**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Web-технологии»**

**Тема: модуль администрирования приложения «Социальная сеть»**

| Студент гр. 1303 |  | Чубан Д.В. |
| --- | --- | --- |
| Преподаватель |  | Беляев С. А. |

Санкт-Петербург

2023

**Цель работы.**

Целью работы является изучение возможностей применения компилятора Babel, библиотеки jQuery, препроцессора LESS, препроцессора SASS/SCSS, инструмента выполнения повторяющихся задач GULP, освоение инструмента сборки Webpack, регистрация разработанных модулей, формирования навыков построения структурированных web-приложений, освоение особенностей стандартных библиотек.

Для достижения поставленной цели требуется решить следующие задачи:

– разработка интерфейса web-приложения с использованием Figma (https://www.figma.com/);

– создание web-сервера на основе express, настройка маршрутов, подготовка и обработка REST-запросов (серверная часть);

– создание шаблонов web-страниц с использованием pug или ejs, указа ние путей подключения js-файлов;

– разработка стилей web-приложения с использованием LESS или SASS/SCSS;

– разработка клиентских js-файлов с использованием библиотеки jQuery и с использованием новейших возможностей в соответствии последним стандартом ECMAScript;

– конфигурирование GULP для решения задач преобразования pug файлов в формат HTML, less-файлов и sass-файлов в css-файлы, обработка js файлов с использованием Babel.

**Выполнение работы.**

С помощью Figma был разработан макет приложения.

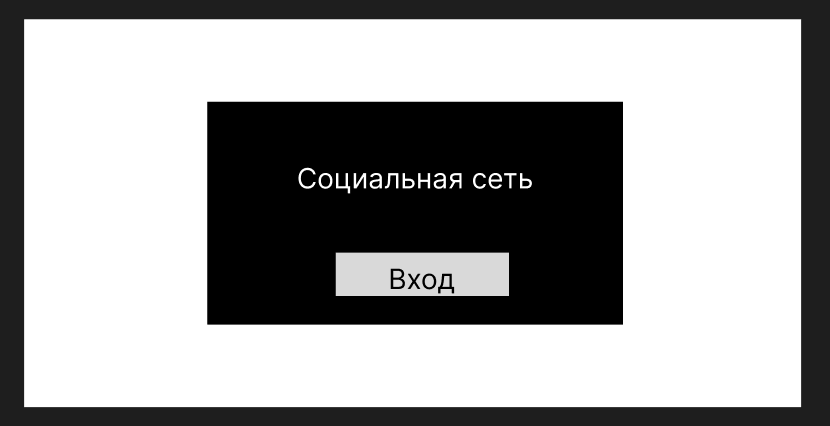


Рисунок 1 – макет страницы входа.



Рисунок 2 – макет главной страницы.



Рисунок 3 – макет списка друзей.

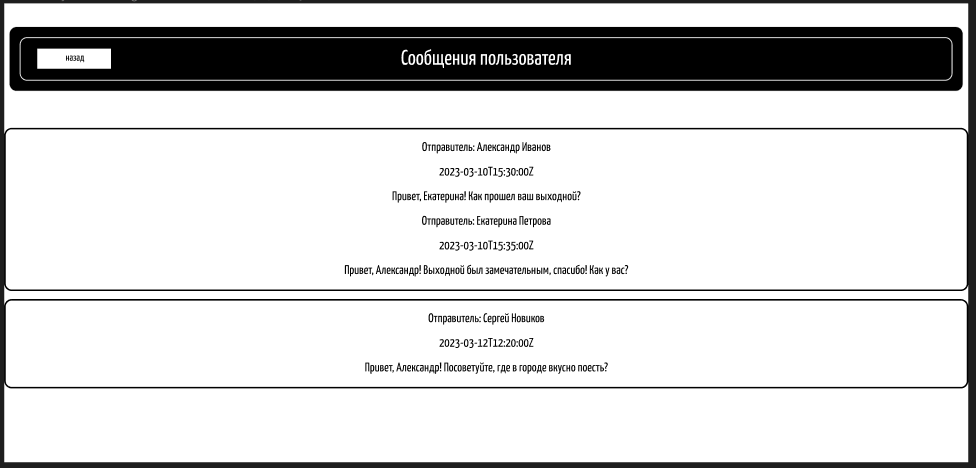


Рисунок 4 – макет сообщений пользователя.

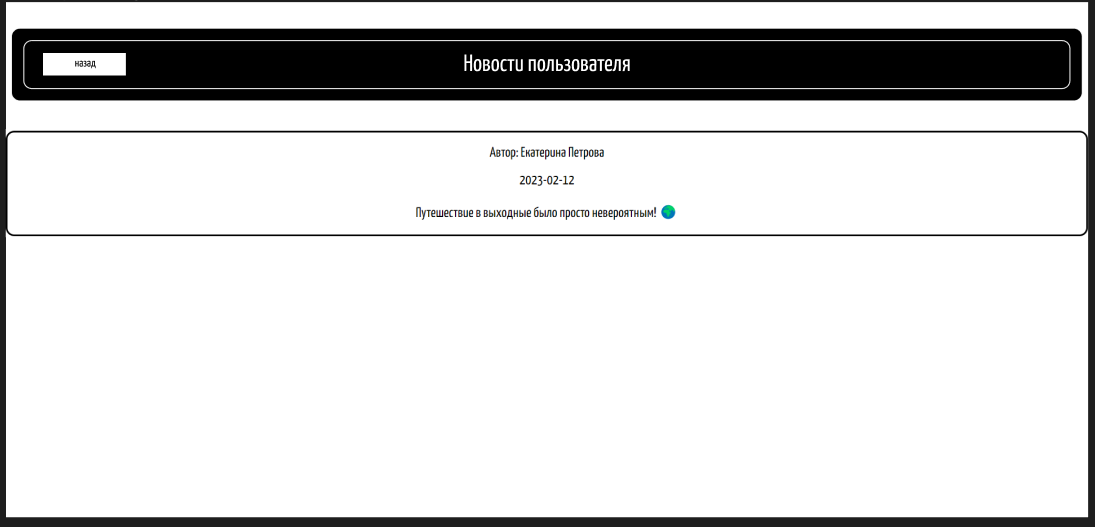


Рисунок 5 – макет новостей пользователя.

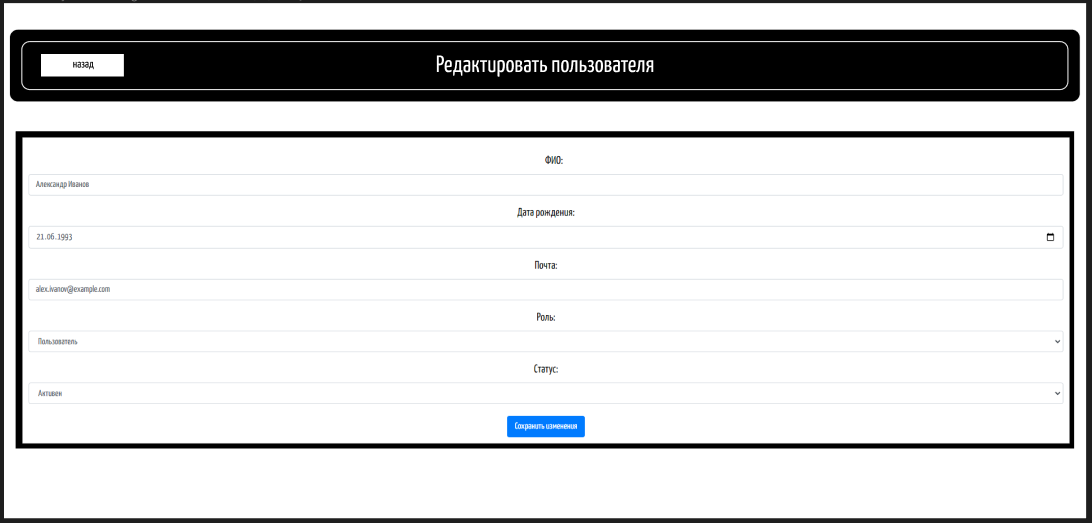


Рисунок 6 – макет страницы редактирования пользователя.

В файле *index.ts* описан сервер, на котором работает приложение.

Для навигации между страницами используется файл *router-pool.ts.* В нем находятся методы переходов между различными страницами. Для каждого раздела приложения создан отдельный router, который входит в router-pool.

Логика работы каждого раздела описана в соответствующем файле-контроллере, которые также объединяются в controller-pool.

Данные о пользователях, новостях и сообщениях хранятся в JSON-файлах. Для загрузки данных в приложение из JSON для взаимодействия с ними используются репозитории с данными, которые для удобства также объединены в один repository-pool.

Описание запросов к серверу находится в файле *frontend.ts.*

Внешний вид страниц описан с использованием pug-файлов и языка Less.

Сборка с помощью GULP описана в файле gulpfile.babel.js

Сборка с помощью WebPack описана в файле webpack.config.client.cjs



Рисунок 7 – итоговый вариант страницы входа.

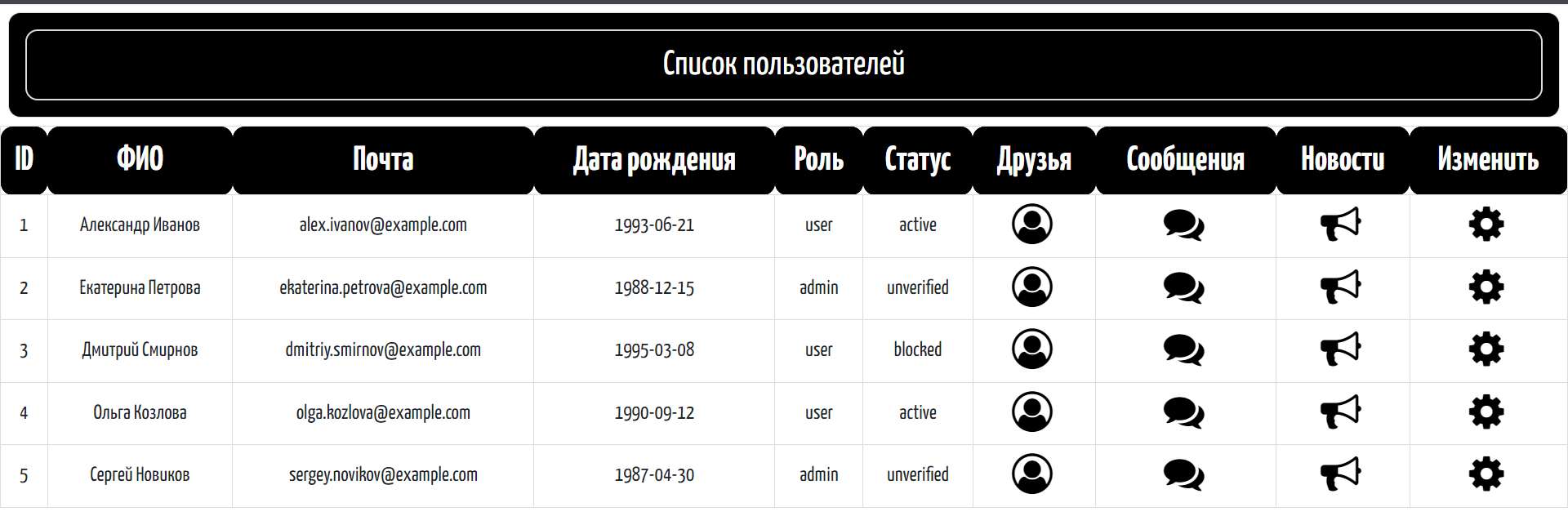


Рисунок 8 – итоговый вариант главной страницы.

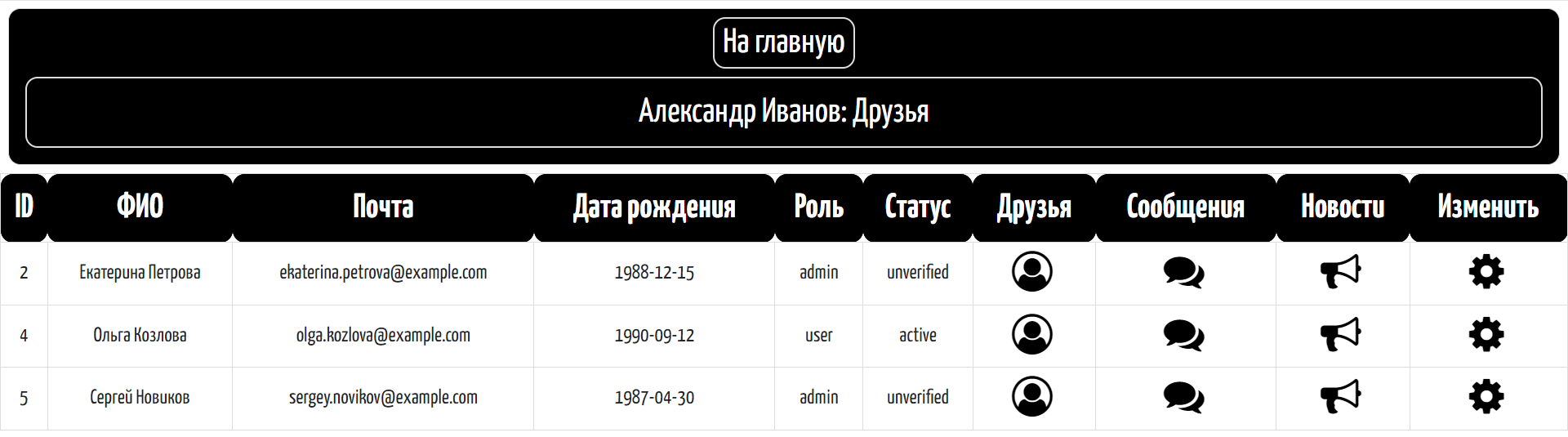


Рисунок 9 –итоговый вариант страницы друзей.

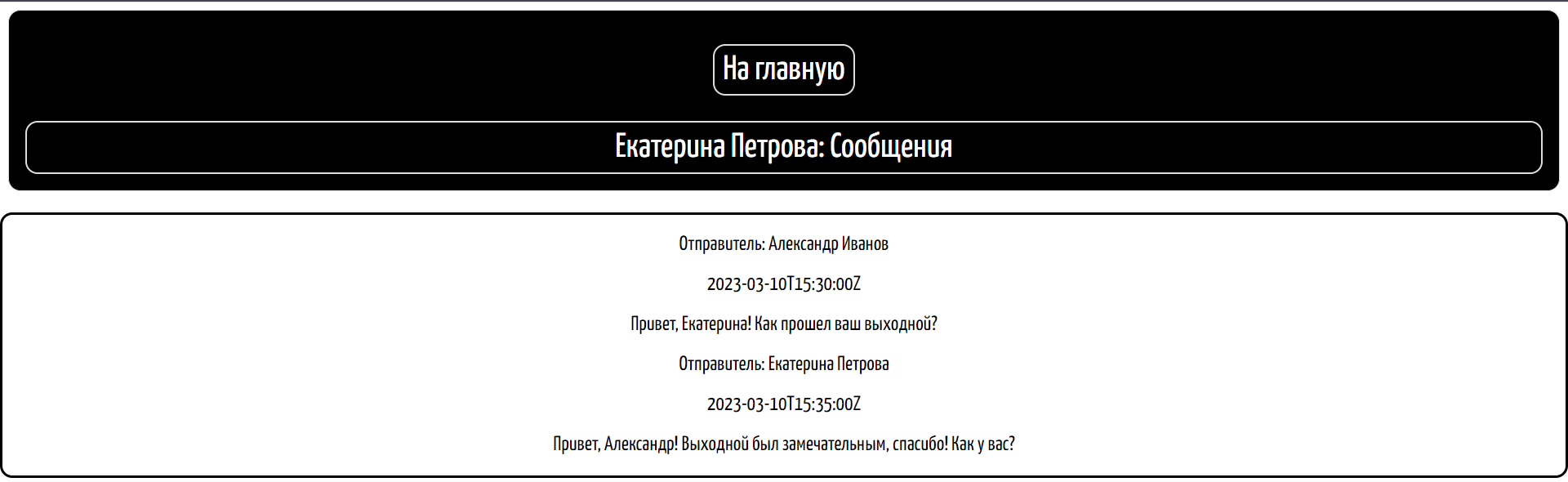


Рисунок 10 – итоговый вариант страницы сообщений.



Рисунок 11 – итоговый вариант страницы новостей пользователя.

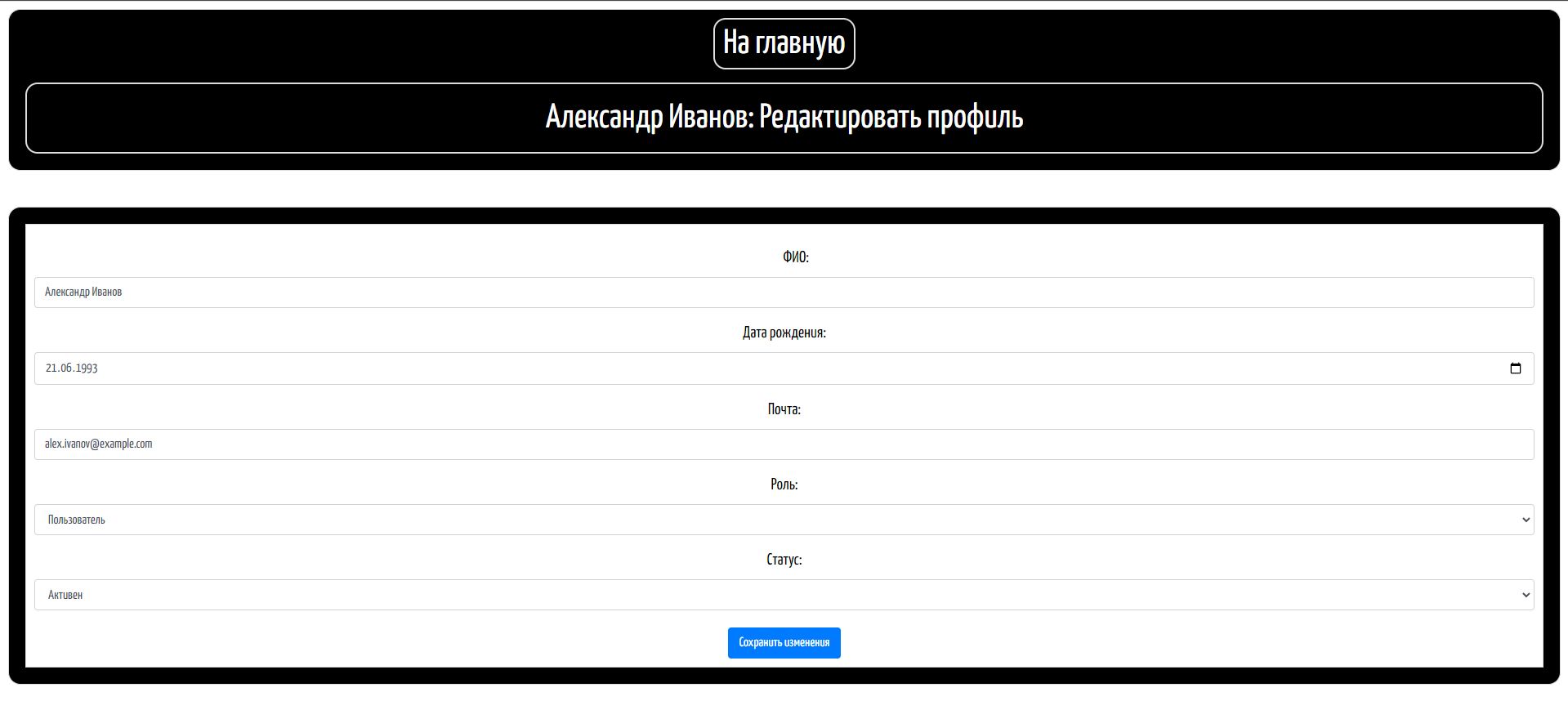


Рисунок 12 – итоговый вариант страницы редактирования пользователя.

**Приложение А  
Исходный код**

**index.ts:**

import fs from 'fs';

import path from 'path';

import http from 'http';

import https from 'https';

import express from 'express';

import compression from 'compression';

import cookieParser from 'cookie-parser';

import morgan from 'morgan';

import {\_\_front\_src\_dir, \_\_project\_dir, \_\_public\_dir} from "./config.js";

import backend from "./loaders/backend.js";

import frontend from "./loaders/frontend.js";

const server = express();

// pug

server.set('view engine', 'pug');

server.set('views', path.join(\_\_front\_src\_dir, 'views'));

// process public files

server.use(express.static(\_\_public\_dir));

// compress all responses with gzip

server.use(compression());

server.use(morgan('dev'));

server.use(express.json());

server.use(express.urlencoded({extended: false}));

server.use(cookieParser());

backend(server);

frontend(server);

const privateKey = fs.readFileSync(path.join(\_\_project\_dir, 'ssl\_keys/example.key'), 'utf-8');

const certificate = fs.readFileSync(path.join(\_\_project\_dir, 'ssl\_keys/example.crt'), 'utf8');

const credentials = {

key: privateKey,

cert: certificate,

};

const httpServer = http.createServer(server);

const httpsServer = https.createServer(credentials, server);

httpServer.listen(8080, () => {

console.log('Server started at http://localhost:8080/');

});

httpsServer.listen(8443, () => {

console.log('Server started at https://localhost:8443/');

});

**config.ts:**

import path from 'path';

import {fileURLToPath} from 'url';

// social-network root directory

export const \_\_project\_dir = path.join(path.dirname(fileURLToPath(import.meta.url)), '../../');

export const \_\_raw\_src\_dir = path.join(\_\_project\_dir, 'build');

export const \_\_front\_src\_dir = path.join(\_\_project\_dir, 'build/client');

export const \_\_public\_dir = path.join(\_\_project\_dir, 'build/client');

export const \_\_data\_dir = path.join(\_\_project\_dir, 'data');

export const \_\_url = 'http://localhost:8080';

**routers/user.ts:**

import e from "express";

import UserController from "../controllers/user.js";

export default class UserRouter{

constructor(private readonly controller: UserController) {}

public getRouter(){

const router = e.Router()

router.get('/', this.controller.getAll)

router.get('/:id(\\d+)', this.controller.getById)

router.post('/', this.controller.create)

router.put('/:id(\\d+)', this.controller.update)

router.delete('/:id(\\d+)', this.controller.delete)

return router

}

}

**routers/router-pool.ts:**

import UserRouter from "./user.js";

import NewsRouter from "./news.js";

import MessageRouter from "./message.js";

import controllerPool from "../controllers/controller-pool.js";

import FriendRouter from "./friend.js";

export default {

userRouter: new UserRouter(controllerPool.userController).getRouter(),

newsRouter: new NewsRouter(controllerPool.newsController, controllerPool.newsFeedController).getRouter(),

messageRouter: new MessageRouter(controllerPool.messageController).getRouter(),

friendRouter: new FriendRouter(controllerPool.userController).getRouter()

}

**routers/news.ts:**

import e from "express";

import NewsController from "../controllers/news.js";

import NewsFeedController from "../controllers/news-feed.js";

export default class NewsRouter{

constructor(private readonly controller: NewsController, private readonly feedController: NewsFeedController) {}

public getRouter(){

const router = e.Router()

// router.get('/:id(\\d+)', this.controller.getNewsByAuthorId);

router.get('/', this.controller.getAllNews)

router.get('/:id(\\d+)', this.feedController.getNewsFeed)

return router

}

}

**routers/message.ts:**

import e from "express";

import MessageController from "../controllers/message.js";

export default class MessageRouter{

constructor(private readonly controller: MessageController) {}

public getRouter(){

const router = e.Router()

router.get('/:id(\\d+)', this.controller.getChats);

router.get('/:id(\\d+)/:friend\_id(\\d+)', this.controller.getMessagesInChat)

return router

}

}

**routers/friend.ts:**

import e from "express";

import UserController from "../controllers/user.js";

export default class FriendRouter{

constructor(private readonly controller: UserController) {}

public getRouter(){

const router = e.Router()

router.get('/:id', this.controller.getFriendsByUserId);

return router

}

}

**repository/user.ts:**

import path from "path";

import {\_\_data\_dir} from "../config.js";

import {User} from "../models/user.js";

import {Error} from "../types/error.js"

import fs from "fs";

export default class UserRepository {

private static readonly SAVE\_FILENAME = path.join(\_\_data\_dir, 'user-repository.json');

public users: Map<number, User> = new Map();

private static id: number;

constructor() {

this.load();

}

public getById = async (id: number) => {

if (!this.users.has(id)) {

throw new Error(404, `User with id ${id} does not exist`);

}

return this.users.get(id)!;

}

public async getUsers(ids?: number[]): Promise<User[]> {

if (ids === undefined) {

return [...this.users.values()];

} else {

return ids

.filter((id) => this.users.has(Number(id)))

.map((id) => this.users.get(Number(id))!);

}

}

public async create(user: User): Promise<User>{

if (!user.isValid()){

throw new Error(400, 'Invalid Request Body')

}

user.id = UserRepository.id++

this.users?.set(user.id, user)

console.info(`[UserRepository] User created: ${user.name}`)

await this.save()

return user;

}

public async update(id: number, user: User): Promise<User>{

if(!user.isValid()){

throw new Error(400, `Bad Request`)

}

if(!this.users?.has(id)){

throw new Error(404, `User with ID ${id} is not found`)

}

const oldUser = this.users.get(id)!

user.id = user.id ?? oldUser.id;

user.name = user.name ?? oldUser.name;

user.email = user.email ?? oldUser.email;

user.date = user.date ?? oldUser.date;

user.status = user.status ?? oldUser.status;

user.role = user.role ?? oldUser.role;

user.friends = user.friends ?? oldUser.friends;

this.users?.set(id, user);

console.info(`[UserRepository] User edited: ${user.name}`)

await this.save()

return user

}

public async delete(id: number): Promise<Boolean>{

if(!this.users?.has(id)){

throw new Error(404, `User with ID ${id} is not found`)

}

this.users?.delete(id)

await this.save()

return true

}

public async getFriendsById(id: number): Promise<User[]>{

if(!this.users.has(id)){

throw new Error(404, `User with id ${id} does not exist`);

}

const friends\_ids = this.users.get(id)!.friends

if(!friends\_ids){

throw new Error(404, 'Friends not found')

}

let friends:User[] = []

for(const elem of friends\_ids){

if(this.users.has(elem)){

friends.push(<User>this.users.get(elem))

}

}

return friends

}

private load(){

if(!fs.existsSync(UserRepository.SAVE\_FILENAME)){

console.warn(`[UserRepository] Storage file ${UserRepository.SAVE\_FILENAME} is not found`)

return;

}

const buf = fs.readFileSync(UserRepository.SAVE\_FILENAME)

const users = JSON.parse(buf.toString());

UserRepository.id = users.length;

for (let userObj of users) {

const user = User.fromObject(userObj);

this.users.set(user.id!, user);

}

console.info(`[UserRepository] Data loaded`);

}

async save() {

const json = Array.from(this.users.values()); // Преобразуйте Map в массив объектов пользователей

const data = JSON.stringify(json, null, 2); // Преобразуйте массив в JSON-строку

if (!fs.existsSync(path.dirname(UserRepository.SAVE\_FILENAME))) {

fs.mkdirSync(path.dirname(UserRepository.SAVE\_FILENAME));

console.log(`[UserRepository] Created directory ${path.dirname(UserRepository.SAVE\_FILENAME)}`);

}

fs.writeFileSync(UserRepository.SAVE\_FILENAME, data);

console.log(`[UserRepository] Data saved to ${UserRepository.SAVE\_FILENAME}`);

}

}

**repository/repository-pool.ts:**

import UserRepository from "./user.js";

import NewsRepository from "./news.js";

import MessageRepository from "./message.js";

export default {

userRepo: new UserRepository(),

newsRepo: new NewsRepository(),

messageRepo: new MessageRepository(),

}

**repository/news.ts:**

import path from "path";

import fs from "fs";

import News from "../models/news.js";

import {\_\_data\_dir} from "../config.js";

import {Error} from "../types/error.js"

export default class NewsRepository {

private static readonly SAVE\_FILENAME = path.join(\_\_data\_dir, 'news-repository.json');

public news: Map<number, News[]> = new Map();

private static id: number;

constructor() {

this.load();

}

public async getAllNews(): Promise<News[]>{

return [...this.news.values()].flat()

}

public getNewsByAuthorId = async (id: number)=> {

if (!this.news.has(id)) {

throw new Error(404, `User with id ${id} does not exist`);

}

return [...this.news.get(id)!]

}

public getNews = async (ids: number[]) =>{

let news = []

for(const id of ids){

if(!this.news.has(id)){

continue

}

const buf = await this.getNewsByAuthorId(id)

news.push(...buf)

}

return news.sort((a:News, b:News) => new Date(a.date!).getTime() - new Date(b.date!).getTime())

}

private load(){

if(!fs.existsSync(NewsRepository.SAVE\_FILENAME)){

console.warn(`[NewsRepository] Storage file ${NewsRepository.SAVE\_FILENAME} is not found`)

return;

}

const buf = fs.readFileSync(NewsRepository.SAVE\_FILENAME)

const news = JSON.parse(buf.toString())

NewsRepository.id = news.length

for(let NewsObj of news){

const news = News.fromObject(NewsObj)

this.news.get(news.author\_id!) ?

this.news.get(<number>news.author\_id)?.push(news) :

this.news.set(<number>news.author\_id, [news])

}

console.info(`[NewsRepository] Data loaded`)

}

public async save(): Promise<void> {

const json = [];

for (const News of this.news.values()) {

json.push(JSON.stringify(News));

}

const data = JSON.stringify(json);

if (!fs.existsSync(path.dirname(NewsRepository.SAVE\_FILENAME))) {

fs.mkdirSync(path.dirname(NewsRepository.SAVE\_FILENAME));

console.log(`[NewsRepository] Created directory ${path.dirname(NewsRepository.SAVE\_FILENAME)}`);

}

fs.writeFile(NewsRepository.SAVE\_FILENAME, data, (err) => {

if (err) {

console.warn(`[NewsRepository] Failed to save News to ${NewsRepository.SAVE\_FILENAME}`);

throw err;

}

console.log(`[NewsRepository] Saved News to ${NewsRepository.SAVE\_FILENAME}`);

});

}

}

**repository/message.ts:**

import path from "path";

import fs from "fs";

import {Error} from "../types/error.js";

import Message from "../models/message.js";

import {\_\_data\_dir} from "../config.js";

export default class MessageRepository{

private static readonly SAVE\_FILENAME = path.join(\_\_data\_dir, 'message-repository.json');

protected messages = new Map<string, Message[]>()

private static id: number;

constructor() {

this.load()

}

public async getMessagesInChat(user\_id: number, friend\_id: number): Promise<void|Message[]>{

const key = [user\_id, friend\_id].sort().join('-')

if(!this.messages.has(key)){

throw new Error(404, `Chat between id ${user\_id} and ${friend\_id} does not exist`);

}

return this.messages.get(key)

}

public async getChats(user\_id: number): Promise<void|number[]>{

let chats: number[] = []

for(const elem of this.messages.keys()){

if(elem.includes(user\_id.toString())){

const users = elem.split('-').map(elem => parseInt(elem))

chats.push(...users.filter(elem => elem != user\_id))

}

}

return chats

}

public async getUserChats(user\_id: number) {

const chats = await this.getChats(user\_id)

if(!chats){

throw new Error(404, 'Chats not found')

}

let result = []

for(const elem of chats){

const buf = await this.getMessagesInChat(user\_id, elem)

if(buf){

result.push(buf)

}

}

return result

}

private load(){

if(!fs.existsSync(MessageRepository.SAVE\_FILENAME)){

console.warn(`[MessageRepository] Storage file ${MessageRepository.SAVE\_FILENAME} is not found`)

return;

}

const buf = fs.readFileSync(MessageRepository.SAVE\_FILENAME)

const messages = JSON.parse(buf.toString())

for(const messageObj of messages){

const message = Message.fromObject(messageObj)

const key = [message.receiver\_id, message.sender\_id].sort().join('-')

if(!this.messages.has(key)){

this.messages.set(key, [message])

}else{

this.messages.get(key)?.push(message)

}

}

console.info(`[MessageRepository] Data loaded`)

}

}

**models/user.ts:**

import validator from "validator"

export {User, STATUS, ROLE}

const STATUS = [

'active',

'inactive',

'unverified',

'blocked'

]

const ROLE = ['admin', 'user']

class User{

constructor(

public id?: number,

public name?: string,

public email?: string,

public date?: Date,

public status: string = "unverified",

public role?: string,

public friends?: number[]

) {}

static fromObject(obj: Object){

return Object.assign(new User(), obj)

}

public setStatus = (status: string) => {

this.status = status;

}

public setRole = (role: string) => {

this.role = role;

}

public isValid = () => {

return (validator.isEmail(this.email!)) &&

(validator.isInt(this.id!.toString())) &&

(this.name!.length > 0) &&

(validator.isIn(this.status, STATUS)) &&

(validator.isIn(this.role!, ROLE));

}

}

**models/news.ts:**

export default class News{

constructor(

public id: number = 1,

public author\_id?: number,

public content?: string,

public date?: Date

) {}

static fromObject(obj: Object){

return Object.assign(new News(), obj)

}

}

**models/message.ts:**

export default class Message{

constructor(

public sender\_id?: number,

public receiver\_id?: number,

public sender\_name?: string,

public text?: string,

public timestamp?: Date

) {}

static fromObject(obj: Object){

return Object.assign(new Message(), obj)

}

}

**loaders/frontend.ts:**

import express, {Express, Request, Response} from 'express';

import repositoryPool from "../repository/repository-pool.js";

export default (server: Express) => {

server.get('/', (req: Request, res: Response) => {

res.render('startPage');

});

server.get('/user', async (req: Request, res: Response) => {

try {

const response = await fetch('http://localhost:8080/api/user', { method: 'GET' });

if (response.ok) {

const userData = await response.json();

res.render('usersPage', { users: userData });

} else {

console.error('Произошла ошибка при получении данных:', response.status);

res.status(500).send('Произошла ошибка при получении данных');

}

} catch (error: any) {

console.error('Произошла ошибка:', error.message);

res.status(500).send('Произошла ошибка');

}

});

server.get('/friends/:id(\\d+)', async (req: Request, res: Response) => {

const currentUserId = parseInt(req.params["id"]);

try {

let renderData: any = {};

let response = await fetch(`http://localhost:8080/api/user/${currentUserId}`, { method: 'GET' });

if (response.ok) {

renderData.currentUser = await response.json();

} else {

console.error('Произошла ошибка при получении данных:', response.status);

res.status(500).send('Произошла ошибка при получении данных');

return;

}

response = await fetch(`http://localhost:8080/api/friend/${currentUserId}`, { method: 'GET' });

if (response.ok) {

renderData.users = await response.json();

} else {

console.error('Произошла ошибка при получении данных:', response.status);

res.status(500).send('Произошла ошибка при получении данных');

return;

}

res.render('usersFriends', { users: renderData.users, user: renderData.currentUser });

} catch (error: any) {

console.error('Произошла ошибка:', error.message);

res.status(500).send('Произошла ошибка');

}

});

server.get('/message/:id(\\d+)', async (req: Request, res: Response) => {

const currentUserId = parseInt(req.params["id"]);

try {

let responseData: any = {}

const response = await fetch(`http://localhost:8080/api/user/${currentUserId}`, {method: "GET"})

if(response.ok){

responseData.currentUser = await response.json()

}

res.render('usersChats', {usersChats: await repositoryPool.messageRepo.getUserChats(currentUserId), user: responseData.currentUser});

} catch (error: any) {

console.error('Произошла ошибка:', error.message);

res.status(500).send('Произошла ошибка');

}

});

server.get('/news/:id(\\d+)', async (req: Request, res: Response) => {

const currentUserId = parseInt(req.params["id"]);

try {

let responseData: any = {}

const response = await fetch(`http://localhost:8080/api/user/${currentUserId}`, {method: "GET"})

if(response.ok){

responseData.currentUser = await response.json()

}

res.render('usersNews', {usersNews: await repositoryPool.newsRepo.getNews(responseData.currentUser["friends"]), user: responseData.currentUser});

}catch (error: any) {

console.error('Произошла ошибка:', error.message);

res.status(500).send('Произошла ошибка');

}

});

server.get('/user/edit/:id(\\d+)', async (req: Request, res: Response) => {

const user\_id = parseInt(req.params['id'])

try {

let responseData: any = {}

const response = await fetch(`http://localhost:8080/api/user/${user\_id}`, {method: "GET"})

if(response.ok){

responseData.currentUser = await response.json()

}

res.render('editUser', {user: responseData.currentUser});

}catch (error: any) {

console.error('Произошла ошибка:', error.message);

res.status(500).send('Произошла ошибка');

}

})

}

**loaders/backend.ts:**

import e from "express";

import routersPool from "../routers/router-pool.js";

export default (server: e.Express) => {

server.use('/api/user', routersPool.userRouter);

server.use('/api/news', routersPool.newsRouter);

server.use('/api/message', routersPool.messageRouter);

server.use('/api/friend', routersPool.friendRouter);

server.use('/api', errorHandler);

};

function errorHandler(err: any, req: e.Request, res: e.Response, next: e.NextFunction) {

console.error(err.message)

let status = err.code || 500

res.status(status).send(status === 500 ? 'Internal server error' : err.message)

}

**controllers/user.ts:**

import UserRepository from "../repository/user.js";

import e from "express";

import {User} from "../models/user.js";

export default class UserController {

constructor(private userRepository: UserRepository) {}

public getById = async (req: e.Request, res: e.Response) => {

try {

const id = Number(req.params["id"]);

const user = await this.userRepository.getById(id);

res.status(200).send(JSON.stringify(user));

} catch (error: any) {

res.status(404).json({ error: error.message });

}

};

public getAll = async (req: e.Request, res: e.Response) => {

try {

const ids = req.query["ids"] ? (req.query["ids"] as string).split(',').map(Number) : undefined;

const users = await this.userRepository.getUsers(ids);

res.status(200).send(JSON.stringify(users));

} catch (error: any) {

res.status(404).json({ error: error.message });

}

};

public create = async (req: e.Request, res: e.Response) => {

try {

const user = User.fromObject(req.body); // Предполагается, что вы передаете данные пользователя в теле запроса

const createdUser = await this.userRepository.create(user);

res.status(201).send(JSON.stringify(createdUser));

} catch (error: any) {

res.status(400).json({ error: error.message });

}

};

public update = async (req: e.Request, res: e.Response) => {

try {

const id = Number(req.body["id"]);

const user = new User(id, req.body["name"], req.body['email'], req.body['date'], req.body['status'], req.body['role']);

const updatedUser = await this.userRepository.update(id, user);

res.status(200).send(JSON.stringify(updatedUser));

} catch (error: any) {

res.status(400).json({ error: error.message });

}

};

public delete = async (req: e.Request, res: e.Response) => {

try {

const id = Number(req.params["id"]);

const success = await this.userRepository.delete(id);

res.json({ success });

} catch (error: any) {

res.status(400).json({ error: error.message });

}

};

public getFriendsByUserId = async (req: e.Request, res: e.Response) => {

try {

const id = Number(req.params["id"]);

const friends = await this.userRepository.getFriendsById(id);

res.status(200).send(JSON.stringify(friends));

} catch (error:any) {

res.status(400).json({ error: error.message });

}

};

}

**controllers/news-feed.ts:**

import UserRepository from "../repository/user.js";

import {Error} from "../types/error.js";

import NewsRepository from "../repository/news.js";

import News from "../models/news.js";

export default class NewsFeedController {

constructor(private newsRepo: NewsRepository, private userRepo: UserRepository) {}

public getNewsFeed = async (id: number) => {

const user = await this.userRepo.getById(id)

console.log(user)

const friends = user.friends;

if (!friends) {

throw new Error(404, `User with id ${id} friend not found`);

}

let result: News[] = [];

for (const elem of friends) {

result.push(...[...await this.newsRepo.getNewsByAuthorId(elem)])

}

return result.sort((a: News, b: News) => {

const dateA = a.date ? new Date(a.date) : null;

const dateB = b.date ? new Date(b.date) : null;

if (dateA && dateB) {

return dateA.getTime() - dateB.getTime();

}

return 0;

});

}

}

**controllers/news.ts:**

import NewsRepository from "../repository/news.js";

import e from "express";

export default class NewsController {

constructor(private newsRepository: NewsRepository) {}

public getAllNews = async (req: e.Request, res: e.Response) => {

try {

const news = await this.newsRepository.getAllNews();

res.status(200).send(JSON.stringify(news));

} catch (error:any) {

res.status(404).json({ error: error.message });

}

};

public getNewsByAuthorId = async (req: e.Request, res: e.Response) => {

try {

const id = parseInt(req.params["id"]);

const news = await this.newsRepository.getNewsByAuthorId(id);

res.status(200).send(JSON.stringify(news));

} catch (error: any) {

res.status(404).json({ error: error.message });

}

};

}

**controllers/message.ts:**

import MessageRepository from "../repository/message.js";

import e from "express";

export default class MessageController {

constructor(private messageRepo: MessageRepository) {}

public getMessagesInChat = async (req: e.Request, res: e.Response) => {

try {

const user\_id: number = parseInt(req.params["id"]);

const friend\_id: number = parseInt(req.params["friend\_id"]);

const messages = await this.messageRepo.getMessagesInChat(user\_id, friend\_id);

if (!messages) {

res.status(404).json({ error: `Chat between id ${user\_id} and ${friend\_id} does not exist` });

} else {

res.status(200).json(messages);

}

} catch (error: any) {

res.status(404).json({ error: error.message });

}

}

public getChats = async (req: e.Request, res: e.Response) => {

try {

const user\_id: number = parseInt(req.params["id"]);

const chats = await this.messageRepo.getChats(user\_id);

res.status(200).json(chats);

} catch (error: any) {

res.status(500).json({ error: error.message });

}

}

}

**controllers/controller-pool.ts:**

import UserController from "./user.js";

import NewsController from "./news.js";

import MessageController from "./message.js";

import repositoryPool from "../repository/repository-pool.js";

import NewsFeedController from "./news-feed.js";

export default {

userController: new UserController(repositoryPool.userRepo),

newsController: new NewsController(repositoryPool.newsRepo),

messageController: new MessageController(repositoryPool.messageRepo),

newsFeedController: new NewsFeedController(repositoryPool.newsRepo, repositoryPool.userRepo)

}