



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01

ОТЧЕТ

по зачетной работе _

Название: Веб-приложение Post. Аналог твиттера

Дисциплина: Языки интернет-программирования

Студент ИУ6-33Б
(Группа)

(Подпись, дата)

В.К. Залыгин
(И.О. Фамилия)

Преподаватель

(Подпись, дата)

Э.Р. Маняшев
(И.О. Фамилия)

Вариант 15
Москва, 2024

Цель

Реализовать веб-приложение – аналог твиттера с следующей функциональностью: авторизация по паре логин/пароль, написание постов с опциональной ссылкой на другие посты, удаление постов, просмотр постов в общей ленте, просмотр постов конкретного пользователя, интернационализация (ru/eng) –, с использованием следующего стека технологий: Kotlin, Spring, Postgres, JS, React, react-router.

Задание

Основная задача зачётной работы - продемонстрировать полученные знания в создании собственного веб-приложения. **Тему зачётной работы допустимо выбрать самостоятельно, но обязательно согласовать с преподавателем.**

Обязательные требования к программе:

- Для реализации использовать Ruby on Rails.
- Необходимо иметь контроллеры, обеспечивающие обработку запросов.
- Необходимо использовать модели для хранения данных в БД.
- Необходимо обеспечить аутентификацию пользователей.
- При реализации клиентской части необходимо применить код на языке Javascript и таблицы стилей CSS.
- Провести интернационализацию приложения и обеспечить вывод надписей на русском языке (см. пример в лекции 11).

Результат приложить в виде двух файлов:

1. архив, содержащий RoR-приложение;
2. pdf-отчет, в котором должны присутствовать фрагменты добавленного кода.

Приложение должно содержать **полный набор тестов**, позволяющих проверить все аспекты его функционирования.

Архив приложения можно подготовить командами:

```
git add . # есть риск собрать здесь мусор, не отмеченный в .gitignore!  
git commit -m "initial"  
git archive master | bzip2 > STUDENT_NAME_final_work.tar.bz2
```

Отчет должен содержать:

- ФИО, номер группы;
- чётко сформулированную задачу, выполняемую в работе;
- список файлов, которые были созданы или модифицированы в процессе выполнения работы, а также исходный код этих файлов;
- изображения форм интерфейса пользователя;

- исходный код тестов и описание цели тестирования каждым конкретным тестом (допустимо в виде комментариев в коде программ тестирования);
- отчет Rubosor по всем контроллерам.

Выполнение Интерфейс приложения

Posting app

[vz](#)

Выйти

Переключить на английский

Написать пост

Новый пост!

Название

Привет, мир!

Содержание

Мой первый пост

Создать пост

Рисунок 1 – страница написания нового поста

Posting app

[vz](#)

Выйти

Переключить на английский



Написать пост

2024-01-19T20:31:16 9065ad11-084c-4dd2-9e7b-e55a32420ebe @vz (R)

<= 4438f774-9b9b-499c-b3c2-a42b677d0fff

Ответ на пост "Привет, мир!"

И тебе привет!



2024-01-19T20:30:54 4438f774-9b9b-499c-b3c2-a42b677d0fff @vz (R)

=> 9065ad11-084c-4dd2-9e7b-e55a32420ebe

Привет, мир!

Мой первый пост





Рисунок 2 – страница с лентой

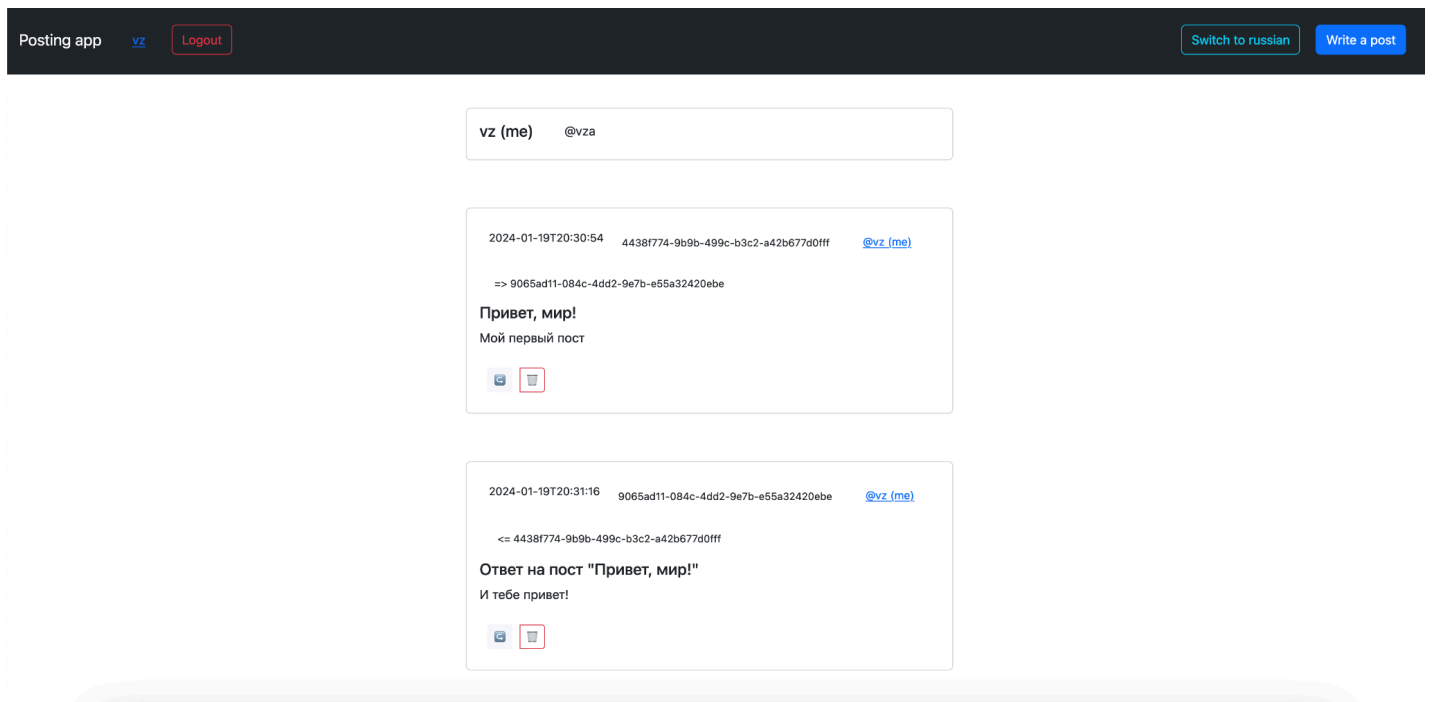


Рисунок 3 – профиль пользователя

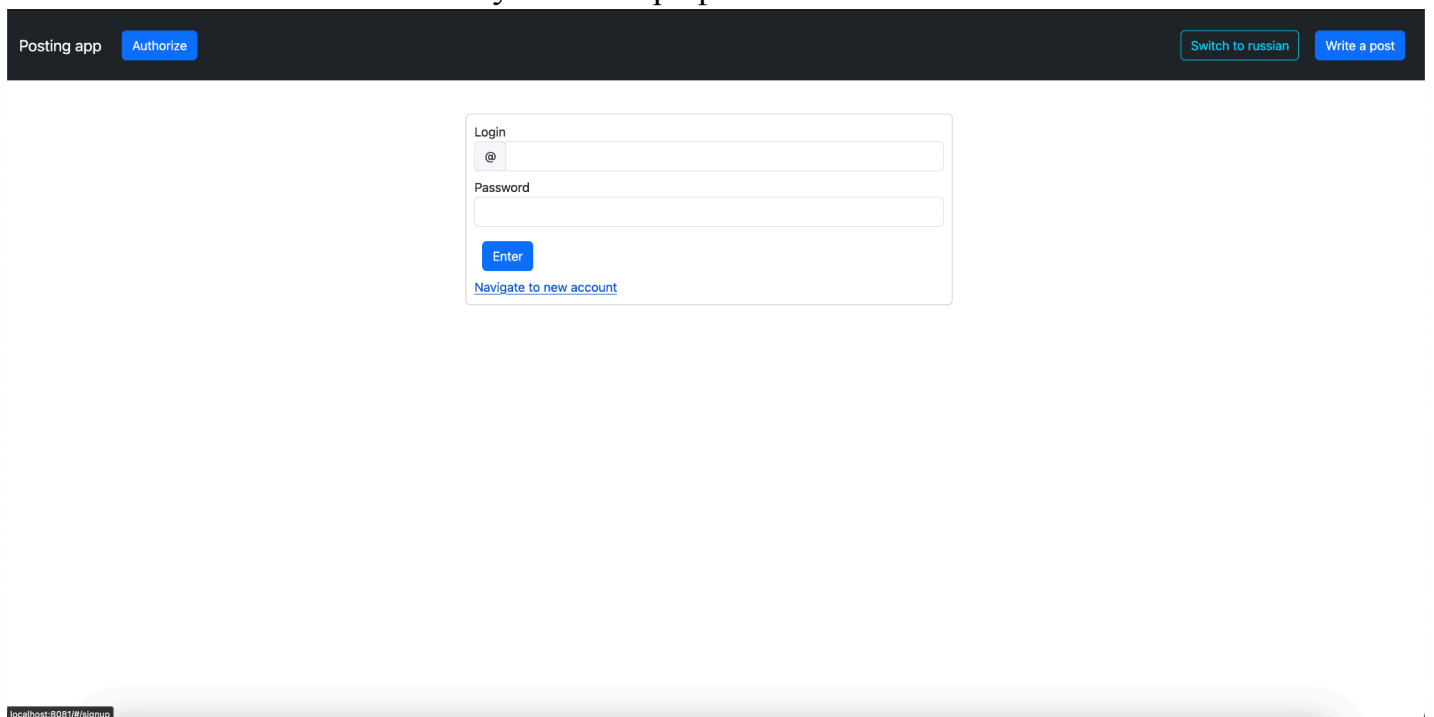


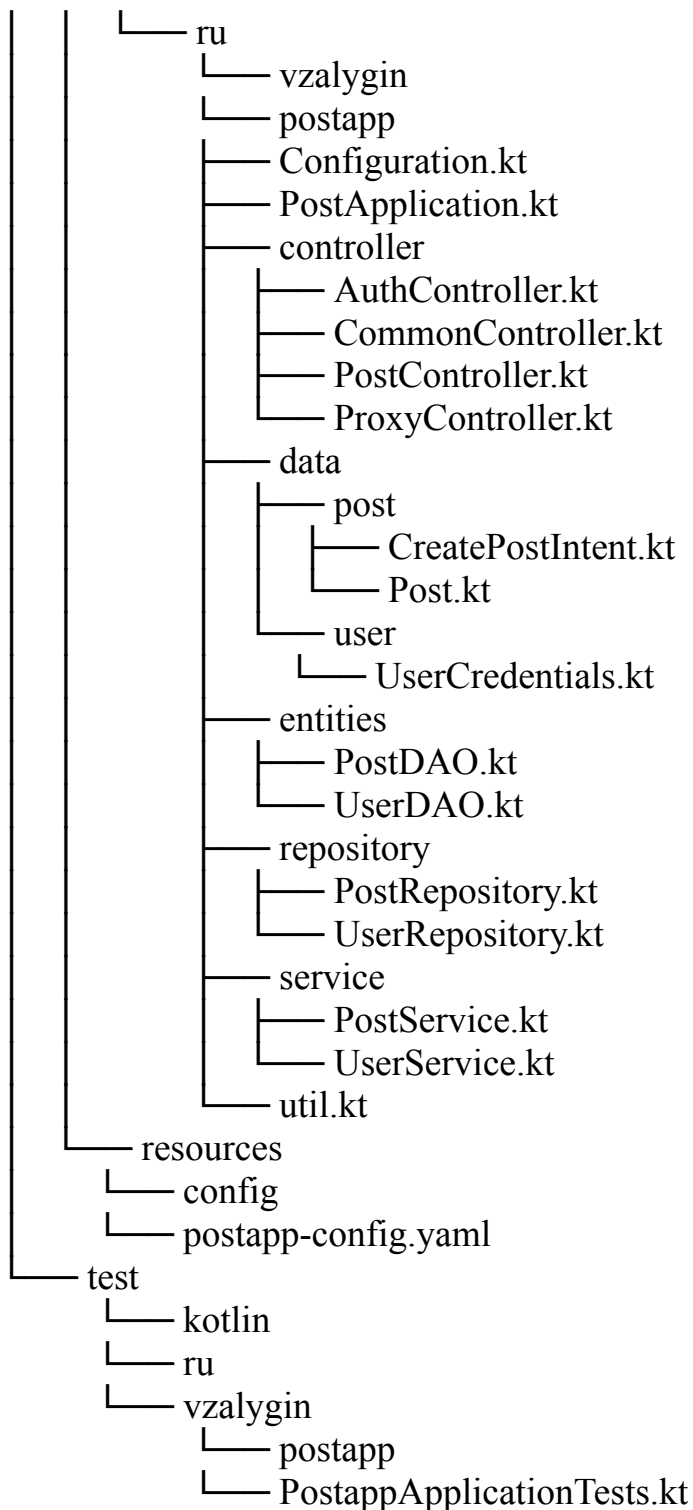
Рисунок 4 – форма авторизации

Backend

Серверная часть реализована с использованием языка Kotlin, веб-фреймворка Spring (и вспомогательного SpringBoot), СУБД Postgres.

Структура проекта:

```
├── main
└── kotlin
```



Configuration.kt

```
package ru.vzalygin.postapp

import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.context.annotation.Bean
import org.springframework.context.annotation.PropertySource
import org.springframework.security.config.annotation.web.builders.HttpSecurity
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity
import org.springframework.security.config.annotation.web.invoke
import org.springframework.security.config.http.SessionCreationPolicy
```

```

import org.springframework.security.core.userdetails.User
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
import org.springframework.security.crypto.password.PasswordEncoder
import org.springframework.security.provisioning.InMemoryUserDetailsManager
import org.springframework.security.provisioning.UserDetailsManager
import org.springframework.security.web.SecurityFilterChain
import org.springframework.security.web.authentication.www.BasicAuthenticationEntryPoint
import org.springframework.web.cors.CorsConfiguration
import org.springframework.web.cors.CorsConfigurationSource
import org.springframework.web.cors.UrlBasedCorsConfigurationSource
import javax.sql.DataSource

@SpringBootApplication
@PropertySource("classpath:/config/postapp-config.yaml")
@EnableWebSecurity
class Configuration {
    @Bean
    fun passwordEncoder(): PasswordEncoder =
        BCryptPasswordEncoder(4)

    @Bean
    fun corsConfigurationSource(): CorsConfigurationSource {
        val configuration = CorsConfiguration()
        configuration.allowedOrigins = listOf("http://localhost:3000")
        configuration.allowedMethods = listOf("GET", "POST", "DELETE")
        val source = UrlBasedCorsConfigurationSource()
        source.registerCorsConfiguration("/**", configuration)
        return source
    }

    @Bean
    fun userDetailsManager(dataSource: DataSource, passwordEncoder: PasswordEncoder):
UserDetailsManager {
        val manager = InMemoryUserDetailsManager()
        val user = User
            .withUsername("user")
            .password("password")
            .roles(USER_ROLE)
            .passwordEncoder(passwordEncoder::encode)
            .build()
        manager.createUser(user);
        return manager
    }

    @Bean
    fun filterChain(http: HttpSecurity): SecurityFilterChain {
        http {
            httpBasic { }
            csrf { disable() }
            cors { }
            headers { frameOptions { sameOrigin } }
            sessionManagement { sessionCreationPolicy=SessionCreationPolicy.STATELESS }
            authorizeHttpRequests {
                authorize("/", permitAll)
                authorize("/static/**", permitAll)
                authorize("/api/auth/signup", permitAll)
                authorize("/api/auth/validate", hasRole(USER_ROLE))
                authorize("/api/feed", permitAll)
                authorize("/api/user/**", permitAll)
                authorize("/api/post/get/**", permitAll)
                authorize("/api/post/create", hasRole(USER_ROLE))
            }
        }
    }
}

```

```

        authorize("/api/post/delete/*", hasRole(USER_ROLE))
        authorize("/api/post/like/*", hasRole(USER_ROLE))
        authorize("/api/post/ping", hasRole(USER_ROLE))
    }
}
return http.build()
}
}

```

UserService.kt

```

package ru.vzalygin.postapp.service

import org.springframework.data.repository.findByIdOrNull
import org.springframework.stereotype.Service
import ru.vzalygin.postapp.entities.UserDAO
import ru.vzalygin.postapp.repository.UserRepository

@Service
class UserService(
    val userRepository: UserRepository
) {
    fun getUserByLoginOrNull(login: String): UserDAO? =
        userRepository.findByIdOrNull(login)
}

```

PostService.kt

```

package ru.vzalygin.postapp.service

import org.springframework.data.repository.findByIdOrNull
import org.springframework.stereotype.Service
import ru.vzalygin.postapp.data.post.CreatePostIntent
import ru.vzalygin.postapp.data.post.Post
import ru.vzalygin.postapp.entities.LikeDAO
import ru.vzalygin.postapp.entities.PostDAO
import ru.vzalygin.postapp.entities.UserDAO
import ru.vzalygin.postapp.repository.LikeRepository
import ru.vzalygin.postapp.repository.PostRepository
import java.util.*

@Service
class PostService(
    val postRepository: PostRepository,
    val likeRepository: LikeRepository
) {
    fun getFeed(): List<Post> {
        return postRepository.findAll().map { fromDao(it) }.sortedBy { it.creationDate }.reversed()
    }

    fun getUsersPosts(user: UserDAO): List<Post> {
        return postRepository.findAllByAuthor(user).map { fromDao(it) }
    }

    fun getPostDAOByIdOrNull(id: UUID): PostDAO? {
        return postRepository.findByIdOrNull(id)
    }

    fun getPostByIdOrNull(id: UUID): Post? {
        return getPostDAOByIdOrNull(id)?.let { fromDao(it) }
    }

    fun createPost(author: UserDAO, createPostIntent: CreatePostIntent): UUID {

```

```

        val id = UUID.randomUUID()
        val post = PostDAO(
            id,
            author,
            Date(),
            createPostIntent.title,
            createPostIntent.content,
            false,
            createPostIntent.answerTo?.let { UUID.fromString(it) }
        )
        postRepository.save(post)
        return id
    }

    fun deletePost(id: UUID) {
        val post = postRepository.findByIdOrNull(id)!!
        postRepository.deleteById(id)
        postRepository.save(
            PostDAO(
                id = post.id,
                author = post.author,
                creationDate = post.creationDate,
                title = "",
                content = "",
                isDeleted = true,
                answerTo = post.answerTo
            )
        )
    }

    fun isPostHasLike(user: UserDAO, post: PostDAO): Boolean =
        likeRepository.findByUserAndPost(user, post) != null

    fun likePost(user: UserDAO, post: PostDAO) {
        val like = likeRepository.findByUserAndPost(user, post)

        if (like != null) {
            likeRepository.deleteById(like.id!!)
        } else {
            likeRepository.save(
                LikeDAO(
                    post, user
                )
            )
        }
    }

    private fun fromDao(dao: PostDAO): Post =
        Post(
            dao.id,
            dao.author,
            dao.creationDate,
            dao.title,
            dao.content,
            dao.answerTo,
            postRepository.findAllByAnswerTo(dao.id).map { child -> child.id!! },
            dao.isDeleted,
        )
}

```

UserRepository.kt

```
package ru.vzalygin.postapp.repository
```



```
import org.springframework.data.repository.CrudRepository
import org.springframework.stereotype.Repository
import ru.vzalygin.postapp.entities.UserDAO
```

```
@Repository
interface UserRepository : CrudRepository<UserDAO, String>
```

PostRepository.kt

```
package ru.vzalygin.postapp.repository
```

```
import org.springframework.data.repository.CrudRepository
import org.springframework.stereotype.Repository
import ru.vzalygin.postapp.entities.PostDAO
import ru.vzalygin.postapp.entities.UserDAO
import java.util.*
```

```
@Repository
interface PostRepository : CrudRepository<PostDAO, UUID> {
    fun findAllByAnswerTo(answerTo: UUID): List<PostDAO>

    fun findAllByAuthor(author: UserDAO): List<PostDAO>
}
```

UserDAO.kt

```
package ru.vzalygin.postapp.entities
```

```
import jakarta.persistence.Entity
import jakarta.persistence.Id
```

```
@Entity
data class UserDAO(
    val name: String,
    @Id
    val login: String,
) {
    constructor() : this("", "")
}
```

PostDAO.kt

```
package ru.vzalygin.postapp.entities
```

```
import jakarta.persistence.*
import java.util.*
```

```
//id, author, creationDate, title, content, answerTo=null, answeredFrom=[], liked=false,
isDeleted=false
```

```
@Entity
data class PostDAO(
    @Id
    val id: UUID,
    @ManyToOne(cascade = [CascadeType.PERSIST])
    @PrimaryKeyJoinColumn
    val author: UserDAO,
    @Column(nullable = false)
    val creationDate: Date,
    @Column(nullable = false)
    val title: String,
    @Column(nullable = false)
    val content: String,
    @Column(nullable = false)
    val isDeleted: Boolean = false,
```

```

        val answerTo: UUID? = null,
    ) {
        constructor() : this(UUID.randomUUID(), UserDao(), Date(0), "", "", false, null)
    }

```

UserCred.kt

```
package ru.vzalygin.postapp.data.user
```

```

data class UserCredentials(
    val name: String,
    val login: String,
    val password: String
)

```

Post.kt

```
package ru.vzalygin.postapp.data.post
```

```

import ru.vzalygin.postapp.entities.UserDao
import java.util.*

```

```

data class Post(
    val id: UUID,
    val author: UserDao,
    val creationDate: Date,
    val title: String,
    val content: String,
    val answerTo: UUID? = null,
    val answeredFrom: List<UUID> = listOf(),
    val isDeleted: Boolean,
)

```

CreatePostIntent.kt

```
package ru.vzalygin.postapp.data.post
```

```

data class CreatePostIntent(
    val title: String,
    val content: String,
    val answerTo: String? = null,
)

```

ProxyController.kt

```
package ru.vzalygin.postapp.controller
```

```

import org.springframework.http.HttpEntity
import org.springframework.http.HttpHeaders
import org.springframework.http.HttpMethod
import org.springframework.web.bind.annotation.*
import org.springframework.web.client.RestTemplate
import java.net.URI

```

```

@RestController
class ProxyController {
    @GetMapping("/")
    fun index(): String {
        return restTemplate.getForObject(URI.create(base), String::class.java)!!
    }

    @GetMapping("/static/{type}/{name}")
    fun static(@PathVariable type: String, @PathVariable name: String): String {
        val headers = HttpHeaders().apply {
            set("Content-type", when (type) {
                "css" -> "text/css"
            })
        }
    }
}

```

```

        "js" -> "text/javascript"
        else -> "text/plain"
    })
}
val entity = HttpEntity<String>(headers)
return restTemplate.exchange(URI.create("$base/static/$type/$name"),
HttpMethod.GET, entity, String::class.java).body!!
}

@RequestMapping(value = ["/static/js/{name}"], method = [RequestMethod.GET],
produces=["text/javascript; charset=utf-8"])
fun js(@PathVariable name: String): String {
    val headers = HttpHeaders().apply {
        set("Content-type", "text/javascript")
    }
    val entity = HttpEntity<String>(headers)
    return restTemplate.exchange(URI.create("$base/static/js/$name"), HttpMethod.GET,
entity, String::class.java).body!!
}

companion object {
    val base = "http://localhost:3000"
    val restTemplate = RestTemplate()
}
}

```

PostController.kt

```

package ru.vzalygin.postapp.controller

import org.springframework.security.core.Authentication
import org.springframework.web.bind.annotation.*
import ru.vzalygin.postapp.data.post.CreatePostIntent
import ru.vzalygin.postapp.data.post.Post
import ru.vzalygin.postapp.entities.PostDAO
import ru.vzalygin.postapp.service.PostService
import ru.vzalygin.postapp.service.UserService
import java.util.*

@RestController
@RequestMapping("/api/post")
class PostController(
    val userService: UserService,
    val postService: PostService
) {
    @GetMapping("/get/{id}")
    fun get(@PathVariable id: UUID): Post {
        return postService.getPostByIdOrNull(id)!!
    }

    @PostMapping("/create")
    fun create(authentication: Authentication, @RequestBody post: CreatePostIntent): UUID {
        // println(authentication.name)
        // println(userService.getUserByLoginOrNull(authentication.name))
        return postService.createPost(
            userService.getUserByLoginOrNull(
                authentication.name
            )!!,
            post
        )
    }

    @DeleteMapping("/delete/{id}")

```

```

fun delete(@PathVariable id: UUID) {
    postService.deletePost(id)
}

@PostMapping("/like/{id}")
fun likePost(authentication: Authentication, @PathVariable id: UUID) {
    postService.likePost(
        userService.getUserByLoginOrNull(
            authentication.name
        )!!,
        postService.getPostDAOByIdOrNull(id)!!
    )
}

@GetMapping("/like/{id}")
fun hasPostLike(authentication: Authentication, @PathVariable id: UUID): Boolean {
    val user = userService.getUserByLoginOrNull(authentication.name)
    val post = postService.getPostDAOByIdOrNull(id)
    return if (user != null && post != null) {
        postService.isPostHasLike(user, post)
    } else {
        false
    }
}

@GetMapping("/ping")
fun ping(authentication: Authentication): String {
    return authentication.name
}
}

```

CommonController.kt

```

package ru.vzalygin.postapp.controller

import org.springframework.data.crossstore.ChangeSetPersister.NotFoundException
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.PathVariable
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController
import ru.vzalygin.postapp.data.post.Post
import ru.vzalygin.postapp.entities.UserDAO
import ru.vzalygin.postapp.service.PostService
import ru.vzalygin.postapp.service.UserService

@RestController
@RequestMapping("/api")
class CommonController(
    val userService: UserService,
    val postService: PostService
) {
    @GetMapping("/feed")
    fun feed(): List<Post> {
        return postService.getFeed()
    }

    @GetMapping("/user/{username}")
    fun user(@PathVariable username: String): Pair<UserDAO, List<Post>> {
        val user = userService.getUserByLoginOrNull(username)
        if (user != null) {
            return Pair(user, postService.getUsersPosts(user))
        } else {

```

```

        throw NotFoundException()
    }
}
}

```

AuthController.kt

```

package ru.vzalygin.postapp.controller

import org.springframework.data.repository.findByIdOrNull
import org.springframework.security.core.Authentication
import org.springframework.security.crypto.password.PasswordEncoder
import org.springframework.security.provisioning.UserDetailsManager
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.PostMapping
import org.springframework.web.bind.annotation.RequestBody
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController
import ru.vzalygin.postapp.USER_ROLE
import ru.vzalygin.postapp.data.user.UserCredentials
import ru.vzalygin.postapp.entities.UserDAO
import ru.vzalygin.postapp.repository.UserRepository
import org.springframework.security.core.userdetails.User as UserAuth

@RestController
@RequestMapping("/api/auth")
class AuthController(
    val userDetailsManager: UserDetailsManager,
    val passwordEncoder: PasswordEncoder,
    val userRepository: UserRepository
) {
    @PostMapping("/signup")
    fun signup(@RequestBody userCredentials: UserCredentials) {
        userRepository.save(
            UserDAO(userCredentials.name, userCredentials.login)
        )
        userDetailsManager.createUser(
            UserAuth.withUsername(userCredentials.login)
                .roles(USER_ROLE)
                .password(userCredentials.password)
                .passwordEncoder(passwordEncoder::encode)
                .build()
        )
        // println(userRepository.findAll())
        // println(userRepository.findByIdOrNull(userCredentials.login))
    }

    @GetMapping("/validate")
    fun validate(authentication: Authentication): String? {
        return userRepository.findByIdOrNull(authentication.name)?.name
    }
}

```

postapp-config.yaml

```

#spring.datasource.url=jdbc:postgresql://${POSTGRES_HOST}/postapp
#spring.datasource.username=${POSTGRES_USER}
#spring.datasource.password=${POSTGRES_PASSWORD}
spring.datasource.url=jdbc:h2:mem:postapp
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password

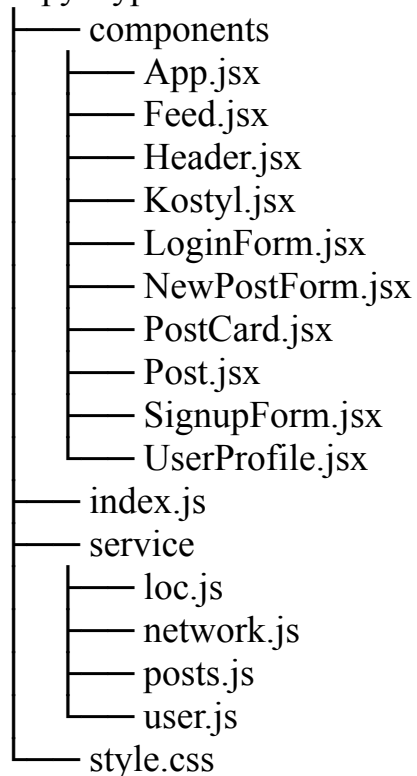
server.port=8081

```

Frontend

Клиентская часть выполнения с использованием фреймворка React и библиотеки react-router.

Структура:



App.jsx

```
import Feed from "../Feed";
import Post from "../Post";
import {
  createHashRouter,
  Navigate,
  RouterProvider,
} from "react-router-dom";
import * as React from 'react';
import UserProfile from "../UserProfile";
import NewPostForm from "../NewPostForm";
import LoginForm from "../LoginForm";
import SignupForm from "../SignupForm";
import { action as newPostAction } from "../NewPostForm";
import { action as loginAction } from "../LoginForm";
import { action as signupAction } from "../SignupForm";
import { AuthContext, makeUser, setState } from "../service/user";
import { useState } from "react";
import { LocaleContext } from "../service/loc";
import Kostyl from '../Kostyl';
```

```

const router = createHashRouter([
  {
    path: "/feed",
    element: <Feed />,
  },
  {
    path: "/post/:id",
    element: <Post />
  },
  {
    path: "/post/new",
    action: newPostAction,
    element: <NewPostForm />
  },
  {
    path: "/login",
    action: loginAction,
    element: <LoginForm />
  },
  {
    path: "/signup",
    action: signupAction,
    element: <SignupForm />
  },
  {
    path: "/:login",
    element: <UserProfile/>,
  },
  {
    path: "/",
    element: <Navigate to="/feed" replace={true} />
  },
  {
    path: "/postred",
    element: <Kostyl />
  }
]);

```

```

const App = () => {
  const [user, setUser] = useState(null);
  const [locale, setLocale] = useState("ru")
  // console.log("cookie")
  // console.log(document.cookie);
  if (!(document.cookie === "" || document.cookie === "token=") && user === null) {

```

```

    const [token, login, name] = document.cookie.split("=")[1].split(':')
    setUser(makeUser(login, name, token))
  }
  return (
    <React.Fragment>
      <LocaleContext.Provider value={{locale, setLocale}}>
        <AuthContext.Provider value={{user, setUser}}>
          <RouterProvider router={router} />
        </AuthContext.Provider>
      </LocaleContext.Provider>
    </React.Fragment>
  );
}

export default App;

```

Feed.jsx

```

import React, { Fragment, useContext, useEffect, useState } from 'react';
import PostCard from "../PostCard";
import { getPostsWithLikes } from "../service/posts";
import Header from '../Header';
import { AuthContext } from '../service/user';

const Feed = () => {
  const { user, _ } = useContext(AuthContext)
  const [ {posts, status}, setPosts ] = useState({posts: [], status: "pending"})

  useEffect(() => {
    getPostsWithLikes(user, (data) => {
      setPosts({posts: data, status: "ready"})
    })
  }, []);

  let postsBlock = null;
  if (status === "ready") {
    postsBlock = <Fragment>{
      posts.map(post => {
        return <PostCard key={post.id}
          id={post.id}
          author={post.author}
          creationDate={post.creationDate}
          title={post.title}
          content={post.content}
          answerTo={post.answerTo}
          answeredFrom={post.answeredFrom}
          liked={post.liked}

```



```

        isDeleted={post.isDeleted}
      />}})
    </Fragment>;
  } else {
    postsBlock = <h4>Loading... Please wait</h4>;
  }

  return (
    <React.Fragment>
      <Header/>
      <main className="container">
        {postsBlock}
      </main>
    </React.Fragment>
  );
};

export default Feed;

```

Header.jsx

```

import React, { useContext } from 'react';
import {
  AuthContext,
  isAuthorized,
} from "../service/user";
import { Link } from 'react-router-dom';
import { LOCALE_BUTTON, LOGIN_PAGE, LOGOUT_PAGE, LocaleContext, WRITE_POST, i18n } from
'../service/loc';

const Header = () => {
  const authContext = useContext(AuthContext);
  const { user, setUser } = authContext;

  const locContext = useContext(LocaleContext);
  const { locale, setLocale } = locContext;

  const logout = () => {
    console.log("jopa");
    document.cookie = "token=";
    setUser(null);
  }

  const postButton = () => {
    if (isAuthorized(authContext)) {
      return <Link to={"/post/new"} className="btn btn-primary my-2
my-lg-0">{i18n(locContext, WRITE_POST)}</Link>

```

```

    } else {
        return <Link to={"/login"} className="btn btn-primary my-2 my-lg-0">{i18n(locContext,
WRITE_POST)}</Link>
    }
  }) ()

const profileDropdown = (() => {
  if (isAuthorized(authContext)) {
    return (
      <React.Fragment>
        <Link to={`/${user.login}`} className="btn btn-link">{user.name}</Link>
        <button className="btn btn-outline-danger" onClick={logout}>{i18n(locContext,
LOGOUT_PAGE)}</button>
      </React.Fragment>
    )
  } else {
    return <Link to={"/login"} className="btn btn-primary" href="/login">{i18n(locContext,
LOGIN_PAGE)}</Link>
  }
}) ()

const switchLocale = () => {
  if (locale === "ru") { setLocale("eng") }
  if (locale === "eng") {setLocale("ru") }
};

const localButton =
  <button className="btn btn-outline-info" onClick={switchLocale}>{i18n(locContext,
LOCALE_BUTTON)}</button>

return (
  <nav className="navbar fixed-top navbar-self navbar-expand-lg navbar-dark bg-dark">
    <Link to={"/feed"} className="navbar-brand">Posting app</Link>
    <div className="collapse navbar-collapse" id="navbarSupportedContent">
      <ul className="navbar-nav mr-auto">
        {profileDropdown}
      </ul>
    </div>
    {localButton}
    {postButton}
  </nav>
);
};

export default Header;

```

LoginForm.jsx

```
import React, { useContext } from 'react';
import Header from './Header';
import { Form, Link, redirect } from 'react-router-dom';
import { getUserProfile, AuthContext, makeUser, validate } from "../../service/user"
import { with_base, with_auth } from '../../service/network';
import { ACCOUNT_NOT_FOUND, CREATE_ACCOUNT, LOGIN, LOGIN_BUTTON, LocaleContext, PASSWORD, i18n }
from '../../service/loc';

// Ensure that action func will be called only after NewPostForm component call
let kostyl = null;
let locKostyl = null;

export const action = async ({ request }) => {
  const formData = await request.formData()
  const login = formData.get("login");
  const password = formData.get("password");
  const profile = makeUser(login, null, password)

  // console.log(profile)
  var myHeaders = new Headers();
  const token = btoa(`${profile.login}:${profile.password}`)
  myHeaders.append("Authorization", "Basic " + token);
  console.log(myHeaders)
  const response = await fetch(with_base("/api/auth/validate"), {
    method: 'GET',
    headers:myHeaders,
  });
  console.log(response)
  if (response.ok) {
    const name = await response.text()
    kostyl(makeUser(login, name, password))
    document.cookie = `token=${password}:${login}:${name}`;
    return redirect("/feed");
  } else {
    console.log(await response.text());
    window.alert(i18n(locKostyl, ACCOUNT_NOT_FOUND));
    return null;
  }
};

const LoginForm = () => {
  const locContext = useContext(LocaleContext);
  const { user, setUser } = useContext(AuthContext);
  kostyl = setUser;
```

```

locKostyl = locContext

return (
  <React.Fragment>
    <Header/>
    <main className='container'>
      <Form className="card w-50 post-card" method='post'>
        <div className="form-col align-items-center form-group">
          <div className="col-auto">
            <label className="sr-only" htmlFor="inlineFormInputGroup">{i18n(locContext,
LOGIN)}</label>

            <div className="input-group mb-2">
              <div className="input-group-prepend">
                <span className="input-group-text" id="inputGroupPrepend">@</span>
              </div>
              <input name="login" type="text" className="form-control"
id="loginInput" placeholder="" required/>
            </div>
            <div className="col-auto">
              <label className="sr-only" htmlFor="inlineFormInput">{i18n(locContext,
PASSWORD)}</label>

              <input name="password" type="password" className="form-control mb-2"
id="passwordInput" placeholder="" required/>
            </div>
          </div>
          <div className="col-auto hor">
            <button type="submit" className="btn btn-primary mb-2">{i18n(locContext,
LOGIN_BUTTON)}</button>
          </div>
          <Link to={"/signup"}>{i18n(locContext, CREATE_ACCOUNT)}</Link>
        </div>
      </Form>
    </main>
  </React.Fragment>
);
};

export default LoginForm;

```

NewPostForm.jsx

```

import React, { useContext } from "react";
import { AuthContext, isAuthorized } from "../service/user";
import { Navigate, redirect, useSearchParams, useNavigate } from "react-router-dom";
import Header from "../Header";
import { Form } from 'react-router-dom';
import { createPost } from "../service/posts";

```

```

import { ANSWER_TO, CONTENT, CREATE_POST, LocaleContext, NEW_POST, TITLE, i18n } from
"../service/loc";

// Ensure that action func will be called only after NewPostForm component call
let kostyl = null;

export const action = async ({ request }) => {
  const formData = await request.formData()
  const title = formData.get("title");
  const content = formData.get("text");
  const answerTo = formData.get("answerTo");
  console.log(formData);
  createPost(kostyl, title, content, answerTo)
  return redirect("/feed");
};

const NewPostForm = () => {
  const [params, _] = useSearchParams();
  const authContext = useContext(AuthContext);
  const locContext = useContext(LocaleContext);
  const { user, setUser } = authContext;
  kostyl = user;

  if (isAuthorized(authContext)) {
    let answerTo = null;
    if (params.get("answerTo") !== null) {
      answerTo =
        <div className="form-group">
          <label>{i18n(locContext, ANSWER_TO)}</label>
          <input name="answerTo" type="text" className="form-control" id="answerToId"
placeholder="id" value={params.get("answerTo")} readonly="readonly"/>
        </div>
    }

    return (
      <React.Fragment>
        <Header/>
        <main className="container">
          <Form className="card w-50 post-card" method="post">
            {answerTo}
            <div className="form-group">
              <h4 htmlFor="postTitle">{i18n(locContext, NEW_POST)}</h4>
              <label className="sr-only"
htmlFor="inlineFormInputGroup">{i18n(locContext, TITLE)}</label>

```

```

        <input name="title" type="text" className="form-control" id="postTitle"
placeholder="" required/>
      </div>
      <div className="form-group">
        <label className="sr-only" htmlFor="inlineFormInputGroup">{i18n(locContext,
CONTENT)}</label>
        <textarea name="text" className="form-control" cols="40" rows="5"
id="postText" placeholder=""></textarea>
      </div>
      <button type="submit" className="btn btn-primary">{i18n(locContext,
CREATE_POST)}</button>
    </Form>
  </main>
</React.Fragment>
)
} else {
  return <Navigate to="/login" replace={true} />
}
};

export default NewPostForm;

```

Post.jsx

```

import React, { useState, useEffect, } from 'react';
import PostCard from './PostCard';
import { getPostById } from '../service/posts';
import { useParams } from 'react-router-dom';
import Header from './Header';

const Post = () => {
  const { id } = useParams();
  const [ post, setPost ] = useState(null)

  useEffect(() => {
    getPostById(id, (data) => {
      setPost(data)
    })
  }, []);

  let postBlock = null
  if (post === null) {
    postBlock = <h4>Загрузка</h4>
  } else {
    postBlock = <PostCard
      id={post.id}
      author={post.author}

```

```

        creationDate={post.creationDate}
        title={post.title}
        content={post.content}
        answerTo={post.answerTo}
        answeredFrom={post.answeredFrom}
        liked={post.liked}
        isDeleted={post.isDeleted}
      />
    }

    return (
      <React.Fragment>
        <Header/>
        <main className='container'>
          {postBlock}
        </main>
      </React.Fragment>
    )
  };

export default Post;

```

PostCard.jsx

```

import React, { useContext, useState } from 'react';
import { AuthContext, isAuthorized } from '../service/user';
import { Link } from 'react-router-dom';
import { setLikeOnPost, setDeletedOnPost } from '../service/posts';
import { DELETE_WARNING, LocaleContext, ME, POST_WAS_REMOVED, i18n } from '../service/loc';

const PostCard = ({id, author, creationDate, title, content, answerTo, answeredFrom, liked, isDeleted}) => {
  const authContext = useContext(AuthContext);
  const locContext = useContext(LocaleContext);
  const { user, _ } = authContext;

  const [state, setState] = useState({like: liked, isDeleted: isDeleted });
  const setLike = () => {
    const value = { like: !(state.like), isDeleted: state.isDeleted};
    setLikeOnPost(id, value.like);
    setState(value)
  };

  const setDeleted = () => {
    if (window.confirm(i18n(locContext, DELETE_WARNING))) {
      const value = { like: state.like, isDeleted: true };
      setDeletedOnPost(user, id);
    }
  };

```

```

        setState(value);
    }
}

const metaBlock = (
    <div className="container-fluid hor">
        <small className="font-weight-light row-links">{creationDate.substring(0,
10+9)}</small>
        <Link to={`/${postred?id=${id}`} `} className="btn btn-small font-weight-light
post-link">{id}</Link>
        <Link to={`/${author.login}`} `} className="btn btn-link btn-sm">
            @{author.name} {(() => {if(isAuthorized(authContext) && author.login ===
user.login){return i18n(locContext, ME);}else{return;}}) ()}
        </Link>
    </div>
);

const answerToBlock = (()=>{if(answerTo !== null)
    return <Link to={`/${postred?id=${answerTo}`} `} target="_blank" rel="noopener noreferrer"
className="btn btn-small font-weight-light post-link" >&lt;= {answerTo}</Link>;
}) ();

const answeredFromBlock =
    <div>
        {answeredFrom.map(answer => {
            return <Link to={`/${post}/${answer}`} `} target="_blank" rel="noopener noreferrer"
className="btn btn-sm font-weight-light post-link">=&gt; {answer}</Link>;
        })}
    </div>;

const likeButtonComponent = (()=>{
    if(state.like===true){
        return <button className="btn btn-liked" onClick={setLike} >♥</button>
    } else {
        return <button className="btn btn-light" onClick={setLike}>♥</button>;
    }
}) ();

const deleteButtonComponent = (()=>{if(isAuthorized(authContext) && author.login ===
user.login) {
    return <button type="button" className="btn btn-outline-danger"
onClick={setDeleted}>🗑️</button>;
}}) ();

const contentBlock = (()=>{if(state.isDeleted === false) {

```



```

        return <React.Fragment>
            <h5 className="card-title">{title}</h5>
            <p className="card-text">{content}</p>
            <div className="btn-group btn-group-sm" role="group" aria-label="Basic
example">
                { /* {likeButtonComponent} */ }
                <Link to={` /post/new?answerTo=${id}`} type="button" className="btn
btn-light"><img alt="add comment icon" data-bbox="161 213 178 226"/></Link>
                {deleteButtonComponent}
            </div>
        </React.Fragment>
    } else {
        return <h5 className="card-title">{i18n(locContext, POST_WAS_REMOVED)}</h5>
    }
  }
}

return (
  <div className="card w-50 post-card">
    <div className="card-body">
      {metaBlock}
      {answerToBlock}
      {answeredFromBlock}
      {contentBlock}
    </div>
  </div>
)
};

export default PostCard;

```

SignupForm.jsx

```

import React, { useContext } from 'react';
import Header from '../Header';
import { Form, Link, redirect } from 'react-router-dom';
import { getUserProfile, AuthContext, makeUser, signup } from "../service/user"
import { CONTRAINTMENTS, LOGIN, LOGIN_BUTTON, LOGIN_INTENT, LOGIN_PAGE, LocaleContext, NAME,
PASSWORD, i18n } from '../service/loc';

// Ensure that action func will be called only after NewPostForm component call
let kostyl = null;

export const action = async ({ request }) => {
  const formData = await request.formData()
  const name = formData.get("name");
  const login = formData.get("login");
  const password = formData.get("password");
  if (login.length < 3) {

```

```

        window.alert("Слишком короткий логин");
        return null;
    } else if (!login.match(/[a-z0-9]+)/)) {
        window.alert("Логин не удовлетворяет требованиям");
        return null;
    }
    if (password.length < 8) {
        window.alert("Слишком короткий пароль");
        return null;
    } else if (!password.match(/[a-z0-9]+)/)) {
        window.alert("Пароль не удовлетворяет требованиям");
        return null;
    }

    const profile = makeUser(login, name, password);
    kostyl(profile);
    signup(profile)
    document.cookie = `token=${password}:${login}:${name}`;

    return redirect("/feed");
};

const SignupForm = () => {
    const locContext = useContext(LocaleContext);
    const { user, setUser } = useContext(AuthContext);
    kostyl = setUser;

    return (
        <React.Fragment>
            <Header/>
            <main className='container'>
                <Form className="card w-50 post-card" method='post'>
                    <div className="form-col align-items-center form-group">
                        <div className="col-auto">
                            <label className="sr-only" htmlFor="inlineFormInput">{i18n(locContext,
NAME)}</label>

                            <input name="name" type="text" className="form-control mb-2"
id="nameInput" placeholder="" required/>
                        </div>
                        <div className="col-auto">
                            <label className="sr-only"
htmlFor="inlineFormInputGroup">{i18n(locContext, LOGIN)}</label>
                            <div className="input-group mb-2">
                                <div className="input-group-prepend">

```

```

        <span className="input-group-text"
id="inputGroupPrepend">@</span>
        </div>
        <input name="login" type="text" className="form-control"
id="loginInput" placeholder="" required/>
        </div>
        <small id="loginHelpBlock" className="form-text text-muted">
            {i18n(locContext, CONTRAINMENTS)}
        </small>
    </div>
    <div className="col-auto">
        <label className="sr-only" htmlFor="inlineFormInput">{i18n(locContext,
PASSWORD)}</label>
        <input name="password" type="password" className="form-control mb-2"
id="passwordInput" placeholder="" required/>
        <small id="passwordHelpBlock" className="form-text text-muted">
            {i18n(locContext, CONTRAINMENTS)}
        </small>
    </div>
    <div className="col-auto hor">
        <button type="submit" className="btn btn-primary
mb-2">{i18n(locContext, LOGIN_BUTTON)}</button>
    </div>
    <Link to={"/login"}>{i18n(locContext, LOGIN_INTENT)}</Link>
</div>
</Form>
</main>
</React.Fragment>
);
};

export default SignupForm;

```

UserProfile.jsx

```

import React, { Fragment, useContext, useState, useEffect } from 'react';
import { Navigate, useParams } from 'react-router-dom';
import Header from './Header';
import { AuthContext, getUserProfile, isAuthorized } from '../service/user';
import PostCard from './PostCard';

const UserProfile = () => {
    const authContext = useContext(AuthContext);
    const { user, _ } = authContext;

    const { login } = useParams();
    const [ profile, setProfile ] = useState(null);

```

```

useEffect(() => {
    getUserProfile(login, (data) => {
        setProfile(data);
    })
}, []);

const meText = (()=>{
    if(isAuthorized(authContext) && login === user.login) return "(me)";
})();

let profileBlock = null;
if (profile === null) {
    profileBlock = <h4>Зарпyska</h4>
} else {
    const userProfile = profile.first
    const userPosts = profile.second
    profileBlock = <Fragment>
        <div className="card w-50 post-card">
            <div className="card-body">
                <div className="hor">
                    <h5 className="card-title">{userProfile.name} {meText}&emsp;&emsp;</h5>
                    <span className='font-weight-light'>@{userProfile.login}</span>
                </div>
            </div>
        </div>
        {userPosts.map(post => <PostCard key={post.id}
            id={post.id}
            author={post.author}
            creationDate={post.creationDate}
            title={post.title}
            content={post.content}
            answerTo={post.answerTo}
            answeredFrom={post.answeredFrom}
            liked={post.liked}
            isDeleted={post.isDeleted}
        /> ) }
    </Fragment>
}

return (
    <Fragment>
        <Header/>
        <main className="container">
            {profileBlock}
        </main>
    </Fragment>
)

```

```

    </Fragment>

    );
}

export default UserProfile;

```

loc.js

```

import { createContext } from "react"

export const LocaleContext = createContext({loc: "ru"});

export const LOCALE_BUTTON = "locale_button";
export const ME = "me";
export const LOGIN_PAGE = "login_page"
export const LOGOUT_PAGE = "logout_page"
export const WRITE_POST = "write_post"
export const LOGIN = "login"
export const PASSWORD = "password"
export const NAME = "name"
export const LOGIN_BUTTON = "login_button"
export const CREATE_ACCOUNT = "create_account"
export const LOGIN_INTENT = "login_intent"
export const DELETE_WARNING = "delete_warning"
export const ACCOUNT_NOT_FOUND = "account_not_found"
export const POST_WAS_REMOVED = "post_was_removed"
export const ANSWER_TO = "answer_to"
export const NEW_POST = "new_post"
export const CREATE_POST = "create_post"
export const TITLE = "title"
export const CONTENT = "content"
export const CONTRAINMENTS = "conts"

const texts = {
  "locale_button": {
    "ru": "Переключить на английский",
    "eng": "Switch to russian"
  },
  "me": {
    "ru": " (Я) ",
    "eng": " (me) "
  },
  "login_page": {
    "ru": "Авторизоваться",
    "eng": "Authorize"
  },
  "logout_page": {

```

```
    "ru": "Выйти",
    "eng": "Logout"
  },
  "write_post": {
    "ru": "Написать пост",
    "eng": "Write a post"
  },
  "login": {
    "ru": "Логин",
    "eng": "Login"
  },
  "name": {
    "ru": "Имя",
    "eng": "Name"
  },
  "password": {
    "ru": "Пароль",
    "eng": "Password"
  },
  "login_button": {
    "ru": "Войти",
    "eng": "Enter"
  },
  "create_account": {
    "ru": "Нет аккаунта? Создайте его!",
    "eng": "Navigate to new account"
  },
  "login_intent": {
    "ru": "Уже есть аккаунт? Тогда просто войдите! :)",
    "eng": "Navigate to login page"
  },
  "delete_warning": {
    "ru": "Вы уверены, что хотите удалить пост?",
    "eng": "Are you sure want to delete the post?"
  },
  "account_not_found": {
    "ru": "Аккаунт не найден",
    "eng": "Account not found"
  },
  "post_was_removed": {
    "ru": "Пост был удален.",
    "eng": "The post was removed"
  },
  "answer_to": {
    "ru": "Ответ на",
```

```

    "eng": "Answer to"
  },
  "new_post": {
    "ru": "Новый пост!",
    "eng": "New post!"
  },
  "create_post": {
    "ru": "Создать пост",
    "eng": "Submit"
  },
  "title": {
    "ru": "Название",
    "eng": "Title"
  },
  "content": {
    "ru": "Содержание",
    "eng": "Content"
  },
  "conts": {
    "ru": "Только латинские маленькие буквы и цифры (не менее 3 символов)",
    "eng": "Only latin small letters and numbers (at least 3 characters)"
  },
};

export const il8n = (context, phrase) => {
  console.log(phrase)
  const { locale, _ } = context
  return texts[phrase][locale]
};

```

network.js

```

import { encode } from "base-64"

export const url_base = "http://localhost:8081";

export const with_base = (url) => { return url_base+url }

export const with_auth = (user) => {
  console.log(user.login, user.password)
  return new Headers({
    "Authorization": "Basic " + encode(`${user.login}:${user.password}`)
  })
}

export const with_auth2 = (user, headers) => {
  console.log(user.login, user.password)

```

```

    return headers.append(
      "Authorization", "Basic " + encode(`${user.login}:${user.password}`)
    )
  }
}

```

posts.js

```

import { with_auth, with_auth2, with_base } from "../network";
import {
  user, makeUser, getUserProfile
} from "../user";

const makePost = (id, author, creationDate, title, content, answerTo=null, answeredFrom=[],
liked=false, isDeleted=false) => {
  return {
    id: id,
    author: author,
    creationDate: creationDate,
    title: title,
    content: content,
    answerTo: answerTo,
    answeredFrom: answeredFrom,
    liked: liked,
    isDeleted: isDeleted
  }
};

export const getPostsWithLikes = (user, setCallback) => {
  fetch(with_base("/api/feed"), {
    method: 'GET',
  })
  .then((response) => response.json())
  .then((data) => {
    console.log(data);
    setCallback(data);
  })
};

export const getPostById = (id, setCallback) => {
  fetch(with_base(`/api/post/get/${id}`), {
    method: 'GET'
  })
  .then((response) => response.json())
  .then((data) => {
    console.log(data);
    setCallback(data);
  })
};

```



```

    })
  }

export const setLikeOnPost = (user, id) => {

};

export const setDeletedOnPost = (user, id) => {
  var myHeaders = new Headers();
  const token = btoa(`${user.login}:${user.password}`)
  myHeaders.append("Authorization", "Basic " + token);

  var requestOptions = {
    method: 'DELETE',
    headers: myHeaders,
    redirect: 'follow'
  };

  fetch("http://localhost:8081/api/post/delete/"+id, requestOptions)
    .then(response => response.text())
    .then(result => console.log(result))
    .catch(error => console.log('error', error));
}

export const createPost = (author, title, content, answerTo) => {
  const postIntent = {title, content, answerTo};
  console.log(postIntent);

  var myHeaders = new Headers();
  myHeaders.append("Content-Type", "application/json");
  const token = btoa(`${author.login}:${author.password}`)
  myHeaders.append("Authorization", "Basic " + token);

  var raw = JSON.stringify({
    "title": title,
    "content": content,
    "answerTo": answerTo,
  });

  var requestOptions = {
    method: 'POST',
    headers: myHeaders,
    body: raw,
    redirect: 'follow'
  };
};

```

```
    fetch("http://localhost:8081/api/post/create", requestOptions)
      .then(response => response.text())
      .then(result => console.log(result))
      .catch(error => console.log('error', error));
  };
};
```

user.js

```
import { createContext } from "react";
import { with_auth, with_base } from "./network";
import { encode } from "base-64";

// Data source
export const makeUser = (login, name, token) => {
  return { login: login, name: name, password: token };
};

export const getUserProfile = (login, setCallback) => {
  fetch(with_base(`/api/user/${login}`), {
    method: 'GET'
  })
  .then((response) => response.json())
  .then((data) => {
    console.log(data);
    setCallback(data);
  });
};

// Auth
export const isAuthorized = (context) => {
  const { user, _ } = context;
  return user !== null;
};

export const signup = (user) => {
  var myHeaders = new Headers();
  myHeaders.append("Content-Type", "application/json");

  var raw = JSON.stringify({
    "name": user.name,
    "login": user.login,
    "password": user.password
  });

  var requestOptions = {
    method: 'POST',
    headers: myHeaders,
```

```

        body: raw,
        redirect: 'follow'
    };

    fetch("http://localhost:8081/api/auth/signup", requestOptions)
    .then(response => response.text())
    .then(result => console.log(result))
    .catch(error => console.log('error', error));

    // fetch(with_base("/api/auth/signup"), {
    //     method: 'POST',
    //     body: JSON.stringify(user)
    // })
    // ).then((response) => console.log(response));
}

export const logIn = (context, user) => {
    const { __, setUser } = context;
    setUser(user)
};

export const logOut = (context) => {
    const { __, setUser } = context;
    setUser(null);
};

export const AuthContext = createContext(null);

```

index.js

```

import React from 'react';
import ReactDOM from 'react-dom/client';
import './style.css';
import App from './components/App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);

```

style.css

```

.navbar-self {
    padding: 15px;
}

.hor {
    display: flex;
}

.row-links {

```

```
padding-top: 10px;
}

.post-link {
  font-size: small;
  height: min-content;
}

.btn-liked {
  background-color: #ea5959;
}

.container {
  margin-top: 100px;
  width: 70%;
  margin-left: 30%;
  display: flex;
  justify-content: center;
  flex-direction: column;
}

.post-card {
  margin: 30px;
}

.form-group {
  padding: 10px;
}

.btn {
  margin: 10px;
}
```