баз данных: Spring Data JPA поддерживает большое количество баз данных, включая MySQL, PostgreSQL, Oracle, MongoDB и многие другие.
* Автоматическое создание запросов: Spring Data JPA автоматически создает SQL-запросы на основе методов репозитория, что упрощает работу с базами данных.
* Уменьшение количества кода: Spring Data JPA позволяет сократить количество кода, необходимого для работы с базами данных, что уменьшает время разработки и улучшает качество кода.
* Интеграция с другими фреймворками: Spring Data JPA интегрируется с другими фреймворками, такими как Spring MVC и Spring Boot, что облегчает разработку приложений.
* Улучшение безопасности: Spring Data JPA предоставляет механизмы безопасности для работы с базами данных, что защищает данные приложения от несанкционированного доступа.
* Удобство в использовании: Spring Data JPA предоставляет удобный интерфейс для работы с базами данных, что упрощает работу разработчиков и повышает производительность приложения.
  1. **MySQL**

MySQL это система управления реляционными базами данных, которая широко используется в различных приложениях и веб-сайтах. Она предоставляет мощные функции для хранения, обработки и извлечения данных, а также обеспечивает высокую производительность и надежность. MySQL поддерживает многие языки программирования и платформы, включая Java, PHP, Python, Ruby и многие другие. Благодаря своей открытой природе и большому сообществу пользователей, MySQL является одной из самых популярных систем управления базами данных в мире.

Плюсы использования MySQL:

1. Бесплатность и открытый исходный код.

2. Простота в использовании и установке.

3. Высокая производительность и надежность.

4. Поддержка многих языков программирования и платформ.

5. Широкое сообщество пользователей и разработчиков.

6. Расширяемость и гибкость в настройке.

7. Безопасность данных и возможность резервного копирования.

8. Поддержка транзакций и целостности данных.

9. Возможность масштабирования и распределенного хранения данных.

10. Интеграция с другими приложениями и системами.

**ГЛАВА 2 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ВЕБ-РИЛОЖЕНИЯ**

* 1. **Разработка авторизации через Facebook посредством OAuth**

Для реализации авторизации используется протокол OAuth, посредством которого приложению предоставляются права на доступ к ресурсам пользователя на другом сервисе.

Обычно, для авторизации приложение делает редирект на страницу авторизации через другой сервис, где у пользователя запрашивается подтверждение выдачи прав. После получения разрешения такие значения как client\_id и client\_secret, которые получались пользователем при регистрации на другом сервисе, используются для получения доступа ко всем ресурсам пользователя.

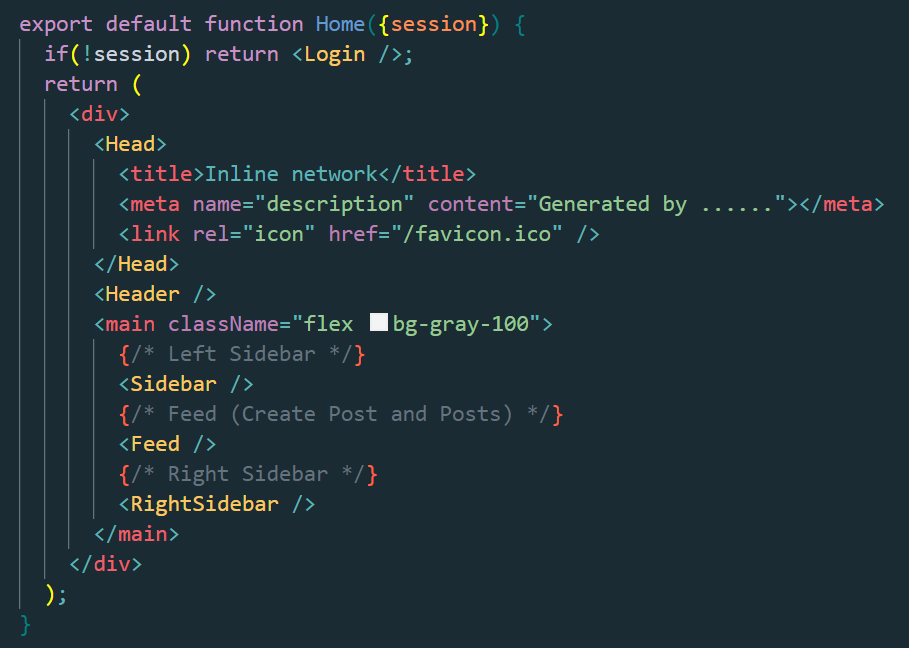
В данном приложении есть доступ к примеру таких значений client\_id и client\_secret от провайдера Facebook, которые хранятся в файле .env.local.

При запуске приложения, изначально открывается страница index.js, но перед открытием происходит проверка запущена ли сессия (Листинг 1). Проверка производится в функции Home в файле index.js.

Если предыдущая сессия не была запущена, то происходит редирект на страницу логина Login.js, на которой пользователю предлагается произвести вход (рис.3).

Далее, после нажатия на Login, перед клиентом открывается выбор регистрации через несколько приложений (рис.4).

Листинг 1



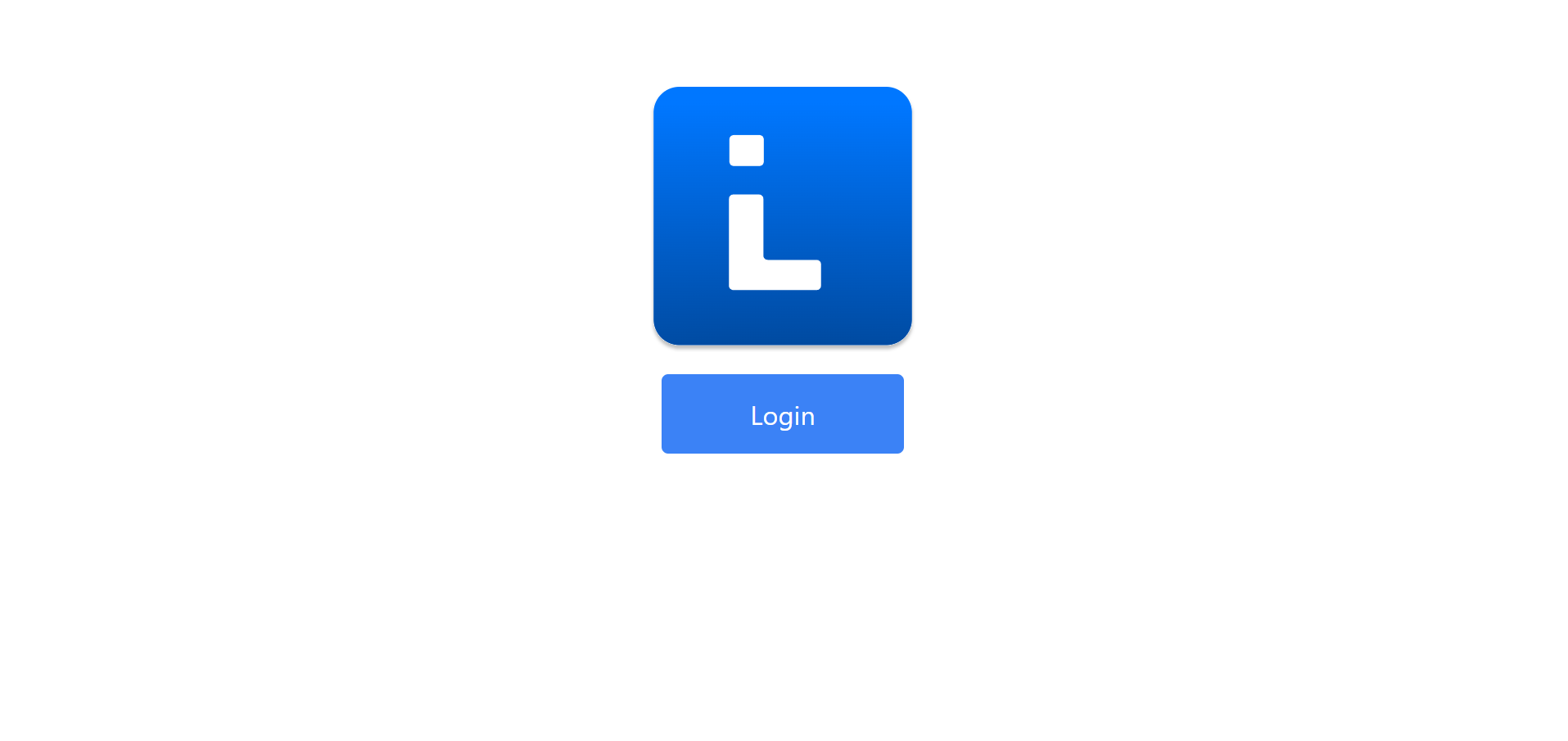


Рисунок 3. Страница для произведения входа

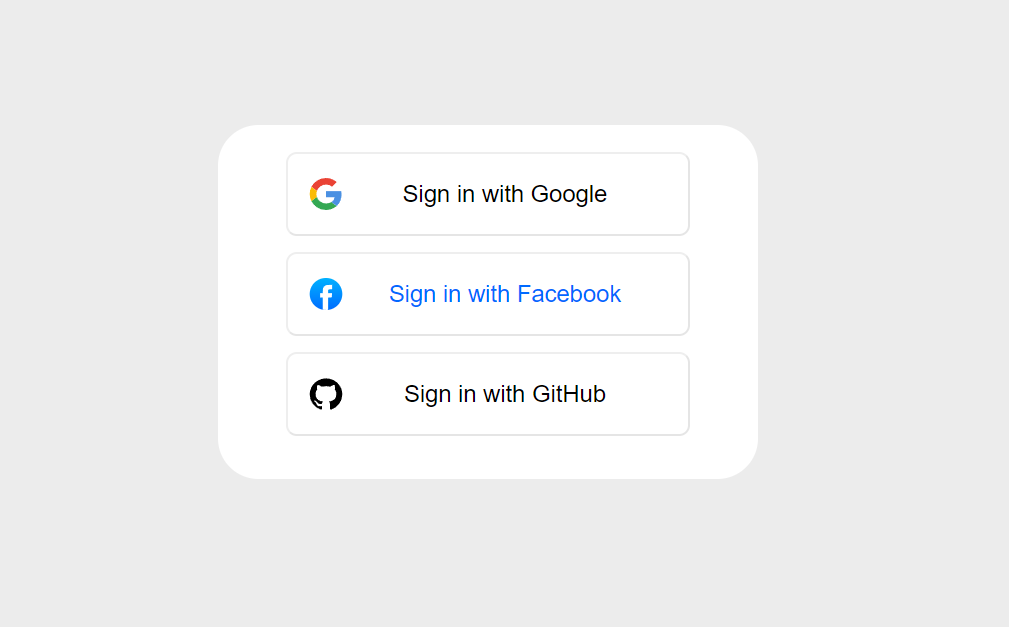


Рисунок 4. Страница выбора социальной сети для авторизации

Так как в файле .env.local хранятся значения только от провайдера Facebook, авторизация возможна только через соответственный сервис.

Для получения своих данных для авторизации через Facebook использовался сервис Meta for Developers.

После открывается основная страница с новостной лентой index.js  
 (рис. 5).

На ней присутствуют такие компоненты как “Меню” (рис.7) и “Header” (рис.6), где присутствуют изображения профиля пользователя, полученные из социальной сети Facebook.

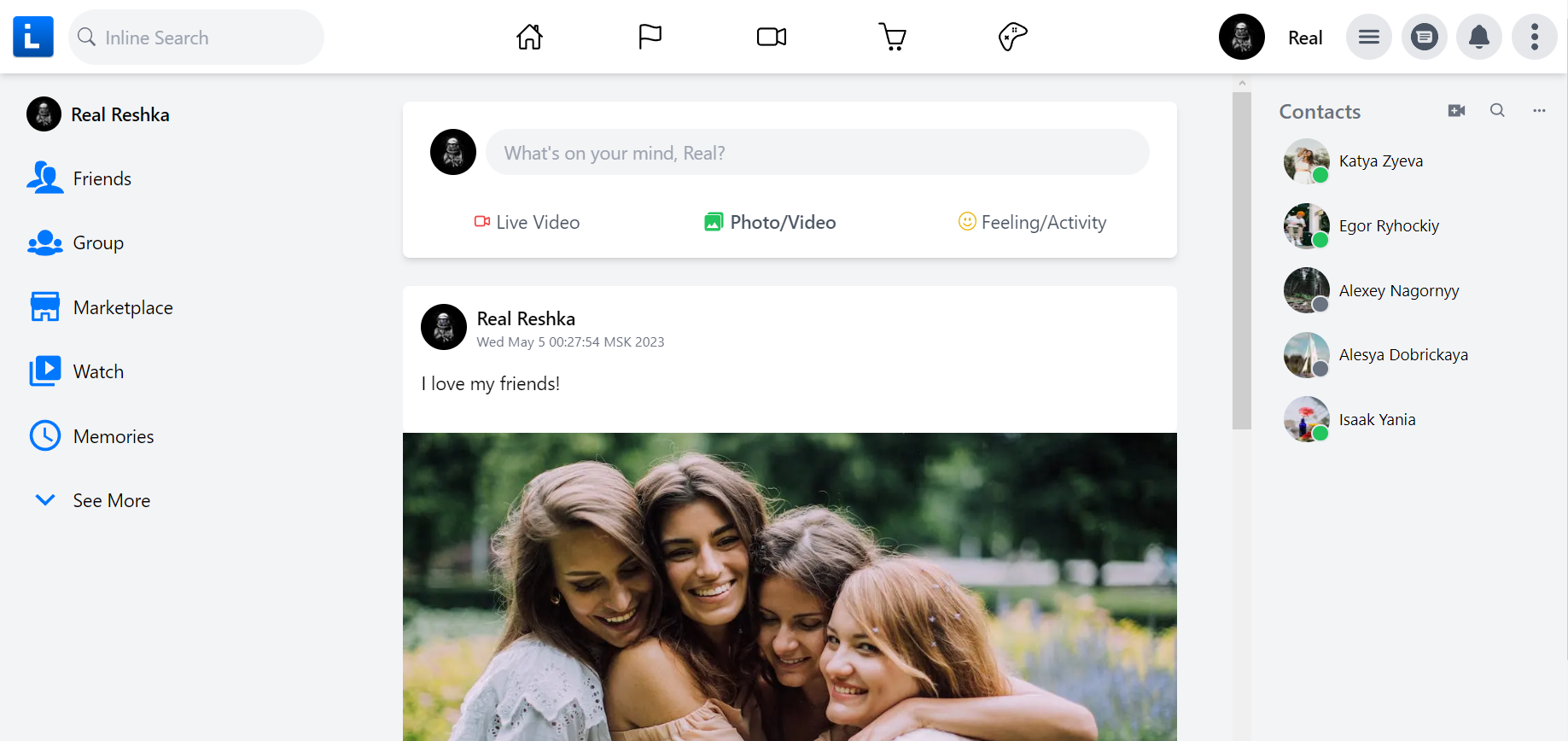


Рисунок 5. Главная страница

При нажатии на изображения пользователя на этих компонентах, можно закончить сессию и произвести выход.



Рисунок 6. Компонент “Header”

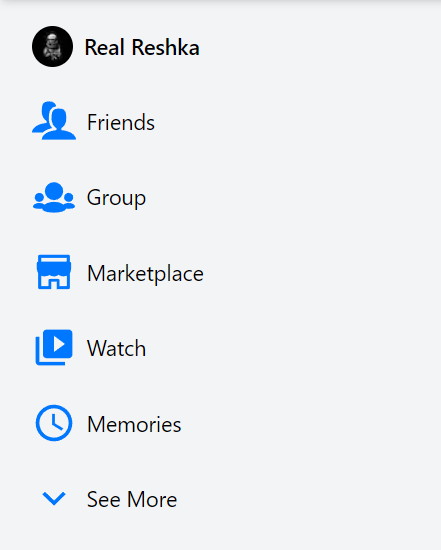


Рисунок 7. Компонент “Меню”

* 1. **Реализация страницы-новостной ленты**

Главная страница с новостной лентой изображена на рисунке 3.На ней присутствуют такие компоненты, как “Header”, “Меню”, “Контакты”, компонент для создания постов и лента постов.

Компонент “Меню” (рис.7) содержит в себе другие страницы веб-приложения.

На компоненте “Контакты” (рис.8) присутствуют контакты пользователя, и, в зависимости от статуса контакта, онлайн человек или нет, отображается индикатор – зелёный или серый соответственно.

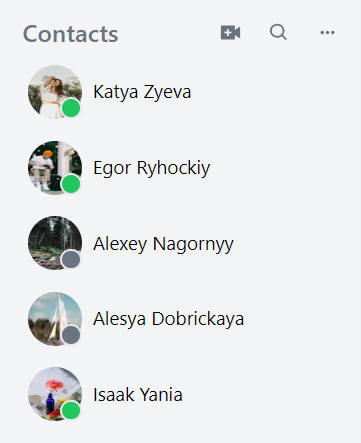


Рисунок 8. Компонент "Контакты"

Компонент для создания постов (рис.9) содержит в себе поле для ввода текста и выбор формата прикрепляемого файла – видео или изображение.

В данном приложении реализовано создание текстовых постов или с прикреплённым изображением.

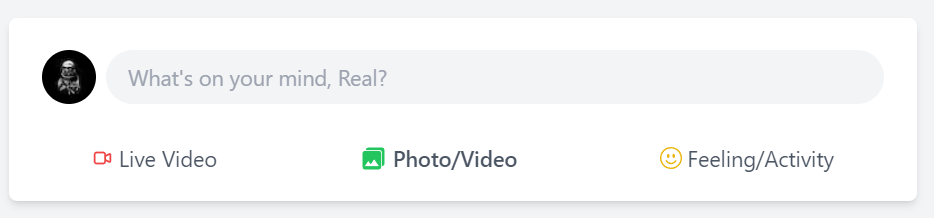


Рисунок 9. Компонент для создания постов

При нажатии на кнопку “Photo/Video” на компоненте для создания постов отображается проводник с файлами (рис.10), где можно выбрать изображения для прикрепления к посту.

После выбора нужного изображения перед публикацией оно будет отображаться в том же компоненте (рис.11). Также там есть кнопка для удаления выбранного изображения.

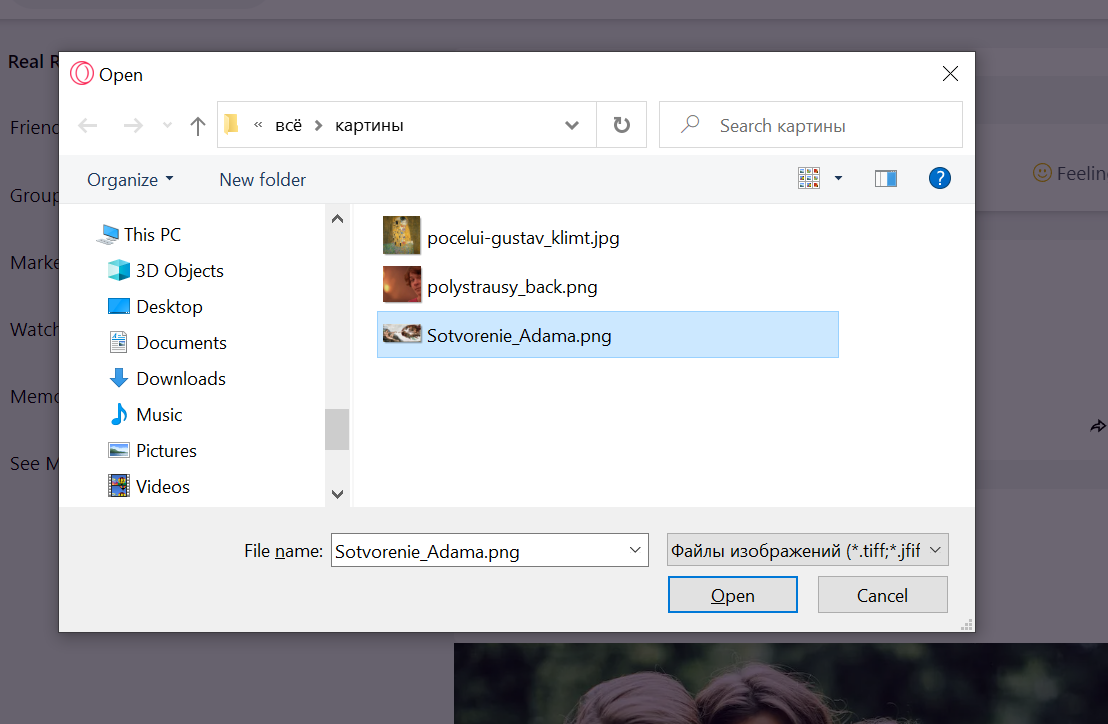


Рисунок 10. Открытие проводника для выбора прикрепляемого изображения

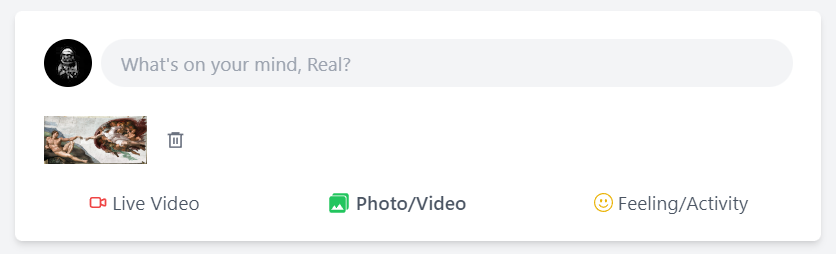


Рисунок 11. Отображение выбранного изображения в компоненте для создания постов

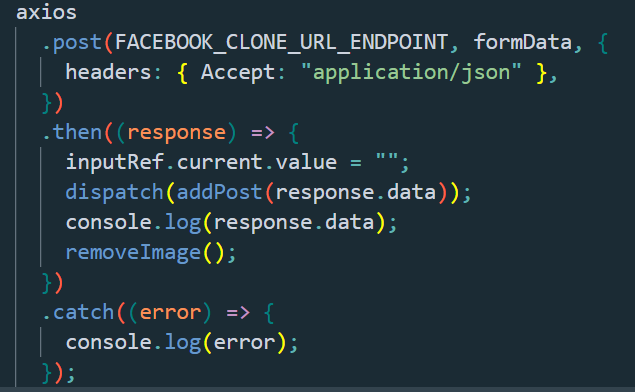
**ГЛАВА 3. ЗАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ ВЕБ-ПРИЛОЖЕНИЯ**

**3.1 Разработка структуры базы данных и передача данных**

Так как в качестве инструмента для разработки серверной части приложения взят фреймворк Spring Data JPA, то создание таблиц базы данных происходит при первом запуске. Однако изначально надо было решить, какие данные нам необходимо.

С помощью axios (Листинг 2) из клиентской части мы получаем мы подключаемся к серверной части приложения и отправляем и получаем оттуда данные.

Листинг 2

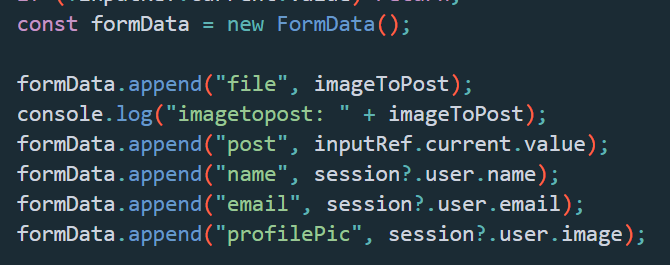


Данные с клиенткой части отправляются на серверную в виде объекта (Листинг 3), в который входят такие поля, как

* file – прикреплённое к посту изображение
* post – текс поста
* name – имя пользователя
* email – email пользователя
* profilePic – фотография профиля пользователя

После получения серверной частью объекта, посредством Spring Data JPA она добавляет такие поля id и время создание поста, связывается с базой данных MySQL и загружает туда данные (Листинг 4).

Листинг 3



Листинг 4



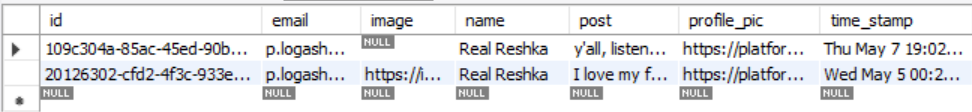


Рисунок 12. Поля базы данных MySQL

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы были изучены такие инструменты для создания веб-приложений, как фреймворк Next.js, библиотека React.js, интернет-протокол для аутентификации OAuth, а так же с их помощью создано веб-приложение, серверная часть которого была реализована с помощью фреймворка Spring Data JPA, а для хранения данных была разработана структура базы данных MySQL.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. React – A JavaScript library for building user interfaces [Электронный ресурс] – Режим доступа: https://reactjs.org/ – Дата доступа: 25.04.2023.
2. Видеоплатформа Udemy [Электронный ресурс] /Java. От простого к сложному. Электрон. дан. — М., Александр Васько, 2023. — URL : [https://www.udemy.com/course/java-simple2advanced/ —](https://www.udemy.com/course/java-simple2advanced/%20—) Дата доступа : 10.03.2023.
3. Социальная сеть Facebook [Электронный ресурс] / https://www.facebook.com — Дата доступа : 22.03.2023.
4. Видеоплатформа YouTube [Электронный ресурс] / Facebook Cloning  [https://www.youtube.com/watch?v=6tRqPTJgcw8/](https://www.youtube.com/watch?v=EHDlebVv6zw/) — Дата доступа : 10.03.2023.

**ПРИЛОЖЕНИЕ. ЛИСТИНГИ ПРОГРАММ**

**Листинг клиентской части приложения**

**components/Contacts.js**

**import React from 'react'**

**import Image from 'next/image'**

**const Contacts = ({name, src, status}) => {**

**return (**

**<div className="flex items-center space-x-2 py-2 pl-1 hover:bg-gray-200 rounded-l-xl cursor-pointer relative">**

**<Image**

**src={src} height={40} width={40}**

**className="rounded-full cursor-pointer" alt='photo'**

**style={{height: 40, width:40,}}**

**/>**

**<p className="hidden sm:inline-flex text-sm">{name}</p>**

**{status === "Online" && (**

**<div className="bg-green-500 h-4 w-4 rounded-full absolute left-5 bottom-2 border-2"></div>**

**)}**

**{status === "Offline" && (**

**<div className="bg-gray-500 h-4 w-4 rounded-full absolute left-5 bottom-2 border-2"></div>**

**)}**

**</div>**

**)**

**}**

**export default Contacts**

**components/CreatePost.js**

**import { React, useState, useRef } from "react";**

**import Image from "next/image";**

**import { useSession } from "next-auth/react";**

**import { HiOutlineVideoCamera } from "react-icons/hi";**

**import { IoMdPhotos } from "react-icons/io";**

**import { BsEmojiSmile } from "react-icons/bs";**

**import { RiDeleteBin6Line } from "react-icons/ri";**

**import { useDispatch } from "react-redux";**

**import axios from "axios";**

**import { addPost } from "../public/src/features/postSlice";**

**const CreatePost = () => {**

**//implementing back to front (and in back it os in PostController.java @CrossOrigin)**

**const FACEBOOK\_CLONE\_URL\_ENDPOINT = "http://localhost:9008/api/v1/post";**

**const { data: session } = useSession();**

**const dispatch = useDispatch(); //чтоб при нажатии отправка файлов шла в слайс (диспатчилась)**

**const inputRef = useRef(null);**

**const hiddenFileInput = useRef(null);**

**const [imageToPost, setImageToPost] = useState(null);**

**const handleClick = () => {**

**hiddenFileInput.current.click();**

**};**

**//функция загрузки поста при нажатии**

**const handleSubmit = (e) => {**

**e.preventDefault();**

**if (!inputRef.current.value) return;**

**const formData = new FormData();**

**formData.append("file", imageToPost);**

**console.log("imagetopost: " + imageToPost);**

**formData.append("post", inputRef.current.value);**

**formData.append("name", session?.user.name);**

**formData.append("email", session?.user.email);**

**formData.append("profilePic", session?.user.image);**

**//при клике, вся инфа о профиле, о контенте поста опубликовывается,посту даётся URL**

**//затем респонсом чистится поле для создания постов (текст и/или фото)**

**// и можно закетчить ошибку**

**axios**

**.post(FACEBOOK\_CLONE\_URL\_ENDPOINT, formData, {**

**headers: { Accept: "application/json" },**

**})**

**.then((response) => {**

**inputRef.current.value = "";**

**dispatch(addPost(response.data));**

**console.log(response.data);**

**removeImage();**

**})**

**.catch((error) => {**

**console.log(error);**

**});**

**};**

**// картинка появляется в посте**

**const addImageToPost = (e) => {**

**const reader = new FileReader();**

**if (e.target.files[0]) {**

**reader.readAsDataURL(e.target.files[0]);**

**reader.onload = (e) => {**

**setImageToPost(e.target.result);**

**};**

**}**

**};**

**//картинка удаляется из поста**

**const removeImage = () => {**

**setImageToPost(null);**

**};**

**return (**

**<div className="bg-white rounded-md shadow-md text-gray-500 p-2">**

**<div className="flex p-4 space-x-2 item">**

**<Image**

**src={session?.user.image}**

**alt="inline logo"**

**height={40}**

**width={40}**

**className="rounded-full cursor-pointer"**

**/>**

**<form className="flex flex-1">**

**<input**

**className="rounded-full h-10 flex-grow focus:outline-none font-regular bg-gray-100 px-4"**

**type="text"**

**ref={inputRef}**

**placeholder={`What's on your mind, ${**

**session?.user.name.split(" ")[0]**

**}?`}**

**></input>**

**<button hidden onClick={handleSubmit}></button>**

**</form>**

**</div>**

**{imageToPost && (**

**<div**

**onClick={removeImage}**

**className="flex items-center px-4 py-2 space-x-4 filter hover:brightness-110 transotion duration-150 cursor pointer"**

**>**

**<img src={imageToPost} className="h-10 object-contain" />**

**<RiDeleteBin6Line className="h-8 hover:text-red-500" />**

**</div>**

**)}**

**<div className="flex justify-evenly py-2">**

**<div className="flex items-center p-1 space-x-1 flex-grow justify-center hover:cursor-pointer hover:bg-gray-100 rounded-md">**

**<HiOutlineVideoCamera sixe={20} className="text-red-500" />**

**<p className="font-regular text-gray-600">Live Video</p>**

**</div>**

**<div**

**onClick={handleClick}**

**className="flex items-center p-1 space-x-1 flex-grow justify-center hover:cursor-pointer hover:bg-gray-100 rounded-md">**

**<IoMdPhotos className="text-green-500" size={20} />**

**<p className="font-semibold text-gray-600">Photo/Video</p>**

**<input**

**ref={hiddenFileInput}**

**onChange={addImageToPost}**

**type="file"**

**accept="image/\*"**

**hidden**

**/>**

**</div>**

**<div className="flex items-center p-1 space-x-1 flex-grow justify-center hover:cursor-pointer hover:bg-gray-100 rounded-md">**

**<BsEmojiSmile sixe={20} className="text-yellow-500" />**

**<p className="font-regular text-gray-600">Feeling/Activity</p>**

**</div>**

**</div>**

**</div>**

**);**

**};**

**export default CreatePost;**

**components/Post.js**

**import axios from "axios";**

**import React, { useEffect } from "react";**

**import { useDispatch, useSelector } from "react-redux";**

**import { addAllPost, selectPost } from "../public/src/features/postSlice";**

**import Post from "./Post";**

**const Posts = () => {**

**const dispatch = useDispatch();**

**const posts = useSelector(selectPost);**

**useEffect(() => {**

**const fetchData = () => {**

**const response = axios**

**.get("http://localhost:9008/api/v1/post")**

**.then((response) => {**

**console.log(response.data);**

**dispatch(addAllPost(response.data));**

**});**

**};**

**fetchData();**

**console.log(posts);**

**}, []);**

**return (**

**<div>**

**{posts.map((post) => (**

**<Post post={post} key={post.id} />**

**))}**

**</div>**

**);**

**};**

**export default Posts;**

**Листинг серверной части приложения**

**src/main/java/com/inline/facebookcloneservice/controller/  
PostController.java**

**package com.inline.facebookcloneservice.service;**

**import com.inline.facebookcloneservice.entity.PostEntity;**

**import com.inline.facebookcloneservice.model.Post;**

**import com.inline.facebookcloneservice.repository.PostEntityRepository;**

**import org.springframework.beans.BeanUtils;**

**import org.springframework.stereotype.Service;**

**import java.util.ArrayList;**

**import java.util.List;**

**import java.util.stream.Collectors;**

**@Service**

**public class PostServiceImpl implements PostService{**

**private PostEntityRepository postEntityRepository;**

**public PostServiceImpl(PostEntityRepository postEntityRepository) {**

**this.postEntityRepository = postEntityRepository;**

**}**

**@Override**

**public Post addPost(Post post) throws Exception {**

**try{**

**PostEntity postEntity = new PostEntity();**

**BeanUtils.copyProperties(post, postEntity);**

**if(post.getFile() !=null && !post.getFile().equalsIgnoreCase("null"))**

**postEntity.setImage(post.getFile());**

**else postEntity.setImage(null);**

**postEntity = postEntityRepository.save(postEntity);**

**post.setId(postEntity.getId());**

**post.setFile(null);**

**post.setImage(postEntity.getImage());**

**}catch(Exception e){**

**throw new Exception("Couldn't save Post" + e);**

**}**

**return post;**

**}**

**@Override**

**public List<Post> getPost() {**

**List<PostEntity> postEntities**

**= postEntityRepository.findAll();**

**List<Post> posts = new ArrayList<>();**

**posts = postEntities.stream()**

**.map((postEntity) ->**

**Post.builder()**

**.id(postEntity.getId())**

**.timeStamp(postEntity.getTimeStamp())**

**.email(postEntity.getEmail())**

**.name(postEntity.getName())**

**.post(postEntity.getPost())**

**.image(postEntity.getImage())**

**.profilePic(postEntity.getProfilePic())**

**.build()**

**).collect(Collectors.toList());**

**return posts;**

**}**

**}**

**src/main/java/com/inline/facebookcloneservice/service/  
PostServiceImpl.java**

**package com.inline.facebookcloneservice.service;**

**import com.inline.facebookcloneservice.entity.PostEntity;**

**import com.inline.facebookcloneservice.model.Post;**

**import com.inline.facebookcloneservice.repository.PostEntityRepository;**

**import org.springframework.beans.BeanUtils;**

**import org.springframework.stereotype.Service;**

**import java.util.ArrayList;**

**import java.util.List;**

**import java.util.stream.Collectors;**

**@Service**

**public class PostServiceImpl implements PostService{**

**private PostEntityRepository postEntityRepository;**

**public PostServiceImpl(PostEntityRepository postEntityRepository) {**

**this.postEntityRepository = postEntityRepository;**

**}**

**@Override**

**public Post addPost(Post post) throws Exception {**

**try{**

**PostEntity postEntity = new PostEntity();**

**BeanUtils.copyProperties(post, postEntity);**

**if(post.getFile() !=null && !post.getFile().equalsIgnoreCase("null"))**

**postEntity.setImage(post.getFile());**

**else postEntity.setImage(null);**

**postEntity = postEntityRepository.save(postEntity);**

**post.setId(postEntity.getId());**

**post.setFile(null);**

**post.setImage(postEntity.getImage());**

**}catch(Exception e){**

**throw new Exception("Couldn't save Post" + e);**

**}**

**return post;**

**}**

**@Override**

**public List<Post> getPost() {**

**List<PostEntity> postEntities**

**= postEntityRepository.findAll();**

**List<Post> posts = new ArrayList<>();**

**posts = postEntities.stream()**

**.map((postEntity) ->**

**Post.builder()**

**.id(postEntity.getId())**

**.timeStamp(postEntity.getTimeStamp())**

**.email(postEntity.getEmail())**

**.name(postEntity.getName())**

**.post(postEntity.getPost())**

**.image(postEntity.getImage())**

**.profilePic(postEntity.getProfilePic())**

**.build()**

**).collect(Collectors.toList());**

**return posts;**

**}**

**}**