Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное   
образовательное учреждение высшего образования   
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторным работам

Дисциплина: «Технологии сетевого программирования»

Тема «Проектирование приложения»

Выполнили: Фатин А.В., Кондорова А.А.

Группа: 6302-010302D

Самара, 2025

Содержание

[Введение 3](#_Toc199173100)

[1 Проектирование приложения 4](#_Toc199173101)

[1.1 Схема взаимодействия компонентов 4](#_Toc199173102)

[1.1.1 Возможности пользователя 4](#_Toc199173103)

[1.1.2 Возможности администратора 4](#_Toc199173104)

[1.2 Логическая схема БД 4](#_Toc199173105)

[2 Описание API 5](#_Toc199173106)

[3 Тестирование API 6](#_Toc199173107)

[4 Аутентификация и авторизация 7](#_Toc199173108)

[4.1 Аутентифиакция 7](#_Toc199173109)

[4.2 Авторизация 7](#_Toc199173110)

[4.3 Безопасность 7](#_Toc199173111)

[5 Структура пользовательского интерфейса 9](#_Toc199173112)

[6 Описание структуры контейнеризации и настройки окружения 10](#_Toc199173113)

[6.1 Общая архитектура контейнеров 10](#_Toc199173114)

[6.2 Docker-compose 10](#_Toc199173115)

[7 Демонстрация работы контейнеров и приложения 12](#_Toc199173116)

[7.1 Запуск приложения 12](#_Toc199173117)

[7.2 Взаимодействие с приложением 12](#_Toc199173118)

[Приложение А 15](#_Toc199173119)

Введение

Наше веб-приложение позволяет публиковать свои посты, подписываться на других пользователей и следить за их постами. Приложение реализовано на базе стека технологий Java 21 + Spring Boot + PostgreSQL с использованием server-side рендеринга через Thymeleaf. Вся система контейнеризирована с помощью Docker.

Проект разрабатывался как учебно-практический, с целью демонстрации навыков в области серверной разработки, проектирования структуры приложения, настройки безопасной аутентификации, работы с базой данных, а также интеграции компонентов в контейнеризированное окружение.

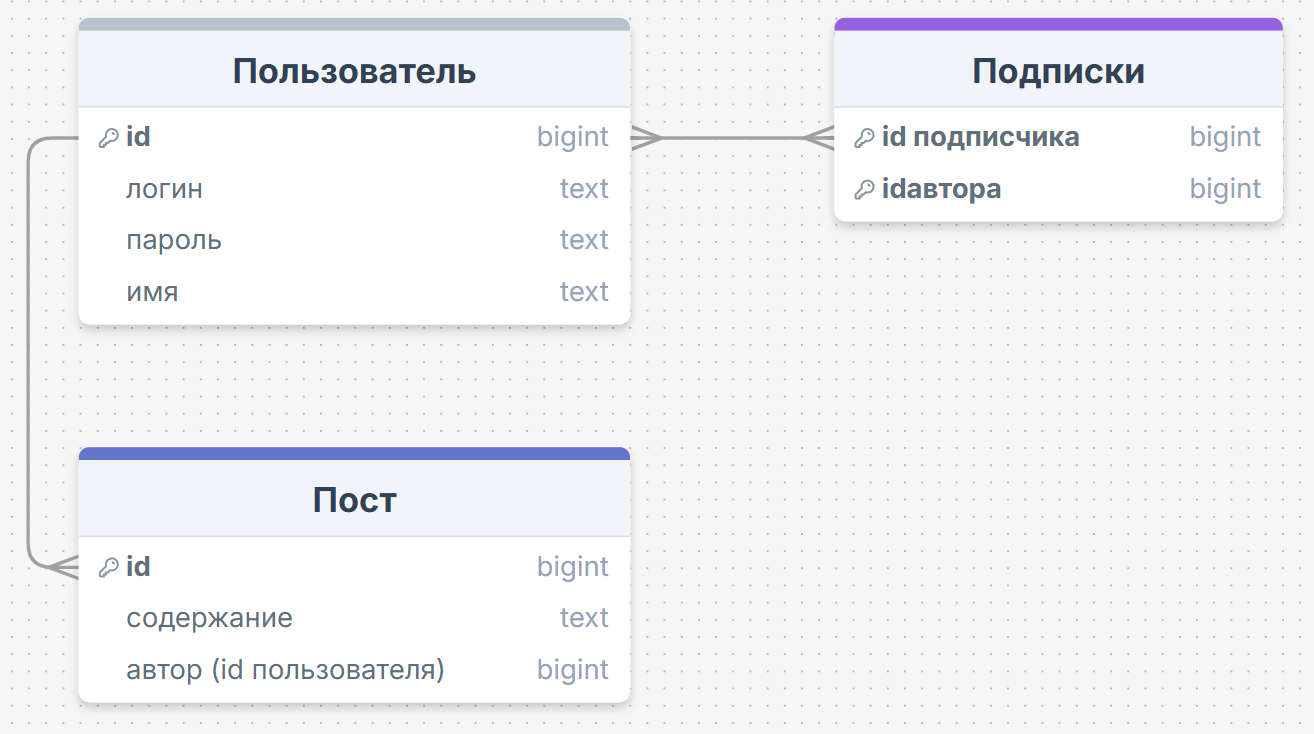
1. Проектирование приложения
   1. Схема взаимодействия компонентов

Клиент отправляет запросы к API (например, вход в систему, публикация постов, подписка на пользователей).

Клиент отправляет запросы к API (например, вход в систему, публикация постов, подписка на пользователей).

База данных (PostgreSQL) хранит пользователей, подписки, посты и обеспечивает связь между ними.

* + 1. Возможности пользователя
* Добавление постов к себе на стену;
* Подписка на других пользователей;
* Просмотр постов авторов, на которых подписан.
  + 1. Возможности администратора
* Просмотр всех пользователей и их постов;
* Удаление любых постов и пользователей.
  1. Логическая схема БД



1. Описание API

В таблице 1 представлены основные эндпоинты API.

Таблица 1 – Описание API

|  |  |  |
| --- | --- | --- |
| Метод | URL | Описание |
| POST | /api/login | Аутентификация пользователя |
| POST | /api/register | Регистрация пользователя |
| POST | /api/users | Создание пользователя |
| GET | /api/users/search?name={name} | Получить пользователя по имени |
| GET | /api/users/{id} | Получить пользователя по id |
| GET | /api/users | Вывести всех пользователей |
| GET | /api/users/{id}/posts | Вывести все посты пользователя |
| GET | /api/posts | Вывести все посты |
| POST | /api/posts | Создать пост |
| GET | /api/users/{id}/all\_posts | Все посты авторов, на которых подписан пользователь |
| POST | /api/subscriptions | Подписаться |
| DELETE | /api/follow/{id} | Отписаться |

1. Тестирование API

Для проверки корректности работы серверной части веб-приложения проводилось ручное тестирование API с использованием инструмента Postman. Несмотря на то, что клиентский интерфейс реализован с помощью серверного рендеринга через Thymeleaf, каждый основной функционал был дополнительно проверен через вызовы соответствующих HTTP-запросов.

Процесс тестирования включал следующие аспекты:

* Проверка регистрации и авторизации пользователей;
* Создание и удаление постов;
* Получение списков пользователей и их постов;
* Обработка ошибок (например, неправильный формат данных, отсутствие прав доступа и др.).

На каждом этапе проверялись:

* Корректность ответа (HTTP-статус, тело ответа, сообщения об ошибках);
* Защита маршрутов с использованием JWT-токенов.

1. Аутентификация и авторизация

Веб-приложение использует механизм JWT для управления доступом пользователей. Аутентификация и авторизация реализованы на стороне backend-приложения с использованием Spring Security.

* 1. Аутентифиакция

При входе в систему пользователь отправляет запрос POST /api/auth/login, передавая свои учётные данные (логин и пароль). В случае успешной аутентификации сервер формирует JWT-токен, который возвращается в теле ответа. Этот токен представляет собой зашифрованную строку, содержащую информацию о пользователе (ID, роль и срок действия), и служит ключом доступа к защищённым ресурсам.

При последующих запросах клиент должен указывать токен в заголовке Authorization: Bearer <токен> или в cookies. Без валидного токена доступ к защищённым маршрутам будет запрещён.

* 1. Авторизация

Каждому пользователю при регистрации присваивается роль:

* CLIENT — клиент, имеющий доступ к просмотру своей страницы, чужим постам и полному списку пользователей;
* ADMIN — администратор, который может удалять любые пост и пользователей.

На основе роли пользователя backend фильтрует доступ к маршрутам, используя аннотации и конфигурации Spring Security. Основная настройка осуществляется в классе SecurityConfig, где определяются правила доступа к API-эндпоинтам с использованием механизма URL-фильтрации и ограничения по ролям.

* 1. Безопасность

Для хранения паролей используется хэширование с BCrypt, что исключает хранение паролей в открытом виде в базе данных. Также реализованы следующие меры:

* Проверка уникальности login при регистрации;
* Валидация входных данных;
* Ограничение доступа к маршрутам с помощью ролей и токенов.

Такой подход обеспечивает надежную защиту данных пользователей и изоляцию функциональности между ролями в приложении.

1. Структура пользовательского интерфейса

Пользовательский интерфейс веб-приложения реализован с использованием Thymeleaf. Интерфейс построен по классической MVC-модели и рендерится на стороне сервера, что обеспечивает простоту интеграции с Spring Boot и улучшает контроль над безопасностью.

Приложение разделено на логически обособленные пользовательские интерфейсы. Основные страницы и компоненты:

* Регистрация и авторизация – отдельные формы для клиентов с соответствующей валидацией и отображением ошибок.
* Домашняя страница пользователя – позволяет написать новый пост, просматривать и удалять существующие посты, переходить в настройки.
* Настройки – изменение пароля, выход из аккаунта, удаление аккаунта.
* Страница всех пользователей – список всех пользователей, с возможностью подписываться и отписываться от них.
* Лента постов – сюда попадают посты авторов, на которых подписан конкретный пользователь.
* Панель администратора – отображает список всех пользователей, с возможностью их удалять, также возможность просматривать посты каждого пользователя.

Навигация между страницами обеспечивается через обычные переходы между маршрутами.

Такой подход обеспечивает простой, но функциональный пользовательский опыт, минимизируя зависимость от внешних библиотек и облегчая отладку в рамках серверного рендеринга.

1. Описание структуры контейнеризации и настройки окружения
   1. Общая архитектура контейнеров

Приложение полностью контейнеризировано с использованием Docker, что обеспечивает изоляцию компонентов, удобство развертывания и масштабируемость.

Архитектура включает два основных контейнера:

1. PostgreSQL — контейнер с официальным образом PostgreSQL 15, в котором развёрнута база данных приложения. Использует том для постоянного хранения данных между перезапусками.
2. Backend — контейнер с Spring Boot приложением, построенный на базе образа 3.9.6-eclipse-temurin-17. Содержит собранный .jar файл, который запускается при старте контейнера.

Контейнеры объединены в одну виртуальную сеть с типом bridge, что позволяет им безопасно взаимодействовать друг с другом внутри одной среды исполнения.

* 1. Docker-compose

Для управления контейнерами используется docker-compose.yml (рисунок 1), который описывает конфигурацию всех сервисов и их связи. Ключевые особенности:

* Переменные окружения используются для настройки доступа к базе данных (имя БД, пользователь, пароль), передаваемые как в контейнер PostgreSQL, так и в backend.
* Зависимости (depends\_on) гарантируют корректный порядок запуска: сначала база данных, затем backend.
* Проброс портов обеспечивает доступ к backend через порт 9090, а к PostgreSQL — через 5432.
* Сборка backend выполняется из локальной директории ./backend с использованием собственного Dockerfile.



Рисунок 1 - docker-compose.yml

1. Демонстрация работы контейнеров и приложения
   1. Запуск приложения

Для запуска всего приложения достаточно выполнить одну команду:

docker-compose up --build

После этого:

* Контейнер PostgreSQL разворачивается с заданными параметрами базы данных;
* Контейнер Backend автоматически подключается к базе и запускает Spring Boot приложение;
* Приложение становится доступно по адресу http://localhost:9090
  1. Взаимодействие с приложением

Пользователь может перейти по адресу http://localhost:9090, где доступен web-интерфейс приложения, реализованный с использованием Thymeleaf.

На рисунках 2-5 показана работа контейнеров и приложения.

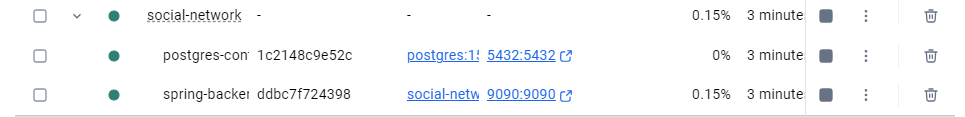


Рисунок 2 – Работа контейнеров.

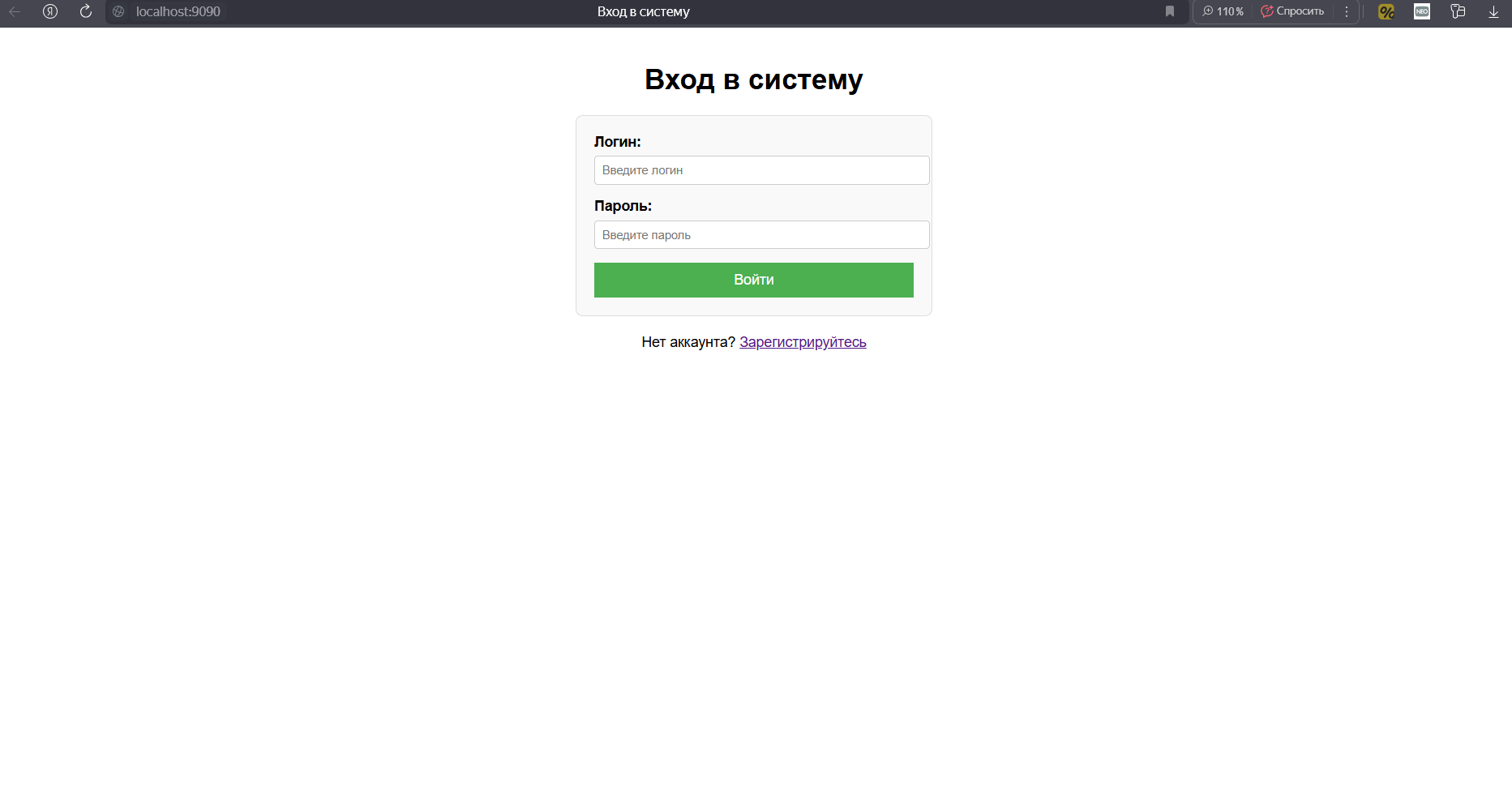


Рисунок 2 – Страница входа.

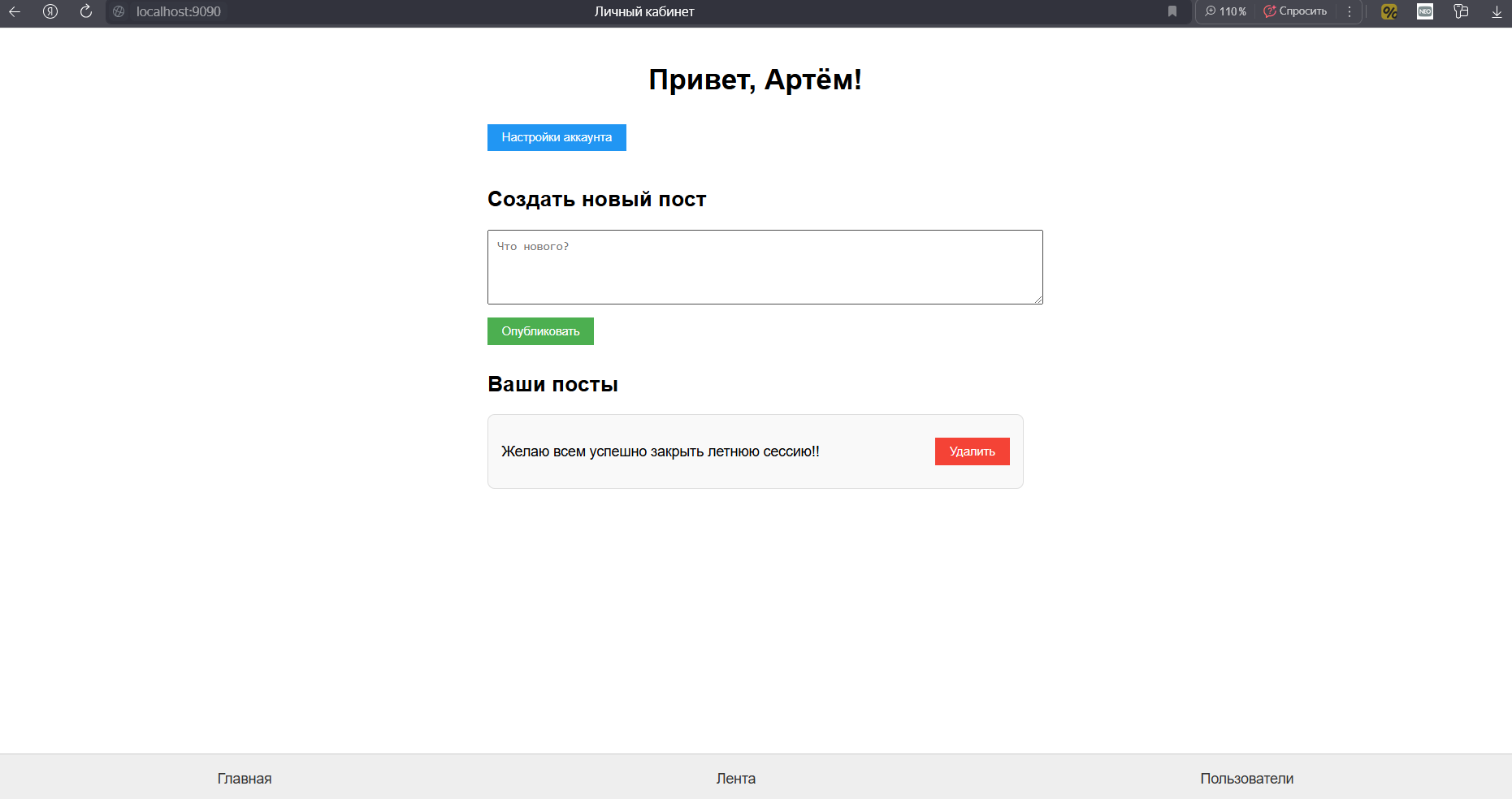


Рисунок 3 – Домашняя страница пользователя.

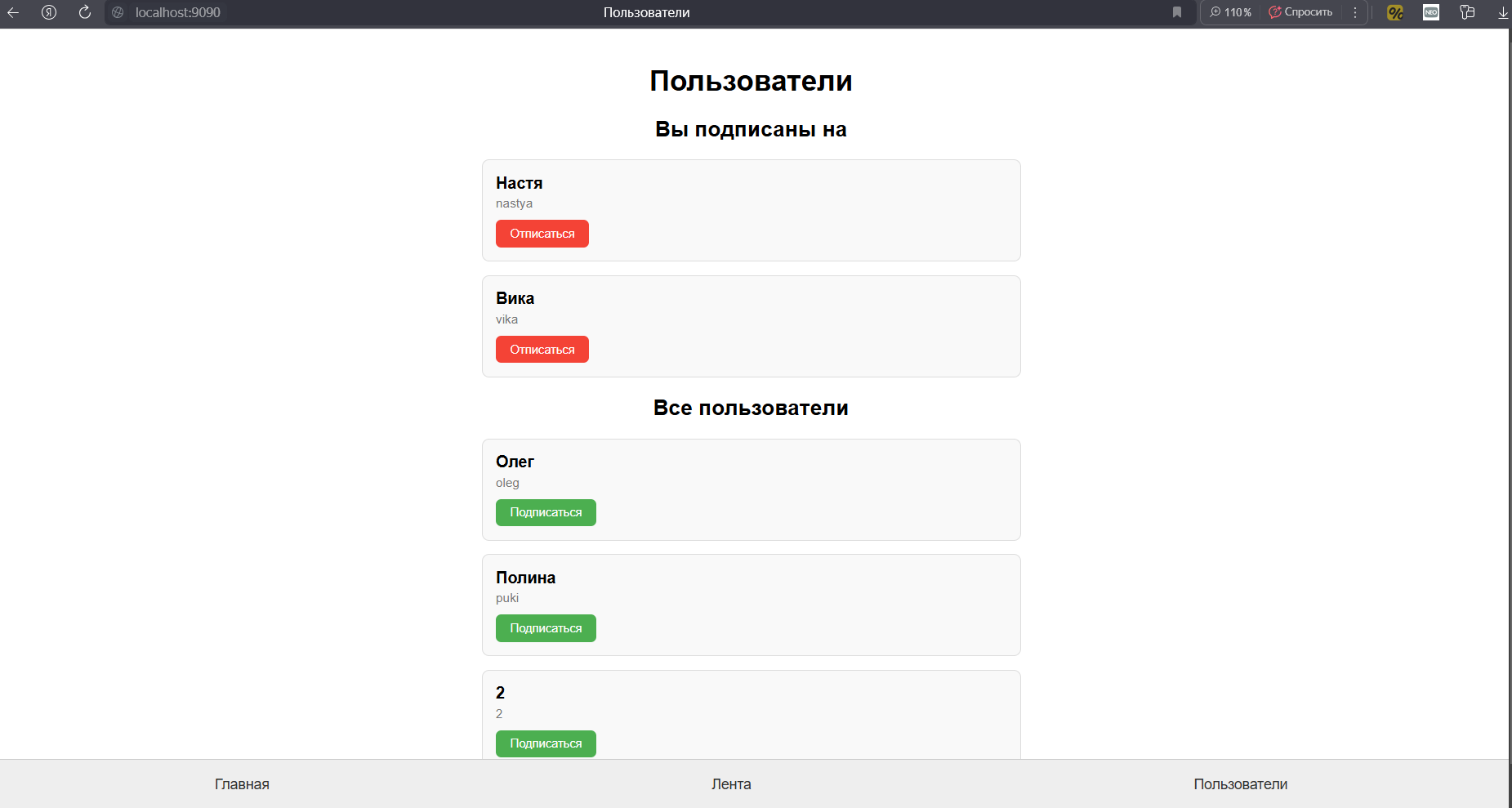


Рисунок 4 – Список всех пользователей.

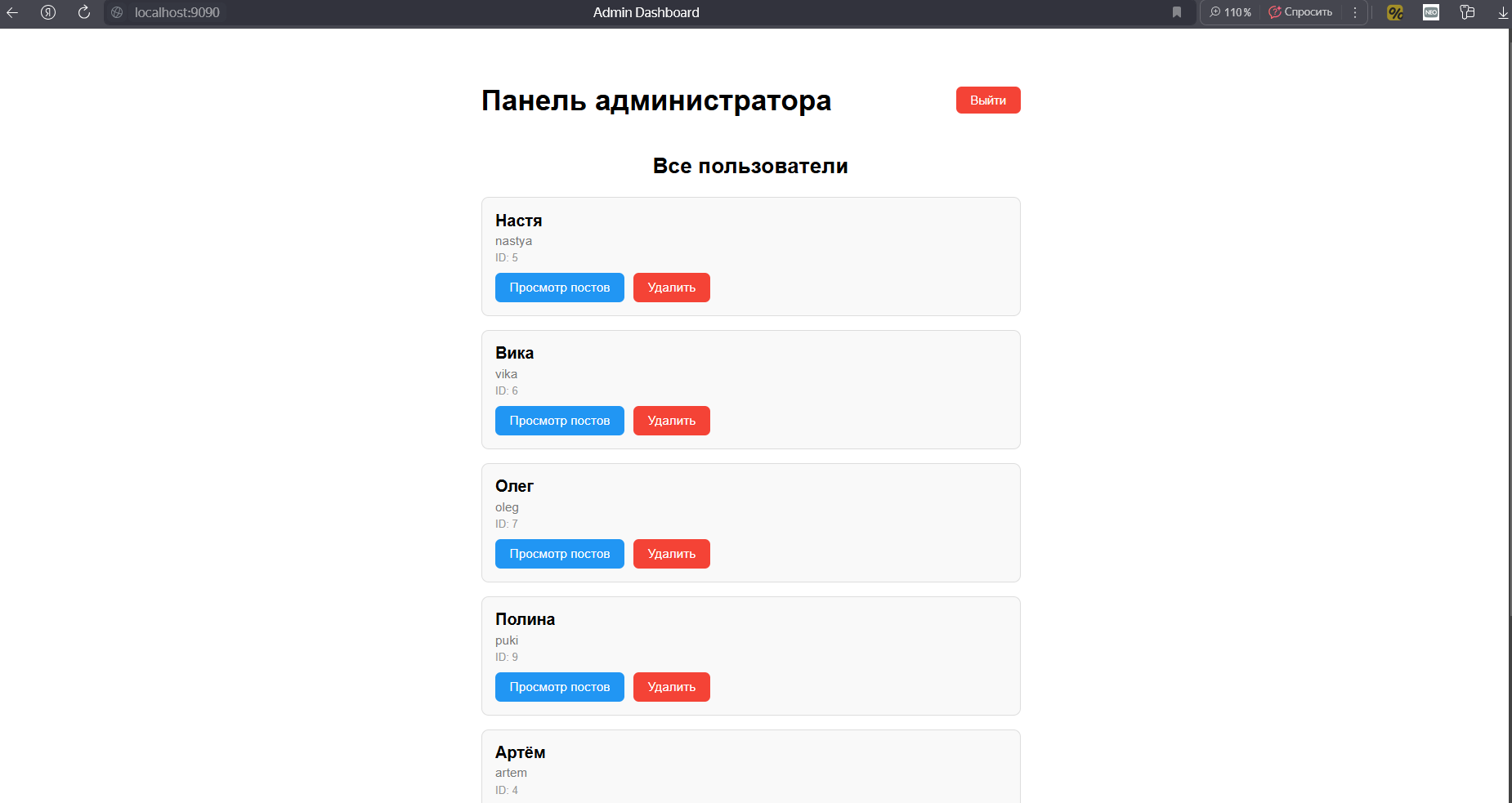


Рисунок 5 – Панель администратора.

Приложение А

Ссылка на GitHub со всем проектом:

<https://github.com/afatin/social-network>