**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**По лабораторной работе № 4**

**по дисциплине «Web-Технологии»**

Тема: **модуль пользователя приложения «социальная сеть»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0303 |  | Болкунов В.О. |
| Преподаватель |  | Беляев С. А. |

Санкт-Петербург

2022

**Цель работы.**

Целью работы является изучение основ языка TypeScript и особенностей применения фреймворка Angular для разработки web-приложений, ведения журналов ошибок, реализации взаимодействия приложений с использованием web-сокетов, организации модульного тестирования web-приложений с использованием Jest. Для достижения поставленной цели требуется решить следующие задачи:

– разработка интерфейса web-приложения;

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

– создание каркаса web-приложения с использованием Angular;

– определение перечня компонентов и сервисов web-приложения;

– создание шаблонов компонентов;

– регистрация и подключение в web-приложение журнала ошибок.

– создание web-сокета для отправки сообщений всем клиентам;

– создание и запуск Jest тестов для приложения.

**Общая формулировка задачи.**

Необходимо создать web-приложение, обеспечивающее использование пользователем социальной сети. Пользователь может зарегистрироваться в социальной сети. Может добавить или удалить свою фотографию, может управлять своими друзьями в социальной сети, может добавить сообщение (новость) на свою страницу, может просматривать список новостей своих друзей. Основные требования:

1. Приложение получает исходные данные из модуля администрирования приложения «Социальная сеть» в виде JSON-файла и работает одновременно с модулем администрирования приложения «Социальная сеть».

2. В качестве сервера используется Node.JS с модулем express.

3. Предусмотрены: – HTML-страница для регистрации пользователя; – HTML-страница для просмотра ленты новостей (пользователя и его друзей); – HTML-страница для добавления сообщения (новости).

4. Если пользователь является администратором, то у него есть возможность перехода в модуль администрирования приложения «Социальная сеть».

5. Переписка и страница новостей обновляются сразу после появления сообщений и новостей от пользователей без необходимости обновлять страницу целиком.

6. Разработаны тесты для серверной части web-приложения с использованием Jest.

7. Все элементы управления реализованы с использованием компонентов Angular. Взаимодействие между компонентами реализовано с использованием сервисов Angular.

8. Для реализации эффектов на HTML-страницах используются директивы Angular.

9. Для всех страниц web-приложения разработан макет интерфейса с использованием Figma (https://www.figma.com/). Преимуществом будет использование звукового сопровождения событий: получений сообщений, появление новостей. Преимуществом будет использование компонентов Angular Material (https://material.angular.io/).

**Выполнение работы.**

**Промежуточное api**

В модуле *api* описаны общие для клиентского и серверного приложений структуры данных и объекты.

* Роль пользователя

export enum Role {  
 *USER*,  
 *ADMIN*,  
}

* Статус пользователя/ресурса

export enum Status {  
 *ACTIVE*,  
 *UNCONFIRMED*,  
 *BLOCKED*,  
}

* Интерфейс индексируемых объектов

export interface Indexed {  
 id: number;  
}

* Интерфейс данных статуса

export interface StatusData {  
 status: Status;  
}

* Интерфейс статусных данных пользователя (его роль и статус)

export interface UserStatusData extends StatusData {  
 role: Role;  
}

* Интерфейс персональных данных

export interface PersonalData {  
 surname: string;  
 name: string;  
 lastName: string;  
 email: string;  
 birthDate: string;  
}

* Интерфейс данных пользователя (участвует в запросах)

export interface UserData extends Indexed, UserStatusData, PersonalData {  
 friends: number[];  
 photoId: number | null;  
}

* Интерфейс пользователя с паролем (хранится на сервере)

export interface User extends UserData {  
 password: string;  
}

* Интерфейс ресурса

export interface Resource extends Indexed, StatusData {  
 userId: number;  
 time: string;  
}

* Фотографии пользователя

export interface Photo extends Resource {  
 file: string;  
}

* Пост (новость) пользователя

export interface Post extends Resource {  
 text: string;  
 photosId: number[];  
}

* Авторизационные данные

export interface UserAuthData {  
 email: string;  
 password: string;  
}

* Регистрационные данные

export interface UserSignUpData extends PersonalData, UserAuthData {}

* Объект с конфигурацией клиент-серверного взаимодействия

export const config

* События отправляемые с сервера по WebSocket

export interface ServerToClientEvents {  
 post: (user: UserData) => void;  
}

* События отправляемые с клиента (отсутствуют)

export type ClientToServerEvents = EventsMap;

**Серверная часть (REST API)**

Объекты на сервере хранятся в виде коллекций, для удобной работы с которыми был создал класс ModelCollection

export class ModelCollection<T extends Indexed>   
 readonly list: T[] = [];  
 private id = 1;  
 add(obj: T): number   
 find(id: number): T | null   
 remove(id: number)   
 getIds(): number[]

Управление пользователями, их ресурсами и файлами на диске осуществляет класс NetworkModel.

export class NetworkModel {  
 private users: ModelCollection<User> = new ModelCollection<User>();  
 private photos: ModelCollection<Photo> = new ModelCollection<Photo>();  
 private posts: ModelCollection<Post> = new ModelCollection<Post>();  
 readonly storagePath: string;  
  
 constructor(users: User[], photos: Photo[], posts: Post[], storagePath: string)signUp(user: UserSignUpData): number | null authorize(data: UserAuthData): User | null updateUserStatus(id: number, stat: UserStatusData) updatePersonal(id: number, data: PersonalData) private getUser(id: number): User | null getUsers(): number[]getUserData(id: number): UserData | nullgetFriends(id: number): number[] | null   
 private getItems<T extends Resource>(userId: number, items: T[]): number[]getPosts(userId: number): number[] getPost(id: number): Post | null addPost(id: number, text: string, photosId: number[]): number getPhotos(userId: number): number[]getPhoto(id: number): Photo | null addPhoto(id: number, path: string): numbergetFriendsPosts(id: number): number[]   
 private updateResourceStatus<T extends Resource>(res: T | null, sd: StatusData )updatePostStatus(id: number, sd: StatusData)updatePhotoStatus(id: number, sd: StatusData)  
 deletePhoto(id: number)updateAvatar(id: number, path: string)friendRequest(id: number, friendId: number) friendDecline(id: number, friendId: number)

Подключениями пользователей и отправкой им сообщений через WebSocket управляет класс SocketManager

export class SocketManager {private pool: Map<number, io.Socket> = new Map();  
 constructor(  
 public readonly server: io.Server<  
 ServerToClientEvents,  
 ClientToServerEvents,  
 InterServerEvents,  
 SocketData  
 >,  
 secretKey: string  
 ) {  
 this.server  
 .use(*wrapSocketMW*(*cookieParser*(secretKey)))  
 .use(*wrapSocketMW*(passport.initialize()))  
.use((socket, next) =>  
 *wrapSocketMW*(…)(socket, next)  
 ).on("connection", (socket) => {…});  
 }post(user: UserData)

REST-api приложения реализовано с помощью роутера express в модуле *router.ts*

export function *createNetworkRouter*(  
 model: NetworkModel,  
 sm: SocketManager,  
 storagePath: string,  
 uploader: Multer  
): *Router* {  
 const router = *Router*();  
 const blockedImg = "blocked.jpg";router.use((req, res, next) => {…});router.get("/self", (req, res) => {…});router.get("/users", (req, res) => {…});router.use("/user/:id", (req, res, next) => {…});router.use(  
 "/user/:id",  
 *Router*()  
 .get("/photos", (req, res) => {…})  
 .get("/posts", (req, res) => {…})  
 .get("/friends", (req, res) => {…})  
 .get("/friendsposts", (req, res) => {…})  
 );router.put("/avatar", uploader.single("avatar"), (req, res) => {…});router.delete("/avatar", (req, res) => {…});  
 router  
 .route("/friend/:id")  
 .all((req, res, next) => {…}).post((req, res) => {…}).delete((req, res) => {…});router.post("/post", uploader.array("photos"), (req, res) => {…});

router  
 .route("/post/:postid")  
 .all((req, res, next) => {…})  
 .get((req, res) => {…}).patch((req, res) => {…});router  
 .route("/photo/:photoid")  
 .all((req, res, next) => {…})  
 .get((req, res) => {…});router  
 .route("/photoinfo/:photoid")  
 .all((req, res, next) => {…})  
 .get((req, res) => {…}).patch((req, res) => {…});router  
 .route("/user/:id")  
 .get((req, res) => {…})  
 .put((req, res) => {…})  
 .patch((req, res) => {…});  
 return router;  
}

Аутентификация пользователя осуществляется с помощью библиотеки passport.js с использованием стратегии Web-токенов. При отправке post-запроса с корректными данными **незаблокированного** пользователя**,** клиенту отправляется cookie с подписанным токеном.

Модуль *auth.ts* предоставляет middlware для аутентификации. Для входа требуется email и пароль пользователя. Также модуль осуществляет регистрацию пользователей и предоставляет точку проверки действительности токена.

С помощью callback-функции аутентификации можно получить объект аутентифицированного токеном пользователя для дальнейшего использования (например проверки привилегий администратора).

export const secretKey = "#S3cR3t\_K3y#";const cookieExtractor: JwtFromRequestFunction = function (req): string {…};  
export { passport };  
export function *createAuthRouter*(model: NetworkModel): *Router* {  
 const authRouter = *Router*();passport.use(  
 new JwtStrategy(  
 {  
 jwtFromRequest: cookieExtractor,  
 secretOrKey: secretKey,  
 },  
 (payload: Indexed, done) => {…}  
 )  
 );authRouter.post("/signup", (req, res) => {…});authRouter.post("/login", (req, res) => {…});authRouter.get(…);  
 return authRouter;  
}

**Клиентская часть**

Был создан макет основных страниц приложения, который представлен в файле *Lab4.fig****.***

Загрузку ресурсов осуществляют промежуточные классы в модуле *loaders*

* Интерфейс загрузчика

export interface ILoader   
 get url(): URL;  
 fetch(): void;

* Интерфейс загрузчика индексируемых объектов

export interface IndexedLoader   
 readonly id: number;  
 get endpoint(): string;

* Абстрактный класс загрузчика списков ресуров

export abstract class ListLoader implements ILoader   
 readonly list: number[] = [];  
 abstract get url(): URL;  
 async fetch()

* Загрузчик списка пользователей

export class UsersListLoader extends ListLoader   
 get url(): URL

* Абстрактный загрузчик списков данных ресурсов пользователя

abstract class UserDataListLoader extends ListLoader implements IndexedLoader   
 readonly id: number;  
 constructor(id: number)   
 abstract get endpoint(): string;  
 get url(): URL

* Загрузчик списка постов пользователя

export class PostsLoader extends UserDataListLoader   
 get endpoint(): string

* Загрузчик списка фотографий пользователя

export class PhotosLoader extends UserDataListLoader   
 get endpoint(): string

* Загрузчик списка друзей пользователя

export class FriendsLoader extends UserDataListLoader   
 get endpoint(): string

* Загрузчик списка постов друзей

export class FriendsPostsLoader extends PostsLoader   
 get endpoint(): string

* Абстрактный загрузчик объектов

export abstract class ObjectLoader<T> implements ILoader, IndexedLoader   
 readonly id: number;  
 data: T | null = null;  
 constructor(id: number)   
 abstract get endpoint(): string;  
 get url(): URL   
 async fetch()

* Загрузчик данных пользователя

export class UserLoader extends ObjectLoader<UserData>   
 get endpoint(): string   
 get avatarUrl(): string   
 get fullName()  
 updateStatus()  
 updatePersonal()

* Абстрактный загрузчик ресурсов

abstract class ResourceLoader<T extends Resource> extends ObjectLoader<T>   
 updateStatus(s: Status)

* Загрузчик поста

export class PostLoader extends ResourceLoader<Post>   
 get endpoint(): string

static *postUrl* = new URL(config.endpoints.post, config.server);  
 static async *makePost*(text: string, files: FileList | null): Promise<number>

* Загрузчик фотографии (достаточно использовать url)

export class PhotoLoader extends ObjectLoader<Blob>   
 get endpoint(): string

* Загрузчик информации о фотографии

export class PhotoInfoLoader extends ResourceLoader<Photo>   
 get endpoint(): string

Для работы с Вебсокетом на стороне клиента реализован класс SocketManager

export class SocketManager   
 private socket: Socket<ServerToClientEvents, ClientToServerEvents>;  
 constructor()  
 addListener(cb: (user: UserData) => void)   
 removeListener(cb: (user: UserData) => void)

Взаимодействие пользователя с системой осуществляет класс UserContoller.

export class UserController   
 readonly self = new URL(config.endpoints.self, config.server);  
 id: number | null = null;  
 loader: UserLoader | null = null;async fetchId()  
get isAdmin(): boolean static readonly *signUpUrl* = new URL(config.endpoints.signUp, config.server);  
 static async *signUp*(user: UserSignUpData): Promise<boolean> static readonly *loginUrl* = new URL(config.endpoints.login, config.server);  
 static async *login*(user: UserAuthData): Promise<boolean>   
 readonly avatarUrl = new URL(config.endpoints.avatar, config.server);  
 async updateAvatar(file: File)   
 async deleteAvatar()  
 getFriendUrl(id: number): URL   
 async requestFriend(loader: UserLoader)   
 async declineFriend(loader: UserLoader)

**Компоненты интерфейса**

* Миксин для компонентов списка, содержит список чисел (id объектов)

в качестве свойства (prop)

export default class List extends Vue   
 @Prop({ required: true }) readonly list!: number[];

* Компонент списка фотографий пользователя

export default class PhotosList extends List

* Компонент списка постов пользователя

export default class PostsList extends List

* Компонент списка пользователей

export default class UsersList extends List

* Компонент переключателя блокировки ресурса (поста и фотографий)

export default class BanSwitcher extends Vue   
 @InjectReactive() readonly user!: UserController;  
 private Status = Status;  
 @Prop({ required: true }) status!: Status;  
 @Emit("switched")  
 private switchStatus(s: Status): Status

* Компонент карусели фотографий прилагаемых к посту

export default class Carousel extends Vue   
 @Prop({ required: true }) readonly list!: number[];  
 private id = 0;  
 $refs!: {  
 indicators: HTMLButtonElement[];  
 images: HTMLDivElement[];  
 };  
 private mounted()

* Компонент фотографии пользователя

export default class Photo extends Vue   
 private Status = Status;  
 private Role = Role;  
 @InjectReactive() readonly toaster!: Toaster | null;  
 @InjectReactive() readonly user!: UserController;  
 @Prop({ required: true }) readonly id!: number;  
 private photo: PhotoLoader | null = null;  
 private photoInfo: PhotoInfoLoader | null = null;  
 private async mounted()  
 private switchStatus(s: Status)

* Компонент новости пользователя

export default class Post extends Vue   
 private Status = Status;  
 private Role = Role;  
 @InjectReactive() readonly user!: UserController;  
 @InjectReactive() readonly toaster!: Toaster | null;  
 @Prop({ type: *Number*, required: true }) readonly id!: number;  
 private post: PostLoader | null = null;  
 private userLoader: UserLoader | null = null;  
 private async mounted()  
 private switchStatus(s: Status)

* Компонент полей формы персональной информации

export default class PersonalDataRedactor extends Vue   
 @Model("data-changed", { required: true }) pd!: PersonalData;

* Компонент полей формы статусной информации

export default class StatusDataRedactor extends Vue   
 private Role = Role;  
 private Status = Status;  
 @Model("status-changed", { required: true }) sd!: UserStatusData;

* Компонент модального окна редактирования пользователя, позволяет

редактировать персональную информацию, статус и роль пользователя.

export default class RedactModal extends Vue   
 private modal!: Modal;  
 private loader: UserLoader | null = null;  
 private buffer: UserData | null = null;  
 private admin = false;  
 $refs!: {  
 form: HTMLFormElement;  
 };  
 private mounted()  
 show(loader: UserLoader, admin = false)   
 private submit()

* Компонент модального окна формы регистрации

export default class SignUpModal extends Vue   
 @InjectReactive() readonly toaster!: Toaster | null;  
 private modal!: Modal;  
 private usud: UserSignUpData | null = null;  
 $refs!: {  
 form: HTMLFormElement;  
 };  
 private mounted()  
 show()  
 private async submit()

* Компонент модального окна формы добавления поста

export default class PostModal extends Vue   
 private modal!: Modal;  
 private text = "";  
 private loader: PostsLoader | null = null;  
 $refs!: {  
 form: HTMLFormElement;  
 files: HTMLInputElement;  
 };  
 private mounted()  
 show(loader: PostsLoader)   
 private async submit()

* Компонент всплывающих сообщений

export default class Toaster extends Vue   
 private Status = Status;  
 @Prop({ default: 1500 }) interval!: number;  
 private msg = "";  
 private status: Status = Status.*UNCONFIRMED*;  
 private visible = false;  
 private timerId = 0;  
 show(status: Status, msg: string)

* Компонент карточки пользователя. В ней можно перейти к странице

пользователя, либо вызвать окно настроек пользователя.

export default class UserCard extends Vue   
 private Views = Views;  
 @InjectReactive() readonly redact!: RedactModal | null;  
 @InjectReactive() readonly user!: UserController;  
 @Prop({ required: true }) readonly id!: number;  
 @Prop({ default: false }) readonly configurable!: boolean;  
 private loader: UserLoader | null = null;  
 private async mounted()  
 private settings()

private requestFriend()

private declineFriend()

* Компонент иконки пользователя

export default class UserThumbnail extends Vue   
 private Views = Views;  
 @Prop({ required: true }) readonly loader!: UserLoader;

* Главный компонент приложения, предоставляет инъекцию

компонентов для модального окна регистрации и всплывающих сообщений.

export default class App extends Vue   
 private Views = Views;  
 $refs!: {  
 toaster: Toaster;  
 signUp: SignUpModal;  
 };  
 private mounted()  
 @ProvideReactive() toaster: Toaster | null = null;  
 @ProvideReactive() signUp: SignUpModal | null = null;

**Компоненты представления роутера**

* Страница авторизации пользователя, неавторизованный пользователь

переадресовывается на неё.

export default class LoginView extends Vue   
 @InjectReactive() readonly toaster!: Toaster | null;  
 private user: UserAuthData = { email: "", password: "" };  
 private valid = true;  
 private readonly url = new URL(  
 config.endpoints.login,  
 config.server  
 ).toString();  
 private async submit()  
 private reset()

* Главный компонент для авторизованного пользователя, предоставляет

инъекции контроллера пользователя, менеджера сокета и компонентов модальных окон.

export default class Network extends Vue {  
 @InjectReactive() readonly toaster!: Toaster | null;  
 private Views = Views;  
 private get routes()  
 $refs!: {  
 avatarUpdate: UpdateAvatarModal;  
 redact: RedactModal;  
 postModal: PostModal;  
 };  
 private async created()  
 private beforeDestroy()  
 private notification = require("../assets/notification.mp3");private postNotification(user: UserData) @ProvideReactive() user: UserController = new UserController();  
@ProvideReactive() socket: SocketManager = new SocketManager();  
 @ProvideReactive() redact: RedactModal | null = null;  
 @ProvideReactive() avatarUpdate: UpdateAvatarModal | null = null;  
 @ProvideReactive() postModal: PostModal | null = null;

* Страница списка всех пользователей

export default class UsersListView extends Vue   
 private users: UsersListLoader = new UsersListLoader();

private mounted()

* Страница профиля пользователя, содержит в себе разделы навигации:

новости, фото, друзья, новости друзей

export default class Profile extends Vue {  
 private id = 0;  
 private loader: ListLoader | null = null;  
 @InjectReactive() readonly user!: UserController;  
 readonly categories = [  
 [Views.*USER\_POSTS*, "Новости"],  
 [Views.*USER\_PHOTOS*, "Фото"],  
 [Views.*USER\_FRIENDS*, "Друзья"],  
 [Views.*USER\_FRIENDSPOSTS*, "Новости друзей"],  
 ];  
 private mounted()  
 @Watch("$route")  
 onRouteChange(to: Route)

* Страница своего профиля

export default class MyProfile extends Vue {  
 @InjectReactive() readonly user!: UserController;  
 @InjectReactive() readonly avatarUpdate!: UpdateAvatarModal;  
 @InjectReactive() readonly postModal!: PostModal;  
 posts: PostsLoader | null = null;  
 private created()  
 private makePost()

* Страница списка друзей

export default class Friends extends Vue {  
 @InjectReactive() readonly user!: UserController;  
}

* Страница новостей

export default class News extends Vue {  
 @InjectReactive() readonly user!: UserController;  
 @InjectReactive() readonly socket!: SocketManager;  
 posts: FriendsPostsLoader | null = null;  
 private created()  
 private beforeDestroy()  
 private postsUpdate(user: UserData)

* Страница своих фотографий

export default class Photos extends Vue   
 @InjectReactive() readonly user!: UserController;  
 photos: PhotosLoader | null = null;  
 private created()

**Примеры экранных форм**.

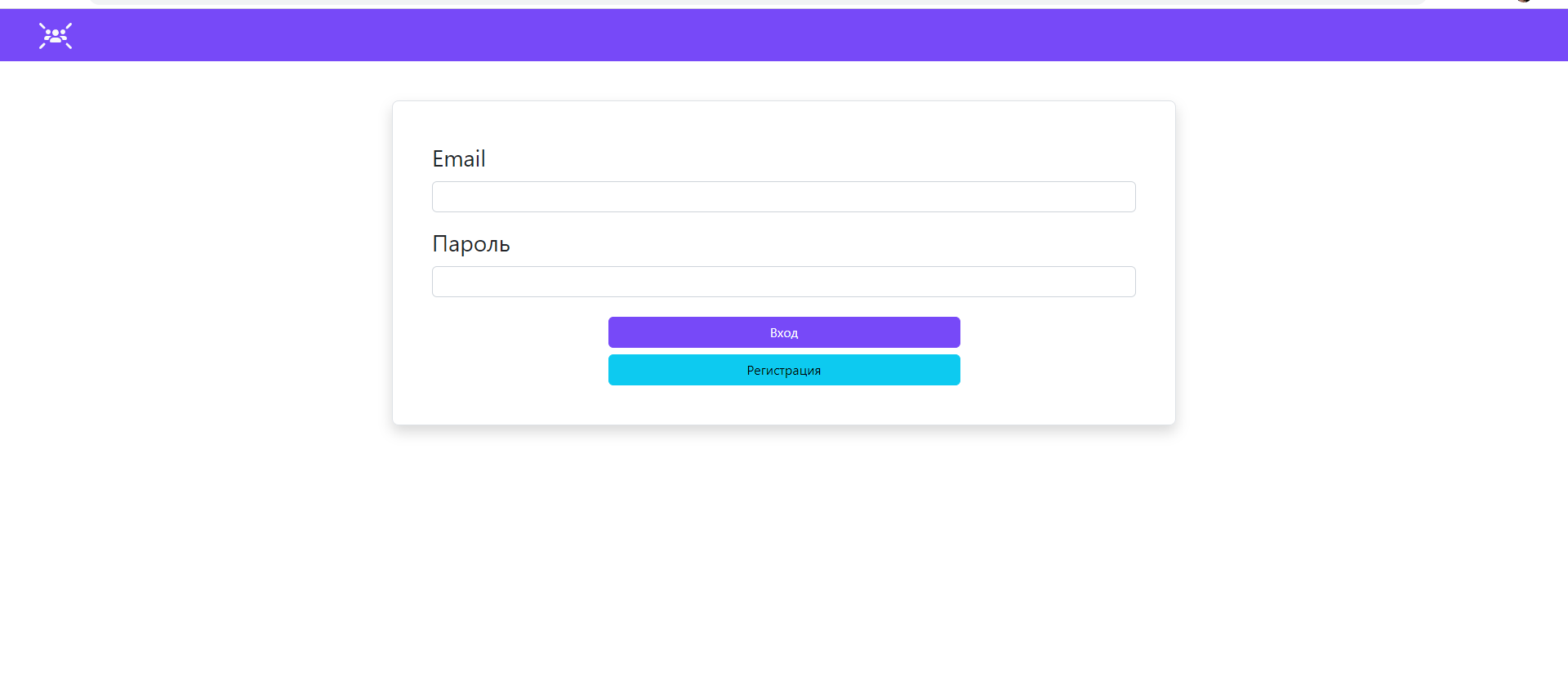


Рисунок : Стартовая страница

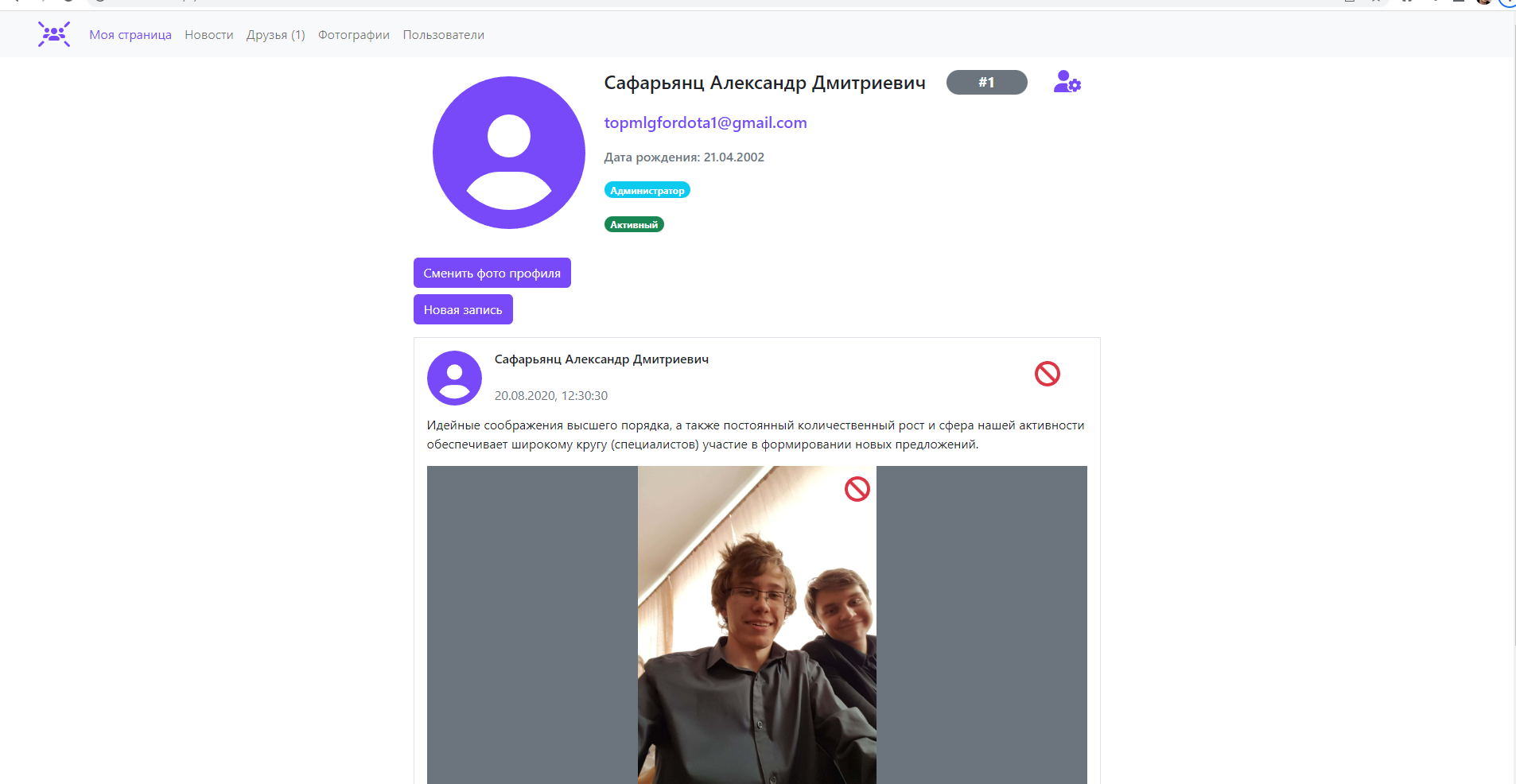


Рисунок : Профиль текущего пользователя

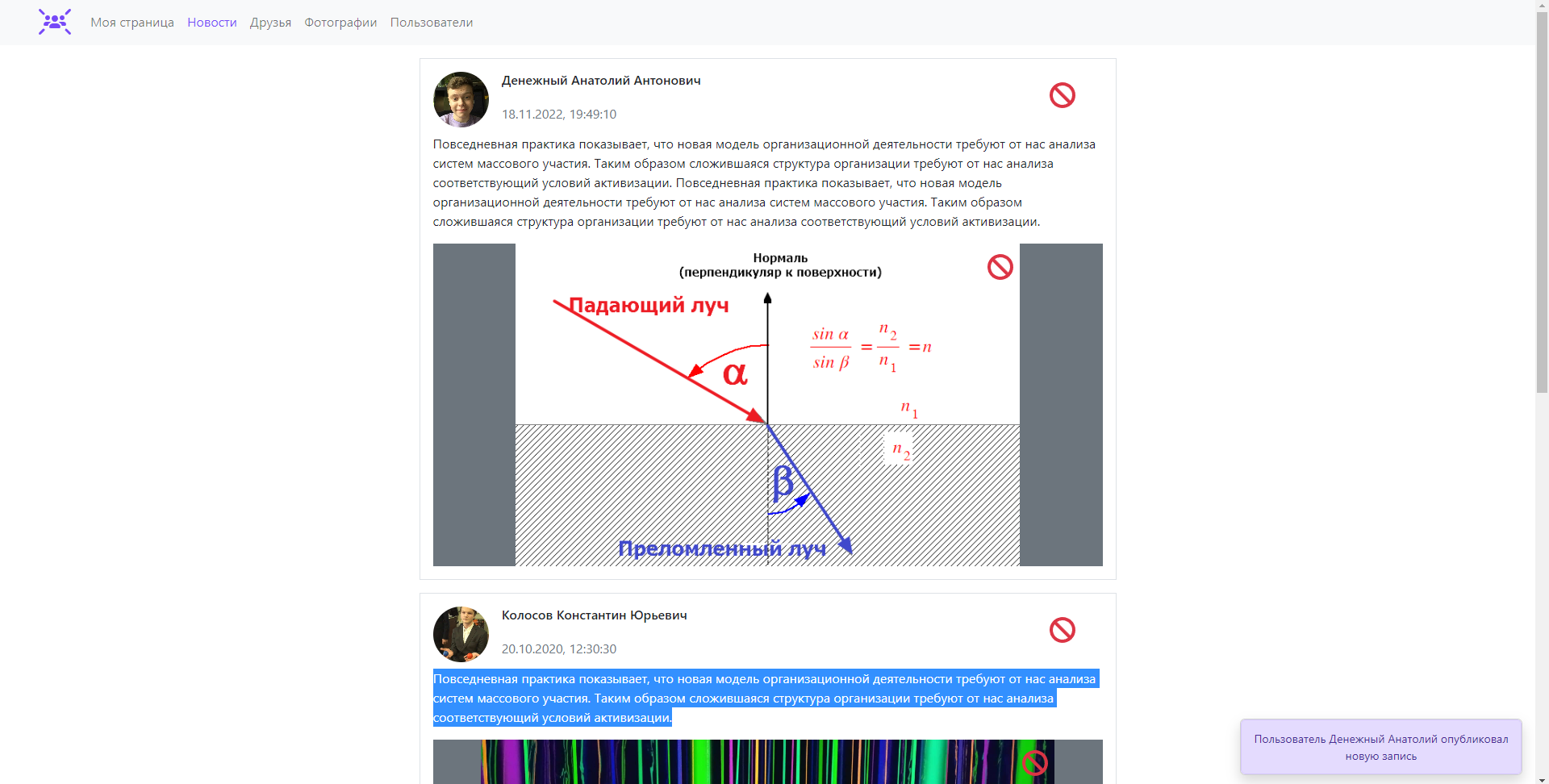


Рисунок : Новости и оповещение

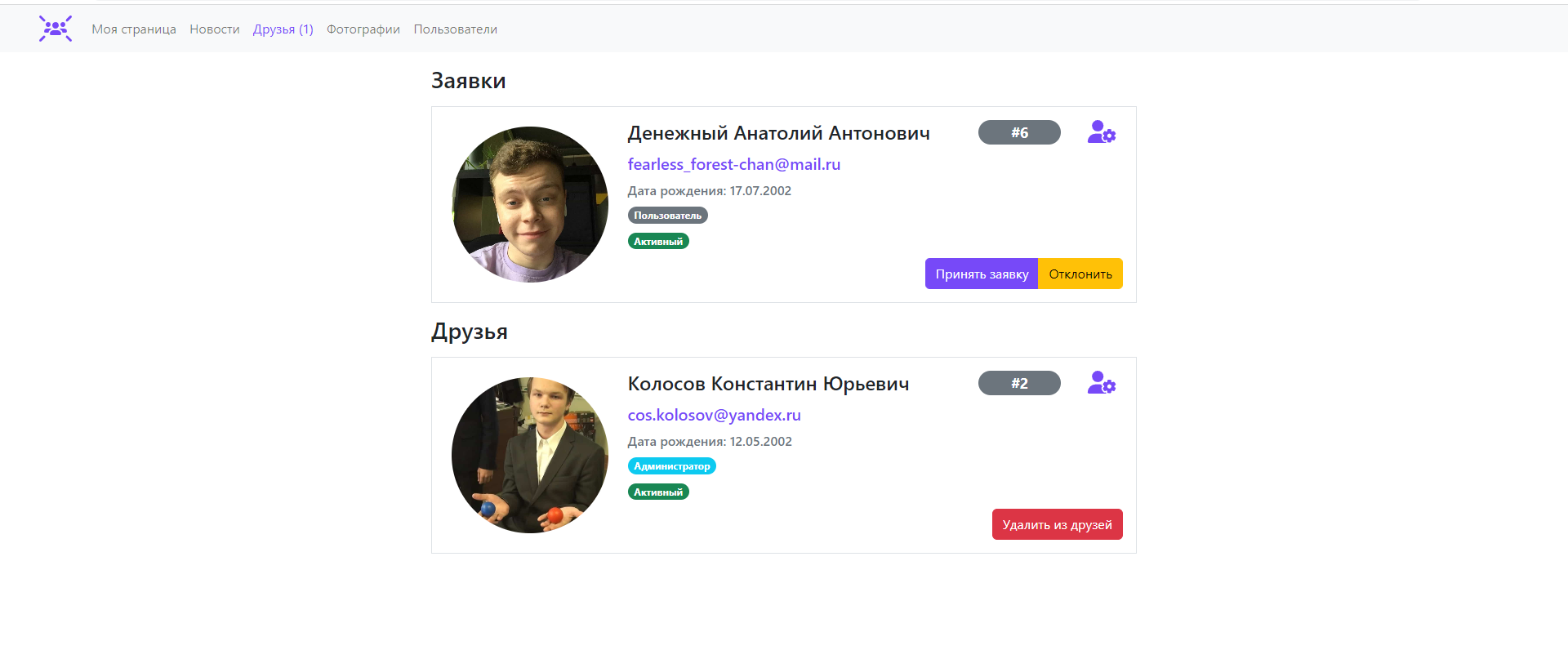


Рисунок : Друзья

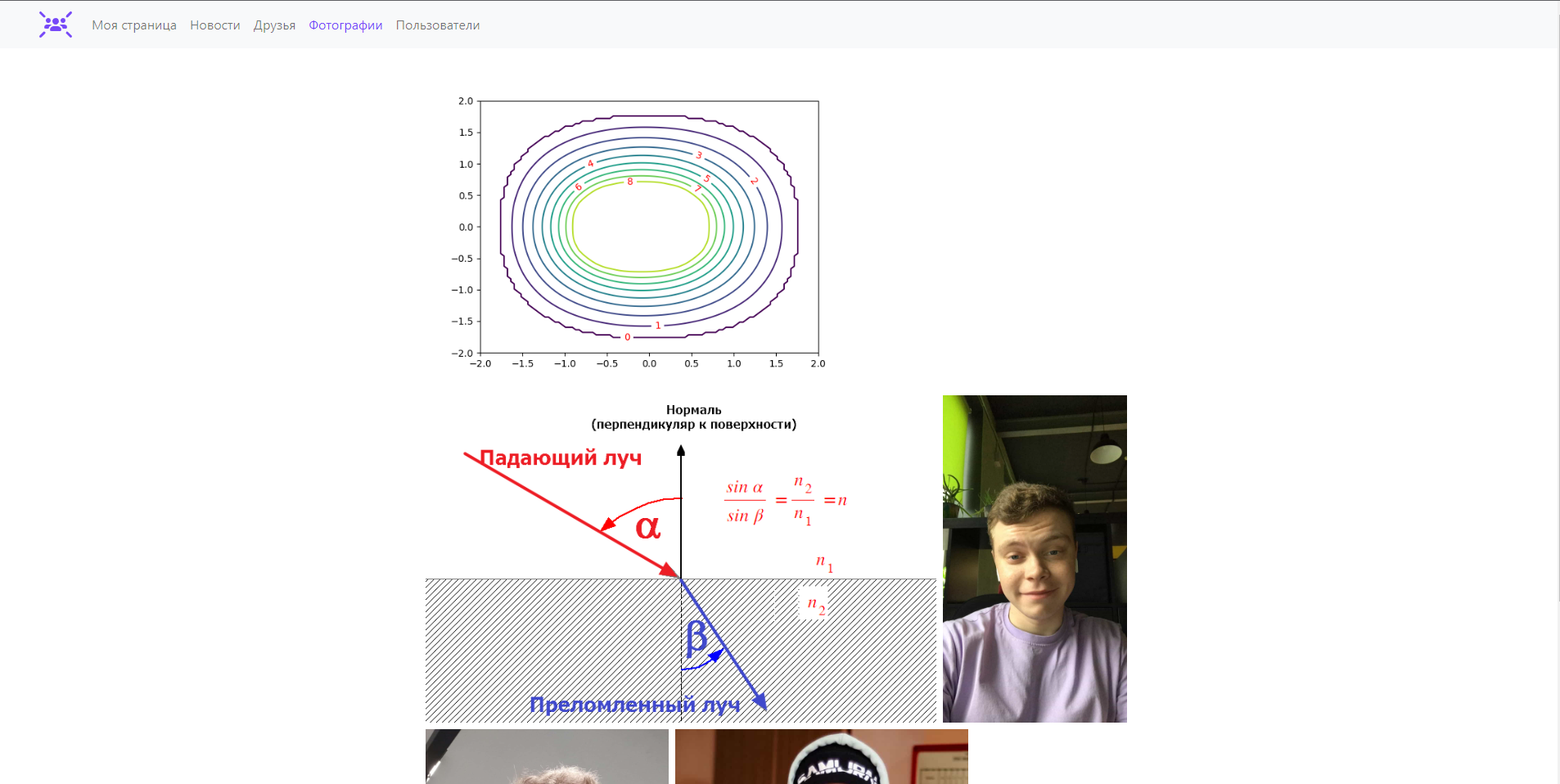


Рисунок : фотографии

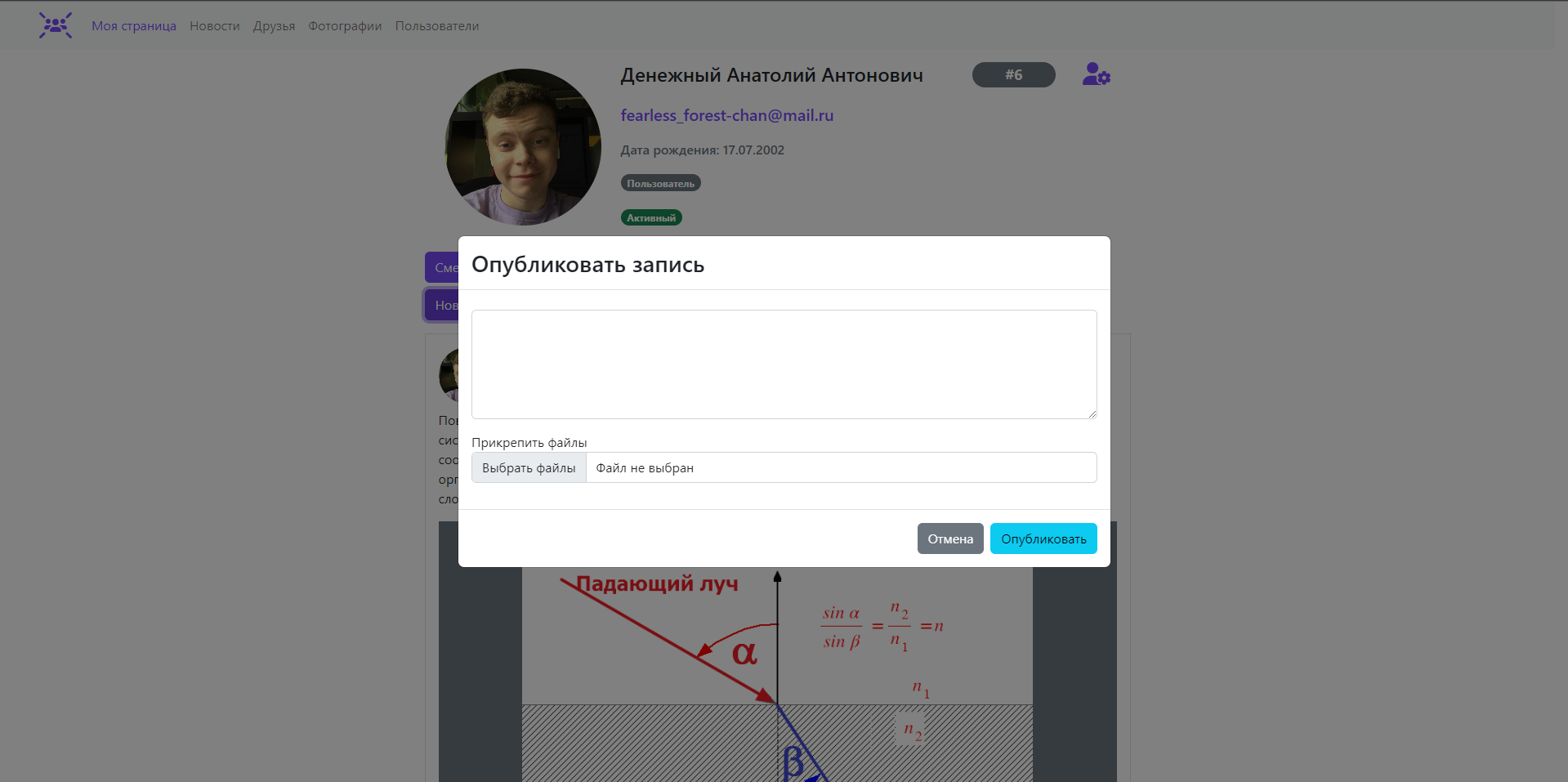


Рисунок : Добавление новости

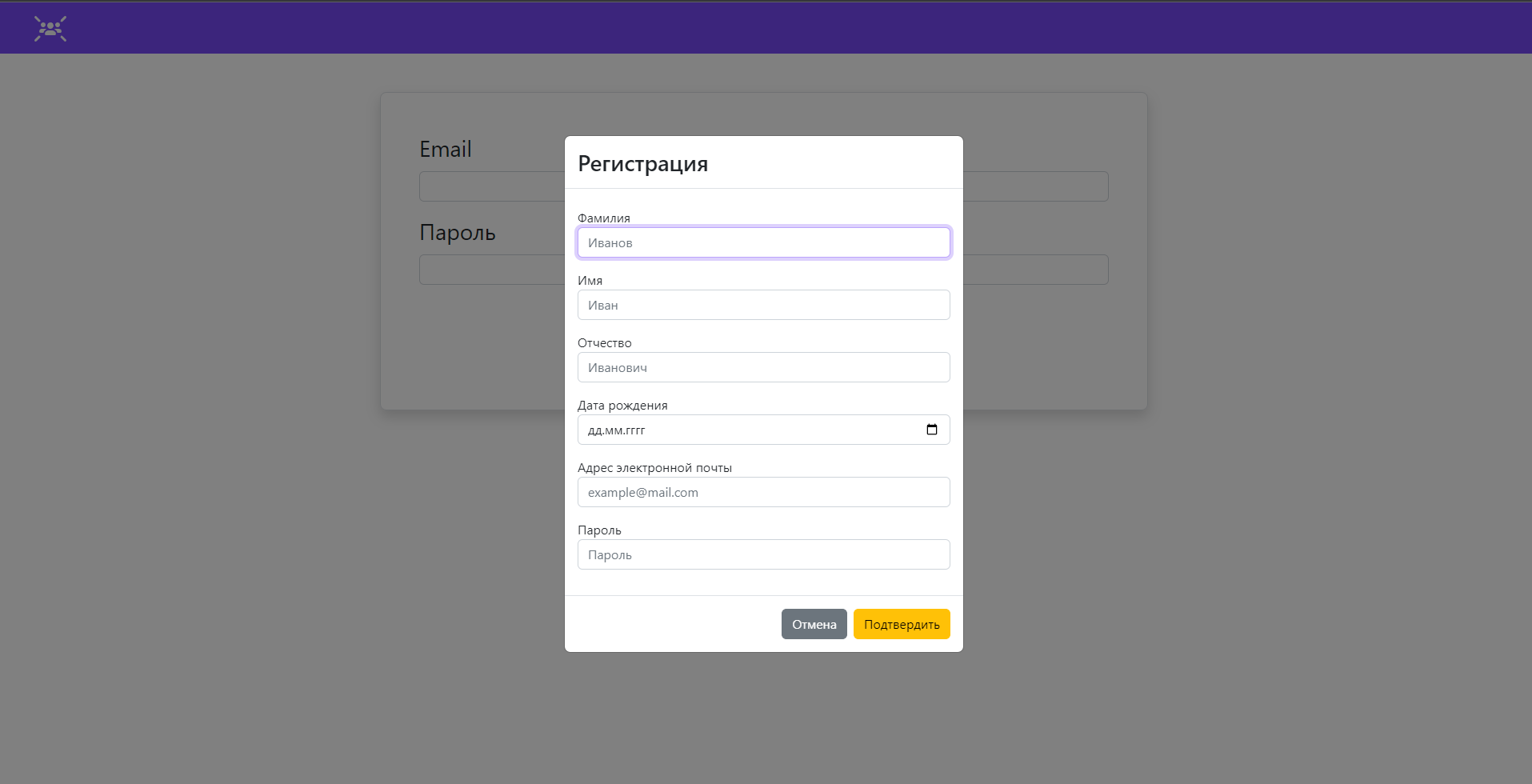


Рисунок : Регистрация

**Логгирование**

В клиентском приложении был создан модуль с плагином RollbarPlugin, который интегрируется во Vue приложение и предоставляет всем компонентам глобальный объект логгирования $rollbar. Также в конфигурацию Vue установлен обработчик ошибок, который логгирует входящие ошибки (все необработанные ошибки приложения) и пробрасывает их дальше.

Сообщения успешно доставляются в сервис библиотеки rollbar.

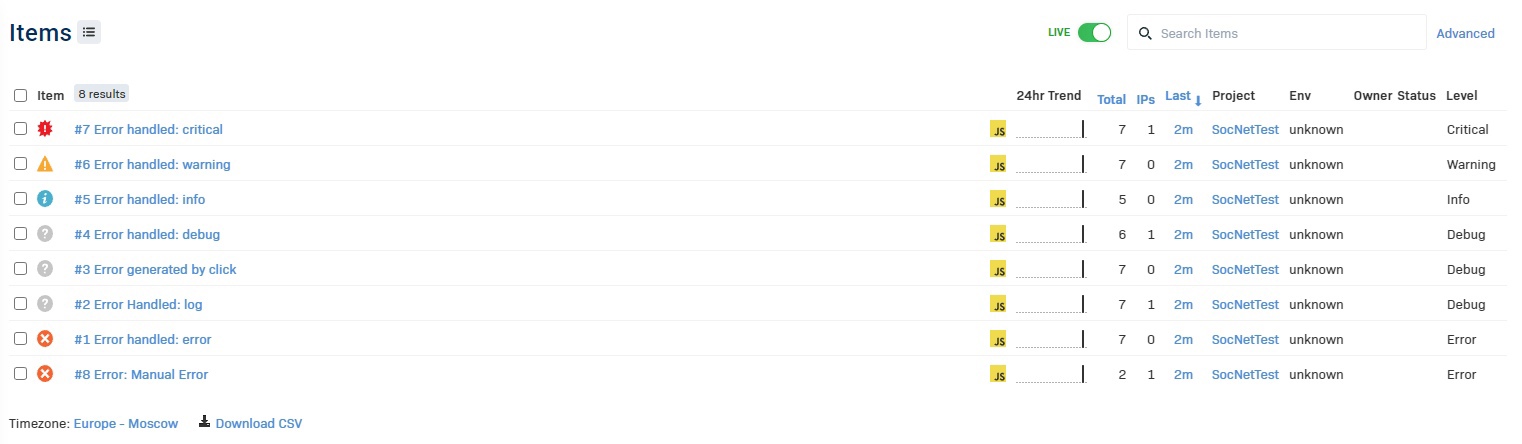


Рисунок : сообщения rollbar-а

**Тестирование**

Для тестирования бизнес-логики серверного приложения были написаны модульные тесты с использованием фреймворка Jest. Приложение успешно проходит все тесты.

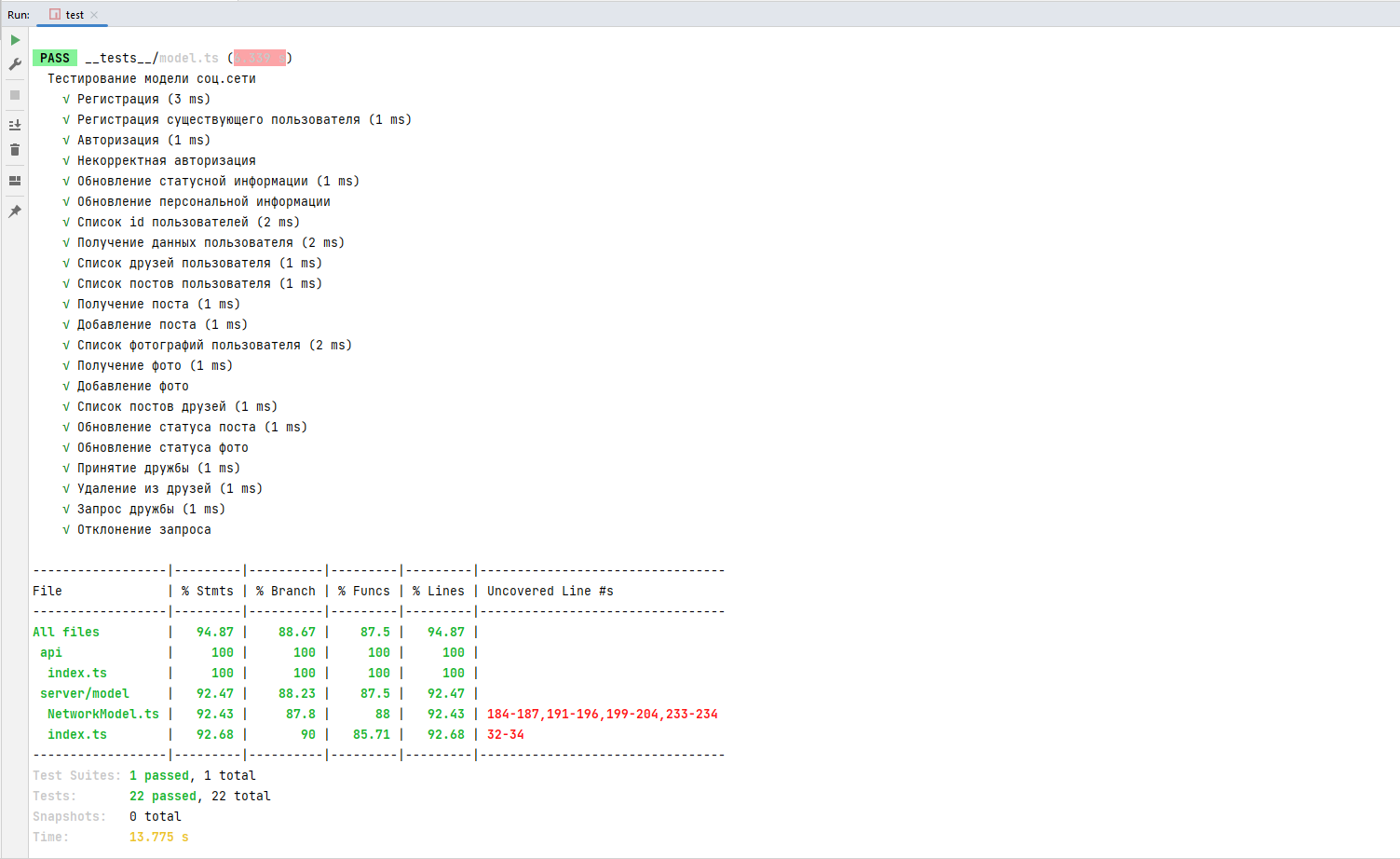


Рисунок : тестирование

**Выводы.**

В ходе было создано веб приложение социальной сети обеспечивающее доступ к сервису как пользователям, так и администраторам. Былое реализовано серверное приложение с REST интерфейсом, предоставляющее также интерфейс WebSocket для отправки сообщения об обновлениях ресурсов. Для соответствующего серверного приложения на основе созданного макета было реализовано клиентское приложение, взаимодействующее с сервером посредством ajax-запросов и принимающее WebSocket сообщения.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

***Файл api/index.ts***

*// Роли пользователей*import { EventsMap } from "socket.io/dist/typed-events";  
  
export enum Role {  
 *// Пользователь  
 USER*,  
 *// Администратор  
 ADMIN*,  
}  
  
*// Статус пользователя / ресурса*export enum Status {  
 *// Активный  
 ACTIVE*,  
 *// Неподтверждённый  
 UNCONFIRMED*,  
 *// Заблокированный  
 BLOCKED*,  
}  
  
*// Индексируемый объект*export interface Indexed {  
 id: number;  
}  
  
*// Объект со статусом*export interface StatusData {  
 status: Status;  
}  
  
*// Статусные данные пользователя*export interface UserStatusData extends StatusData {  
 role: Role;  
}  
  
*// Персональные данные пользователя*export interface PersonalData {  
 surname: string;  
 name: string;  
 lastName: string;  
 email: string;  
 birthDate: string;  
}  
  
*// Данные пользователя соц. сети*export interface UserData extends Indexed, UserStatusData, PersonalData {  
 friends: number[];  
 friendsRequests: number[];  
 photoId: number | null;  
}  
  
*// Объект пользователя на сервере*export interface User extends UserData {  
 password: string;  
}  
  
*// Объект ресурса (фотография/пост)*export interface Resource extends Indexed, StatusData {  
 userId: number;  
 time: string;  
}  
  
*// Фотография*export interface Photo extends Resource {  
 file: string;  
}  
  
*// Новость*export interface Post extends Resource {  
 text: string;  
 photosId: number[];  
}  
  
*// Авторизационные данные*export interface UserAuthData {  
 email: string;  
 password: string;  
}  
  
*// Данные для регистрации*export interface UserSignUpData extends PersonalData, UserAuthData {}  
  
*// Настройки клиент-серверного взаимодействия*export const config = {  
 *// Адрес хоста сервера* serverHost: "localhost",  
 *// Адрес вебсокета* webSocket: "http://localhost:3000/",  
 *// Адрес api* server: "http://localhost:3000/api/",  
 *// Адрес клиентского приложения* client: "http://localhost:8080",  
 *// Точки запросов* endpoints: {  
 *// Получение пользователем своего объекта пользователя* self: "self/",  
 *// Вход в систему* login: "login/",  
 *// Регистрация* signUp: "signup/",  
 *// Список всех пользователей* usersList: "users/",  
 *// Информация о пользователе (по id)* user: "user/",  
 *// Загрузка фотографии профиля* avatar: "avatar/",  
 *// Список id постов пользователя* postsList: "posts/",  
 *// Получение / публикация поста* post: "post/",  
 *// Список id фотографий пользователя* photosList: "photos/",  
 *// Получение фотографии* photo: "photo/",  
 *// Информация о фотографии* photoInfo: "photoinfo/",  
 *// Список id друзей пользователя* friends: "friends/",  
 *// Список id постов друзей* friendsPosts: "friendsposts/",  
 *// Управление друзьями* friend: "friend/",  
 },  
};  
  
*// События отправляемые сервером по Вебсокету*export interface ServerToClientEvents {  
 post: (user: UserData) => void;  
}  
  
export type ClientToServerEvents = EventsMap;

***Файл \_\_tests\_\_/model.ts***

import { beforeEach, describe, expect, test } from "@jest/globals";  
import { load, NetworkModel } from "../server/model";  
import \* as \_ from "lodash";  
import {  
 PersonalData,  
 Photo,  
 Post,  
 Role,  
 Status,  
 UserStatusData,  
 User,  
 StatusData,  
} from "../api";  
  
const storagePath = "./storage",  
 users = load("users.json", storagePath) as User[],  
 photos = load("photos.json", storagePath) as Photo[],  
 posts = load("posts.json", storagePath) as Post[];  
  
let model: NetworkModel | null = null;  
  
describe("Тестирование модели соц.сети", () => {  
 beforeEach(() => {  
 model = new NetworkModel(users, photos, posts, storagePath);  
 });  
  
 test("Регистрация", () => {  
 expect(  
 model?.signUp({  
 name: "Игорь",  
 surname: "Летов",  
 lastName: "Фёдорович",  
 birthDate: "10.09.1964",  
 email: "igor@mail.ru",  
 password: "pryg\_skok",  
 })  
 ).toBe(9);  
 });  
  
 test("Регистрация существующего пользователя", () => {  
 expect(  
 model?.signUp({  
 name: "Владислав",  
 surname: "Болкунов",  
 lastName: "Олегович",  
 birthDate: "08.10.2002",  
 email: "vladbolkunovv@gmail.com",  
 password: "vladik",  
 })  
 ).toBe(null);  
 });  
  
 test("Авторизация", () => {  
 expect(  
 model?.authorize({ email: "vladbolkunovv@gmail.com", password: "vladik" })  
 ).not.toBe(null);  
 });  
  
 test("Некорректная авторизация", () => {  
 expect(  
 model?.authorize({  
 email: "vladbolkunovv@gmail.com",  
 password: "password",  
 })  
 ).toBe(null);  
 });  
  
 test("Обновление статусной информации", () => {  
 model?.updateUserStatus(7, { role: Role.ADMIN, status: Status.BLOCKED });  
 const user = model?.getUserData(7);  
 expect({  
 status: user?.status,  
 role: user?.role,  
 } as UserStatusData).toEqual({  
 status: Status.BLOCKED,  
 role: Role.ADMIN,  
 } as UserStatusData);  
 });  
  
 test("Обновление персональной информации", () => {  
 const data: PersonalData = {  
 name: "Илья",  
 surname: "Комаров",  
 lastName: "Владимирович",  
 email: "example@mail.com",  
 birthDate: "1970-01-01",  
 };  
 model?.updatePersonal(7, data);  
 const user = model?.getUserData(7);  
 expect({  
 name: user?.name,  
 surname: user?.surname,  
 lastName: user?.lastName,  
 email: user?.email,  
 birthDate: user?.birthDate,  
 } as PersonalData).toEqual(data);  
 });  
  
 test("Список id пользователей", () => {  
 expect(model?.getUsers()).toEqual(\_.range(1, 9));  
 });  
  
 test("Получение данных пользователя", () => {  
 const user = model?.getUserData(7);  
 expect(user).toEqual({  
 id: 7,  
 name: "Владислав",  
 surname: "Болкунов",  
 lastName: "Олегович",  
 email: "vladbolkunovv@gmail.com",  
 birthDate: "2002-10-08",  
 status: 0,  
 role: 0,  
 friends: [2, 6, 8],  
 friendsRequests: [],  
 photoId: 14,  
 });  
 });  
  
 test("Список друзей пользователя", () => {  
 expect(model?.getFriends(7)).toEqual([2, 6, 8]);  
 });  
  
 test("Список постов пользователя", () => {  
 expect(model?.getPosts(7)).toEqual([3, 2]);  
 });  
  
 test("Получение поста", () => {  
 expect(model?.getPost(2)).toEqual({  
 id: 2,  
 userId: 7,  
 text: "We are anonymous.",  
 photosId: [4],  
 status: 2,  
 time: "2020-06-23T15:30:30",  
 } as Post);  
 });  
  
 test("Добавление поста", () => {  
 expect(model?.addPost(2, "Post text", [1, 2, 3])).toBe(8);  
 });  
  
 test("Список фотографий пользователя", () => {  
 expect(model?.getPhotos(7)).toEqual([14, 7, 5, 4]);  
 });  
  
 test("Получение фото", () => {  
 expect(model?.getPhoto(5)).toEqual({  
 id: 5,  
 userId: 7,  
 file: "5.jpg",  
 status: 2,  
 time: "2020-06-20T12:30:30",  
 } as Photo);  
 });  
  
 test("Добавление фото", () => {  
 expect(model?.addPhoto(2, "path")).toBe(15);  
 });  
  
 test("Список постов друзей", () => {  
 expect(model?.getFriendsPosts(7)).toEqual([6, 5]);  
 });  
  
 test("Обновление статуса поста", () => {  
 model?.updatePostStatus(3, { status: Status.BLOCKED });  
 const post = model?.getPost(3);  
 expect({ status: post?.status } as StatusData).toEqual({  
 status: 2,  
 } as StatusData);  
 });  
  
 test("Обновление статуса фото", () => {  
 model?.updatePhotoStatus(4, { status: Status.BLOCKED });  
 const photo = model?.getPhoto(4);  
 expect({ status: photo?.status } as StatusData).toEqual({  
 status: 2,  
 } as StatusData);  
 });  
  
 test("Принятие дружбы", () => {  
 model?.friendRequest(4, 7);  
 const u1 = model?.getUserData(4),  
 u2 = model?.getUserData(7);  
 expect(u1?.friends?.includes(7) && u2?.friends?.includes(4)).toBe(true);  
 });  
  
 test("Удаление из друзей", () => {  
 model?.friendDecline(7, 6);  
 const u1 = model?.getUserData(7),  
 u2 = model?.getUserData(6);  
 expect(!u1?.friends?.includes(6) && !u2?.friends?.includes(7)).toBe(true);  
 });  
  
 test("Запрос дружбы", () => {  
 model?.friendRequest(7, 1);  
 const u = model?.getUserData(1);  
 expect(u?.friendsRequests?.includes(7)).toBe(true);  
 });  
  
 test("Отклонение запроса", () => {  
 model?.friendDecline(1, 6);  
 const u = model?.getUserData(1);  
 expect(!u?.friendsRequests?.includes(6)).toBe(true);  
 });  
});

***Файл server/model/NetworkModel.ts***

import {  
 PersonalData,  
 Photo,  
 Post,  
 Resource,  
 Role,  
 Status,  
 StatusData,  
 User,  
 UserAuthData,  
 UserData,  
 UserSignUpData,  
 UserStatusData,  
} from "../../api";  
import { ModelCollection, nowISO } from "./index";  
import fs from "fs/promises";  
import path from "path";  
  
*// Логика соц сети*export class NetworkModel {  
 private users: ModelCollection<User> = new ModelCollection<User>();  
 private photos: ModelCollection<Photo> = new ModelCollection<Photo>();  
 private posts: ModelCollection<Post> = new ModelCollection<Post>();  
 readonly storagePath: string;  
  
 constructor(  
 users: User[],  
 photos: Photo[],  
 posts: Post[],  
 storagePath: string  
 ) {  
 users.forEach((u) => this.users.add(u));  
 photos.forEach((p) => this.photos.add(p));  
 posts.forEach((p) => this.posts.add(p));  
 this.storagePath = storagePath;  
 }  
  
 *// Регистрация пользователя* signUp(user: UserSignUpData): number | null {  
 if (!this.users.list.find((u) => u.email == user.email))  
 return this.users.add({  
 id: 0,  
 role: Role.USER,  
 status: Status.UNCONFIRMED,  
 photoId: null,  
 friends: [],  
 friendsRequests: [],  
 name: user.name,  
 surname: user.surname,  
 lastName: user.lastName,  
 birthDate: user.birthDate,  
 email: user.email,  
 password: user.password,  
 });  
 else return null;  
 }  
  
 *// Авторизация* authorize(data: UserAuthData): User | null {  
 const user = this.users.list.find((user) => user.email == data.email);  
 if (user && user.password == data.password) return user;  
 return null;  
 }  
  
 *// Обновление статусной информации (роль, статус)* updateUserStatus(id: number, stat: UserStatusData) {  
 const user = this.getUser(id);  
 if (user) [user.status, user.role] = [stat.status, stat.role];  
 }  
  
 *// Обновление персональной информации* updatePersonal(id: number, data: PersonalData) {  
 const user = this.getUser(id);  
 if (user)  
 [user.surname, user.name, user.lastName, user.email, user.birthDate] = [  
 data.surname,  
 data.name,  
 data.lastName,  
 data.email,  
 data.birthDate,  
 ];  
 }  
  
 *// Получение объекта пользователя (вместе с паролем)* private getUser(id: number): User | null {  
 return this.users.find(id);  
 }  
  
 *// Список id пользователей* getUsers(): number[] {  
 return this.users.getIds();  
 }  
  
 *// Данные пользователя (без пароля)* getUserData(id: number): UserData | null {  
 const user = this.getUser(id);  
 return user  
 ? ({  
 ...user,  
 password: undefined,  
 } as UserData)  
 : null;  
 }  
  
 *// id друзей пользователя* getFriends(id: number): number[] | null {  
 return this.getUser(id)?.friends ?? null;  
 }  
  
 private getItems<T extends Resource>(userId: number, items: T[]): number[] {  
 return items.filter((item) => item.userId == userId).map((item) => item.id);  
 }  
  
 *// Список id постов пользователя* getPosts(userId: number): number[] {  
 return this.getItems(userId, this.posts.list).reverse();  
 }  
  
 *// Получение поста* getPost(id: number): Post | null {  
 return this.posts.find(id);  
 }  
  
 *// Добавление поста* addPost(id: number, text: string, photosId: number[]): number {  
 return this.posts.add({  
 id: 0,  
 text: text,  
 userId: id,  
 photosId: photosId,  
 time: nowISO(),  
 status: Status.ACTIVE,  
 });  
 }  
  
 *// Список id фотографий пользователя* getPhotos(userId: number): number[] {  
 return this.getItems(userId, this.photos.list).reverse();  
 }  
  
 *// Получение фотографии* getPhoto(id: number): Photo | null {  
 return this.photos.find(id);  
 }  
  
 *// Добавление фотографии* addPhoto(id: number, path: string): number {  
 return this.photos.add({  
 id: 0,  
 status: Status.ACTIVE,  
 time: nowISO(),  
 file: path,  
 userId: id,  
 });  
 }  
  
 *// Список id постов друзей* getFriendsPosts(id: number): number[] {  
 return (  
 this.getUser(id)  
 ?.friends.map((user) => this.getPosts(user))  
 .flat() ?? []  
 ).sort((a, b) => b - a);  
 }  
  
 private updateResourceStatus<T extends Resource>(  
 res: T | null,  
 sd: StatusData  
 ) {  
 if (res) res.status = sd.status;  
 }  
  
 *// Обновление статуса поста* updatePostStatus(id: number, sd: StatusData) {  
 this.updateResourceStatus(this.getPost(id), sd);  
 }  
  
 *// Обновление статуса фотографии* updatePhotoStatus(id: number, sd: StatusData) {  
 this.updateResourceStatus(this.getPhoto(id), sd);  
 }  
  
 deletePhoto(id: number) {  
 const photo = this.getPhoto(id);  
 if (photo) fs.unlink(path.resolve(this.storagePath, photo.file));  
 this.photos.remove(id);  
 }  
  
 *// Обновление фотографии профиля* updateAvatar(id: number, path: string) {  
 const user = this.getUser(id);  
 if (user) {  
 if (user.photoId) this.deletePhoto(user.photoId);  
 user.photoId = this.addPhoto(id, path) ?? user.photoId;  
 }  
 }  
  
 deleteAvatar(id: number) {  
 const user = this.getUser(id);  
 if (user?.photoId) {  
 this.deletePhoto(user.photoId);  
 user.photoId = null;  
 }  
 }  
  
 *// Запрос/принятие дружбы* friendRequest(id: number, friendId: number) {  
 const u1 = this.getUser(id),  
 u2 = this.getUser(friendId);  
 if (u1 && u2) {  
 if (u1.friendsRequests.includes(u2.id)) {  
 u1.friendsRequests.splice(u1.friendsRequests.indexOf(u2.id), 1);  
 u1.friends.push(u2.id);  
 u2.friends.push(u1.id);  
 } else {  
 u2.friendsRequests.push(u1.id);  
 }  
 }  
 }  
  
 *// Отклонение запроса дружбы / удаление из друзей* friendDecline(id: number, friendId: number) {  
 if (id !== friendId) {  
 const u1 = this.getUser(id),  
 u2 = this.getUser(friendId);  
 if (u1 && u2) {  
 if (u1.friends.includes(u2.id)) {  
 u1.friends.splice(u1.friends.indexOf(u2.id), 1);  
 u2.friends.splice(u2.friends.indexOf(u1.id), 1);  
 } else if (u1.friendsRequests.includes(u2.id)) {  
 u1.friendsRequests.splice(u1.friendsRequests.indexOf(u2.id), 1);  
 } else if (u2.friendsRequests.includes(u1.id)) {  
 u2.friendsRequests.splice(u2.friendsRequests.indexOf(u1.id), 1);  
 }  
 }  
 }  
 }  
}

***Файл server/model/index.ts***

import { Indexed } from "../../api";  
import fs from "fs";  
import path from "path";  
  
*// Текущее время в формате iso*export function *nowISO*() {  
 return new Date().toISOString();  
}  
  
*// Загрузка JSON файла*export function *load*<T>(file: string, storagePath: string): T {  
 return JSON.parse(  
 fs.*readFileSync*(path.resolve(storagePath, file), "utf-8")  
 ) as T;  
}  
  
*// Коллекция объектов модели*export class ModelCollection<T extends Indexed> {  
 readonly list: T[] = [];  
 private id = 1;  
  
 add(obj: T): number {  
 this.list.push({ ...obj, id: this.id });  
 return this.id++;  
 }  
  
 find(id: number): T | null {  
 return this.list.find((obj) => obj.id == id) ?? null;  
 }  
  
 remove(id: number) {  
 const i = this.list.findIndex((e) => e.id === id);  
 if (i !== -1) this.list.splice(i, 1);  
 }  
  
 getIds(): number[] {  
 return this.list.map((obj) => obj.id);  
 }  
}  
  
export \* from "./NetworkModel";

***Файл server/websocket/SocketManager.ts***

import io from "socket.io";  
import cookieParser from "cookie-parser";  
import { passport } from "../auth";  
import { InterServerEvents, SocketData, wrapSocketMW } from "./index";  
import {  
 ClientToServerEvents,  
 ServerToClientEvents,  
 UserData,  
} from "../../api";  
  
*// Класс управления web-сокетом*export class SocketManager {  
 *// Пул сокетов для пользователей по их id* private pool: Map<number, io.Socket> = new Map();  
  
 constructor(  
 public readonly server: io.Server<  
 ServerToClientEvents,  
 ClientToServerEvents,  
 InterServerEvents,  
 SocketData  
 >,  
 secretKey: string  
 ) {  
 this.server  
 .use(wrapSocketMW(cookieParser(secretKey)))  
 .use(wrapSocketMW(passport.initialize()))  
 *// Проверка токена пользователя* .use((socket, next) =>  
 wrapSocketMW(  
 passport.authenticate("jwt", (err, user, info) => {  
 if (user) {  
 socket.data.user = user;  
 next();  
 }  
 })  
 )(socket, next)  
 )  
 *// Обработка подключения, добавление сокета в пул* .on("connection", (socket) => {  
 if (socket.data.user) {  
 this.pool.set(socket.data.user.id, socket);  
 socket.on("disconnect", (reason) => {  
 this.pool.delete(socket.data.user!.id);  
 });  
 }  
 });  
 }  
  
 *// Оповещение друзьям пользователя о новости* post(user: UserData) {  
 user.friends.forEach((f) => {  
 this.pool.get(f)?.emit("post", user);  
 });  
 }  
}

***Файл server/websocket/index.ts***

import { UserData } from "../../api";  
import { EventsMap } from "socket.io/dist/typed-events";  
import { Handler, NextFunction, Request, Response } from "express";  
import io from "socket.io";  
import { ExtendedError } from "socket.io/dist/namespace";  
  
export type InterServerEvents = EventsMap;  
  
*// Данные в соединении вебсокета*export interface SocketData {  
 user: UserData;  
}  
  
*// Декоратор express-обработчиков для работы с socket.io*export function wrapSocketMW(mw: Handler) {  
 return (  
 socket: io.Socket,  
 next: (err?: ExtendedError | undefined) => void  
 ) => {  
 mw(socket.request as Request, {} as Response, next as NextFunction);  
 };  
}  
  
export \* from "./SocketManager";

***Файл server/auth.ts***

import passport from "passport";  
import { JwtFromRequestFunction, Strategy as JwtStrategy } from "passport-jwt";  
import { Router } from "express";  
import jwt from "jsonwebtoken";  
import { Indexed, Status, UserAuthData, UserSignUpData } from "../api";  
import { NetworkModel } from "./model";  
  
*// Закрытый ключ*export const secretKey = "#S3cR3t\_K3y#";  
  
*// Извлечение токена из куки*const cookieExtractor: JwtFromRequestFunction = function (req): string {  
 return req.signedCookies?.token ?? null;  
};  
  
export { passport };  
  
export function createAuthRouter(model: NetworkModel): Router {  
 const authRouter = Router();  
  
 *// Проверка токена пользователя* passport.use(  
 new JwtStrategy(  
 {  
 jwtFromRequest: cookieExtractor,  
 secretOrKey: secretKey,  
 },  
 (payload: Indexed, done) => {  
 const data = model.getUserData(payload.id);  
 if (!data || data?.status === Status.BLOCKED) done(null, false);  
 else done(null, data);  
 }  
 )  
 );  
  
 *// Регистрация пользователя* authRouter.post("/signup", (req, res) => {  
 const user = req.body as UserSignUpData;  
 const id = model.signUp(user);  
 res.status(id ? 200 : 401).end();  
 });  
  
 *// Вход по email и паролю, устанавливает токен в куки* authRouter.post("/login", (req, res) => {  
 const data = req.body as UserAuthData,  
 user = model.authorize(data);  
 if (user) {  
 if (user.status === Status.BLOCKED) res.sendStatus(403);  
 else {  
 const token = jwt.sign({ id: user.id }, secretKey, {  
 expiresIn: "1h",  
 });  
 res.cookie("token", token, {  
 maxAge: 60 \* 60 \* 1000,  
 httpOnly: true,  
 signed: true,  
 });  
  
 res.sendStatus(200);  
 }  
 } else res.sendStatus(401);  
 });  
  
 *// Точка проверки действительности токена* authRouter.get(  
 "/login",  
 passport.authenticate("jwt", { session: false }),  
 (req, res) => {  
 res.sendStatus(200);  
 }  
 );  
  
 return authRouter;  
}

***Файл server/router.ts***

import { Router } from "express";  
import path from "path";  
import { NetworkModel } from "./model";  
import {  
 Indexed,  
 PersonalData,  
 Post,  
 Role,  
 Status,  
 StatusData,  
 UserStatusData,  
} from "../api";  
import { Multer } from "multer";  
import { SocketManager } from "./websocket";  
  
export function createNetworkRouter(  
 model: NetworkModel,  
 sm: SocketManager,  
 storagePath: string,  
 uploader: Multer  
): Router {  
 const router = Router();  
 const blockedImg = "blocked.jpg";  
  
 *// Проверка объекта пользователя в res.locals.user* router.use((req, res, next) => {  
 if (!res.locals.user) res.status(500).end("User authentication error");  
 else if (res.locals.user.status === Status.BLOCKED)  
 res.status(403).end("Blocked user");  
 else next();  
 });  
  
 *// Получение собственного id* router.get("/self", (req, res) => {  
 res.json({ id: res.locals.user.id } as Indexed);  
 });  
  
 *// Список пользователей* router.get("/users", (req, res) => {  
 res.json(model.getUsers());  
 });  
  
 *// Middleware для извлечения id пользователя в маршрутах /user/id/\** router.use("/user/:id", (req, res, next) => {  
 res.locals.id = parseInt(req.params.id);  
 next();  
 });  
  
 *// Маршруты ресурсов пользователя* router.use(  
 "/user/:id",  
 Router()  
 .get("/photos", (req, res) => {  
 res.json(model.getPhotos(res.locals.id));  
 })  
 .get("/posts", (req, res) => {  
 res.json(model.getPosts(res.locals.id));  
 })  
 .get("/friends", (req, res) => {  
 res.json(model.getFriends(res.locals.id));  
 })  
 .get("/friendsposts", (req, res) => {  
 res.json(model.getFriendsPosts(res.locals.id));  
 })  
 );  
  
 *// Загрузить аватарку* router.put("/avatar", uploader.single("avatar"), (req, res) => {  
 if (req.file) {  
 model.updateAvatar(res.locals.user.id, req.file.filename);  
 res.sendStatus(200);  
 } else res.sendStatus(204);  
 });  
  
 *// Удаление фотографии профиля* router.delete("/avatar", (req, res) => {  
 model.deleteAvatar(res.locals.user.id);  
 res.sendStatus(200);  
 });  
  
 router  
 .route("/friend/:id")  
 .all((req, res, next) => {  
 res.locals.id = parseInt(req.params.id);  
 if (res.locals.id) next();  
 })  
 *// Отправить/принять заявку* .post((req, res) => {  
 model.friendRequest(res.locals.user.id, res.locals.id);  
 res.end();  
 })  
 *// Отменить/отклонить заявку / удалить друга* .delete((req, res) => {  
 model.friendDecline(res.locals.user.id, res.locals.id);  
 res.end();  
 });  
  
 *// Опубликовать запись* router.post("/post", uploader.array("photos"), (req, res) => {  
 const post = req.body as Post;  
 sm.post(res.locals.user);  
 res.json({  
 id: model.addPost(  
 res.locals.user.id,  
 post.text,  
 ((req.files ?? []) as Express.Multer.File[]).map((f) =>  
 model.addPhoto(res.locals.user.id, f.filename)  
 )  
 ),  
 } as Indexed);  
 });  
  
 *// Обращения к посту* router  
 .route("/post/:postid")  
 .all((req, res, next) => {  
 res.locals.id = parseInt(req.params.postid);  
 next();  
 })  
 .get((req, res) => {  
 const post = model.getPost(res.locals.id);  
 if (post) {  
 if (  
 post.status !== Status.BLOCKED ||  
 res.locals.user.role === Role.ADMIN  
 )  
 res.json(post);  
 else  
 res.json({  
 ...post,  
 text: "<Ресурс заблокирован администрацией>",  
 } as Post);  
 } else res.status(204).json(null);  
 })  
 *// Обновление статуса поста* .patch((req, res) => {  
 if (res.locals.user.role === Role.ADMIN) {  
 model.updatePostStatus(res.locals.id, req.body as StatusData);  
 res.sendStatus(200);  
 } else res.sendStatus(401);  
 });  
  
 *// Обращения к изображению* router  
 .route("/photo/:photoid")  
 .all((req, res, next) => {  
 res.locals.id = parseInt(req.params.photoid);  
 next();  
 })  
 .get((req, res) => {  
 const photo = model.getPhoto(res.locals.id);  
 if (photo) {  
 if (  
 photo.status !== Status.BLOCKED ||  
 res.locals.user.role === Role.ADMIN  
 )  
 res.download(path.resolve(storagePath, photo.file));  
 else res.download(path.resolve(storagePath, blockedImg));  
 } else res.sendStatus(204);  
 });  
  
 *// Обращение к информации фото* router  
 .route("/photoinfo/:photoid")  
 .all((req, res, next) => {  
 res.locals.id = parseInt(req.params.photoid);  
 next();  
 })  
 .get((req, res) => {  
 const photoid = parseInt(req.params.photoid);  
 res.json(model.getPhoto(photoid));  
 })  
 *// Обновление статуса изображения* .patch((req, res) => {  
 if (res.locals.user.role === Role.ADMIN) {  
 model.updatePhotoStatus(res.locals.id, req.body as StatusData);  
 res.sendStatus(200);  
 } else res.sendStatus(401);  
 });  
  
 *// Маршруты данных пользователя* router  
 .route("/user/:id")  
 .get((req, res) => {  
 res.json(model.getUserData(res.locals.id));  
 })  
 .put((req, res) => {  
 if (  
 res.locals.user.role === Role.ADMIN ||  
 res.locals.user.id == res.locals.id  
 ) {  
 const data = req.body as PersonalData;  
 model.updatePersonal(res.locals.id, data);  
 res.sendStatus(200);  
 } else res.sendStatus(401);  
 })  
 .patch((req, res) => {  
 if (res.locals.user.role === Role.ADMIN) {  
 const data = req.body as UserStatusData;  
 model.updateUserStatus(res.locals.id, data);  
 res.sendStatus(200);  
 } else res.sendStatus(401);  
 });  
  
 return router;  
}

***Файл server/index.ts***

import express from "express";  
import cors from "cors";  
import { config } from "../api";  
import { createAuthRouter, passport, secretKey } from "./auth";  
import cookieParser from "cookie-parser";  
import { createNetworkRouter } from "./router";  
import { NetworkModel, load } from "./model";  
import multer from "multer";  
import io from "socket.io";  
import { SocketManager } from "./websocket";  
  
const app = express();  
const port = 3000;  
  
const storagePath = "./storage";  
export const uploader = multer({ dest: storagePath });  
  
const model = new NetworkModel(  
 load("users.json", storagePath),  
 load("photos.json", storagePath),  
 load("posts.json", storagePath),  
 storagePath  
);  
  
const httpServer = app.listen(port, config.serverHost, function () {  
 console.log(`Server started at http://${config.serverHost}:${port}`);  
});  
  
const socket = new SocketManager(  
 new io.Server(httpServer, {  
 cors: { origin: config.client, credentials: true },  
 }),  
 secretKey  
);  
  
app.use(  
 cors({  
 origin: config.client,  
 credentials: true,  
 })  
);  
app.use(cookieParser(secretKey));  
app.use(express.json());  
app.use(express.urlencoded({ extended: true }));  
app.use(passport.initialize());  
app.use("/api", createAuthRouter(model));  
*// Подключение маршрутизатора соцсети и передача всем его обработчикам объекта авторизованного пользователя*app.use(  
 "/api",  
 (req, res, next) =>  
 passport.authenticate("jwt", { session: false }, (err, user, info) => {  
 res.locals.user = user;  
 next();  
 })(req, res, next),  
 createNetworkRouter(model, socket, storagePath, uploader)  
);

***Файл src/rollbar/rollbar.d.ts***

import Rollbar from "rollbar";  
  
declare module "vue/types/vue" {  
 interface Vue {  
 $rollbar: Rollbar;  
 }  
}

***Файл src/rollbar/index.ts***

import \_Vue, { PluginObject } from "vue";  
import Rollbar from "rollbar";  
  
*// Плагин роллбара для использования во vue*export class RollbarPlugin implements PluginObject<never> {  
 constructor(readonly options: Rollbar.Configuration) {}  
 install(Vue: typeof \_Vue): void {  
 Vue.prototype.$rollbar = new Rollbar(this.options);  
 }  
}

***Файл src/router/index.ts***

import Vue from "vue";  
import VueRouter, { RouteConfig } from "vue-router";  
import axios from "axios";  
import { config } from "@/../api";  
  
Vue.use(VueRouter);  
  
export enum Views {  
 *LOGIN* = "login",  
  
 *PROFILE* = "profile",  
 *NEWS* = "news",  
 *FRIENDS* = "friends",  
 *PHOTOS* = "photos",  
  
 *USER\_POSTS* = "user\_posts",  
 *USER\_PHOTOS* = "user\_photos",  
 *USER\_FRIENDS* = "user\_friends",  
 *USER\_FRIENDSPOSTS* = "user\_friends\_posts",  
  
 *USERS\_LIST* = "users\_list",  
}  
  
const routes: Array<RouteConfig> = [  
 {  
 path: "/login",  
 name: Views.*LOGIN*,  
 component: () => import("../views/LoginView.vue"),  
 },  
 {  
 path: "/",  
 component: () => import("../views/Network.vue"),  
 children: [  
 {  
 path: "",  
 name: Views.*PROFILE*,  
 component: () => import("../views/MyProfile.vue"),  
 },  
 {  
 path: "news",  
 name: Views.*NEWS*,  
 component: () => import("../views/News.vue"),  
 },  
 {  
 path: "friends",  
 name: Views.*FRIENDS*,  
 component: () => import("../views/Friends.vue"),  
 },  
 {  
 path: "photos",  
 name: Views.*PHOTOS*,  
 component: () => import("../views/Photos.vue"),  
 },  
 {  
 path: "user/:user\_id",  
 component: () => import("../views/Profile.vue"),  
 children: [  
 {  
 name: Views.*USER\_POSTS*,  
 path: "",  
 component: () => import("../components/lists/PostsList.vue"),  
 },  
 {  
 name: Views.*USER\_PHOTOS*,  
 path: "photos",  
 component: () => import("../components/lists/PhotosList.vue"),  
 },  
 {  
 name: Views.*USER\_FRIENDS*,  
 path: "friends",  
 component: () => import("../components/lists/UsersList.vue"),  
 },  
 {  
 name: Views.*USER\_FRIENDSPOSTS*,  
 path: "friends\_posts",  
 component: () => import("../components/lists/PostsList.vue"),  
 },  
 ],  
 },  
 {  
 path: "users",  
 name: Views.*USERS\_LIST*,  
 component: () => import("../views/UsersListView.vue"),  
 },  
 ],  
 },  
];  
  
const router = new VueRouter({  
 routes,  
});  
  
*// Проверка авторизации пользователя*async function isAuthenticated(): Promise<boolean> {  
 try {  
 await axios.head(  
 new URL(config.endpoints.login, config.server).toString(),  
 {  
 withCredentials: true,  
 }  
 );  
 return true;  
 } catch (err) {  
 return false;  
 }  
}  
  
*// Перенаправление неавторизованного пользователя на страницу входа*router.beforeEach(async (to, from, next) => {  
 const auth = await isAuthenticated();  
 if (auth) {  
 if (to.name === Views.*LOGIN*) next({ name: Views.*PROFILE* });  
 else next();  
 } else {  
 if (to.name !== Views.*LOGIN*) next({ name: Views.*LOGIN* });  
 else next();  
 }  
});  
  
export default router;

***Файл src/styles.main.scss***

$primary: #7749f8;  
@import "bootstrap/scss/bootstrap";

***Файл src/util/loaders/base.ts***

*// Интерфейс загрузчика*export interface ILoader {  
 get url(): URL;  
 fetch(): void;  
}  
  
*// Интерфейс загрузчика индексируемых данных*export interface IndexedLoader {  
 readonly id: number;  
 get endpoint(): string;  
}

***Файл src/util/loaders/listLoaders.ts***

import { config } from "../../../api";  
import axios from "axios";  
import { ILoader, IndexedLoader } from "./base";  
  
*// Абстрактный загрузчик списка*export abstract class ListLoader implements ILoader {  
 readonly list: number[] = [];  
  
 abstract get url(): URL;  
  
 async fetch() {  
 if (this.url)  
 this.list.splice(  
 0,  
 this.list.length,  
 ...(await axios.get<number[]>(this.url.toString())).data  
 );  
 }  
}  
  
*// Загрузчик списка пользователей*export class UsersListLoader extends ListLoader {  
 get url(): URL {  
 return new URL(config.endpoints.usersList, config.server);  
 }  
}  
  
*// Абстрактный загрузчик списков данных пользователя*abstract class UserDataListLoader extends ListLoader implements IndexedLoader {  
 readonly id: number;  
  
 constructor(id: number) {  
 super();  
 this.id = id;  
 }  
  
 abstract get endpoint(): string;  
  
 get url(): URL {  
 return new URL(  
 config.endpoints.user + `${this.id}/` + this.endpoint,  
 config.server  
 );  
 }  
}  
  
*// Загрузчик списка постов пользователя*export class PostsLoader extends UserDataListLoader {  
 get endpoint(): string {  
 return config.endpoints.postsList;  
 }  
}  
  
*// Загрузчик списка фотографий пользователя*export class PhotosLoader extends UserDataListLoader {  
 get endpoint(): string {  
 return config.endpoints.photosList;  
 }  
}  
  
*// Загрузчик списка друзей пользователя*export class FriendsLoader extends UserDataListLoader {  
 get endpoint(): string {  
 return config.endpoints.friends;  
 }  
}  
  
*// Загрузчик постов друзей пользователя*export class FriendsPostsLoader extends PostsLoader {  
 get endpoint(): string {  
 return config.endpoints.friendsPosts;  
 }  
}

***Файл src/util/loaders/dataLodaers.ts***

import axios from "axios";  
import { ILoader, IndexedLoader } from "./base";  
import {  
 config,  
 Indexed,  
 PersonalData,  
 Photo,  
 Post,  
 Resource,  
 Status,  
 StatusData,  
 UserData,  
 UserStatusData,  
} from "../../../api";  
  
*// Абстрактный загрузчик объектов*export abstract class ObjectLoader<T> implements ILoader, IndexedLoader {  
 id: number;  
 data: T | null = null;  
  
 constructor(id: number) {  
 this.id = id;  
 }  
  
 abstract get endpoint(): string;  
  
 get url(): URL {  
 return new URL(this.endpoint + `${this.id}/`, config.server);  
 }  
  
 async fetch() {  
 this.data = (await axios.get<T>(this.url.toString())).data;  
 }  
}  
  
*// Загрузчик пользователя*export class UserLoader extends ObjectLoader<UserData> {  
 get endpoint(): string {  
 return config.endpoints.user;  
 }  
  
 get avatarUrl(): string {  
 return new URL(  
 config.endpoints.photo + `${this.data?.photoId}`,  
 config.server  
 ).toString();  
 }  
  
 get fullName() {  
 return this.data  
 ? `${this.data.surname} ${this.data.name} ${this.data.lastName ?? ""}`  
 : "";  
 }  
  
 updateStatus(sd: UserStatusData) {  
 if (this.data) [this.data.status, this.data.role] = [sd.status, sd.role];  
 return axios.patch<UserStatusData>(this.url.toString(), {  
 status: this.data?.status,  
 role: this.data?.role,  
 } as UserStatusData);  
 }  
  
 updatePersonal(pd: PersonalData) {  
 if (this.data)  
 [  
 this.data.name,  
 this.data.surname,  
 this.data.lastName,  
 this.data.email,  
 this.data.birthDate,  
 ] = [pd.name, pd.surname, pd.lastName, pd.email, pd.birthDate];  
 return axios.put<PersonalData>(this.url.toString(), {  
 name: this.data?.name,  
 surname: this.data?.surname,  
 lastName: this.data?.lastName,  
 birthDate: this.data?.birthDate,  
 email: this.data?.email,  
 } as PersonalData);  
 }  
}  
  
abstract class ResourceLoader<T extends Resource> extends ObjectLoader<T> {}  
  
*// Загрузчик ресурсов*abstract class PublicResourceLoader<  
 T extends Resource & StatusData  
> extends ResourceLoader<T> {  
 updateStatus(s: Status) {  
 if (this.data) this.data.status = s;  
 return axios.patch<StatusData>(this.url.toString(), {  
 status: s,  
 } as StatusData);  
 }  
}  
  
*// Загрузчик поста*export class PostLoader extends PublicResourceLoader<Post> {  
 get endpoint(): string {  
 return config.endpoints.post;  
 }  
  
 *// Публикация новости* static *postUrl* = new URL(config.endpoints.post, config.server);  
 static async *makePost*(text: string, files: FileList | null): Promise<number> {  
 const data = new *FormData*();  
 data.set("text", text);  
 if (files)  
 for (let f = 0; f < files.length; f++)  
 data.append("photos", files.item(f)!);  
 return (await axios.post<Indexed>(this.*postUrl*.toString(), data)).data.id;  
 }  
}  
  
*// Загрузчик фотографии (достаточно использовать сгенерированный url)*export class PhotoLoader extends ObjectLoader<Blob> {  
 get endpoint(): string {  
 return config.endpoints.photo;  
 }  
}  
  
*// Загрузчик информации о фотографии*export class PhotoInfoLoader extends PublicResourceLoader<Photo> {  
 get endpoint(): string {  
 return config.endpoints.photoInfo;  
 }  
}

***Файл src/util/loaders/index.ts***

export \* from "./listLoaders";  
export \* from "./dataLoaders";

***Файл src/util/SocketManager.ts***

import { io, Socket } from "socket.io-client";  
import {  
 ClientToServerEvents,  
 config,  
 ServerToClientEvents,  
 UserData,  
} from "../../api";  
  
export class SocketManager {  
 private socket: Socket<ServerToClientEvents, ClientToServerEvents>;  
  
 constructor() {  
 this.socket = io(config.webSocket, { withCredentials: true });  
 }  
  
 addListener(cb: (user: UserData) => void) {  
 this.socket.on("post", cb);  
 }  
  
 removeListener(cb: (user: UserData) => void) {  
 this.socket.removeListener("post", cb);  
 }  
}

***Файл src/util/UserController.ts***

import { config, Indexed, Role, UserAuthData, UserSignUpData } from "../../api";  
import axios from "axios";  
import { UserLoader } from "@/util/loaders/dataLoaders";  
  
export class UserController {  
 readonly self = new URL(config.endpoints.self, config.server);  
 id: number | null = null;  
 loader: UserLoader | null = null;  
  
 *// Загрузка собственного id* async fetchId() {  
 this.id = (await axios.get<Indexed>(this.self.toString())).data.id;  
 this.loader = new UserLoader(this.id);  
 }  
  
 *// Проверка прав администратора* get isAdmin(): boolean {  
 return this.loader?.data?.role === Role.ADMIN;  
 }  
  
 *// Регистрация* static readonly signUpUrl = new URL(config.endpoints.signUp, config.server);  
 static async signUp(user: UserSignUpData): Promise<boolean> {  
 try {  
 await axios.post<Indexed>(this.signUpUrl.toString(), user);  
 return true;  
 } catch (err) {  
 return false;  
 }  
 }  
  
 *// Вход* static readonly loginUrl = new URL(config.endpoints.login, config.server);  
 static async login(user: UserAuthData): Promise<boolean> {  
 try {  
 await axios.post<Indexed>(this.loginUrl.toString(), user);  
 return true;  
 } catch (err) {  
 return false;  
 }  
 }  
  
 readonly avatarUrl = new URL(config.endpoints.avatar, config.server);  
 async updateAvatar(file: File) {  
 const fd = new FormData();  
 fd.append("avatar", file);  
 await axios.put(this.avatarUrl.toString(), fd);  
 }  
  
 async deleteAvatar() {  
 await axios.delete(this.avatarUrl.toString());  
 }  
  
 getFriendUrl(id: number): URL {  
 return new URL(config.endpoints.friend + id, config.server);  
 }  
  
 async requestFriend(loader: UserLoader) {  
 await axios.post(this.getFriendUrl(loader.id).toString());  
 await this.loader?.fetch();  
 await loader.fetch();  
 }  
  
 async declineFriend(loader: UserLoader) {  
 await axios.delete(this.getFriendUrl(loader.id).toString());  
 await this.loader?.fetch();  
 await loader.fetch();  
 }  
}

***Файл src/util/index.ts***

export \* from "./loaders";  
export \* from "./UserController";  
export \* from "./SocketManager";

***Файл src/icons.ts***

export {  
 faWrench,  
 faUserGear,  
 faNewspaper,  
 faImages,  
 faUserGroup,  
 faRadio,  
 faCircleUser,  
 faCircleArrowLeft,  
 faBan,  
 faClockRotateLeft,  
 faUsersRays,  
} from "@fortawesome/free-solid-svg-icons";

***Файл src/main.ts***

import Vue from "vue";  
import App from "./App.vue";  
import router from "./router";  
import axios from "axios";  
import \* as icons from "./icons";  
import "bootstrap";  
import { library } from "@fortawesome/fontawesome-svg-core";  
import { FontAwesomeIcon } from "@fortawesome/vue-fontawesome";  
import { RollbarPlugin } from "@/rollbar";  
  
library.add(icons);  
Vue.component("font-awesome-icon", FontAwesomeIcon);  
  
Vue.use(  
 new RollbarPlugin({  
 accessToken: "482cd9074ca34bb496d382ce456c28e5",  
 captureUncaught: true,  
 captureUnhandledRejections: true,  
 captureIp: true,  
 })  
);  
  
*// Логирование ошибок через rollbar*Vue.config.errorHandler = async (err, vm, info) => {  
 vm.$rollbar.log("Error Handled:");  
 vm.$rollbar.error(err);  
 throw err;  
};  
  
Vue.config.productionTip = false;  
  
new Vue({  
 router,  
 render: (h) => h(App),  
}).$mount("#app");  
  
*// Добавить куки ко всем запросам*axios.interceptors.request.use(function (config) {  
 config.withCredentials = true;  
 return config;  
});

***Файл src/App.vue***

<template>  
 <div id="app" class="min-vh-100">  
 <router-view />  
  
 <Toaster class="position-fixed bottom-0 end-0 p-3" ref="toaster" />  
 <SignUpModal ref="signUp" />  
 </div>  
</template>  
  
<script lang="ts">  
import { *Component*, *ProvideReactive*, Vue } from "vue-property-decorator";  
import { Views } from "./router";  
import Toaster from "@/components/Toaster.vue";  
import SignUpModal from "@/components/forms/SignUpModal.vue";  
  
@Component({  
 components: {  
 SignUpModal,  
 Toaster,  
 },  
})  
export default class App extends Vue {  
 private Views = Views;  
  
 $refs!: {  
 toaster: Toaster;  
 signUp: SignUpModal;  
 };  
  
 private mounted() {  
 this.toaster = this.$refs.toaster;  
 this.signUp = this.$refs.signUp;  
 }  
  
 @ProvideReactive() toaster: Toaster | null = null;  
 @ProvideReactive() signUp: SignUpModal | null = null;  
}  
</script>  
  
<style lang="scss">  
@import "styles/main";  
</style>

***Файл src/components/forms/PersonalDataRedactor.vue***

<template>  
 <div v-if="pd">  
 <div class="mb-3">  
 <label>Фамилия</label>  
 <input  
 required  
 class="form-control"  
 type="text"  
 v-model="pd.surname"  
 placeholder="Иванов"  
 />  
 </div>  
  
 <div class="mb-3">  
 <label>Имя</label>  
 <input  
 required  
 class="form-control"  
 type="text"  
 v-model="pd.name"  
 placeholder="Иван"  
 />  
 </div>  
  
 <div class="mb-3">  
 <label>Отчество</label>  
 <input  
 class="form-control"  
 type="text"  
 v-model="pd.lastName"  
 placeholder="Иванович"  
 />  
 </div>  
  
 <div class="mb-3">  
 <label>Дата рождения</label>  
 <input  
 required  
 class="form-control"  
 type="date"  
 v-model="pd.birthDate"  
 placeholder="1970-01-01"  
 />  
 </div>  
  
 <div class="mb-3">  
 <label>Адрес электронной почты</label>  
 <input  
 required  
 class="form-control"  
 type="email"  
 v-model="pd.email"  
 placeholder="example@mail.com"  
 />  
 </div>  
 </div>  
</template>  
  
<script lang="ts">  
import { Component, Model, Vue } from "vue-property-decorator";  
import { PersonalData } from "../../../api";  
  
*// Поля редактирования персональной информации*@Component  
export default class PersonalDataRedactor extends Vue {  
 @Model("data-changed", { required: true }) pd!: PersonalData;  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/components/forms/PostModal.vue***

<template>  
 <form  
 class="modal fade"  
 data-bs-backdrop="static"  
 ref="form"  
 @submit.prevent="submit"  
 >  
 <div class="modal-dialog modal-dialog-centered modal-lg">  
 <div class="modal-content">  
 <h3 class="modal-header">Опубликовать запись</h3>  
 <div class="modal-body">  
 <div class="mb-3">  
 <textarea class="form-control" v-model="text" rows="5" />  
 </div>  
  
 <div class="mb-3">  
 <label>Прикрепить файлы</label>  
 <input class="form-control" type="file" ref="files" multiple />  
 </div>  
 </div>  
 <div class="modal-footer">  
 <button type="button" class="btn btn-secondary" @click="modal.hide()">  
 Отмена  
 </button>  
 <button  
 type="button"  
 class="btn btn-info"  
 @click="$refs.form.requestSubmit()"  
 >  
 Опубликовать  
 </button>  
 </div>  
 </div>  
 </div>  
 </form>  
</template>  
  
<script lang="ts">  
import { *Component*, Vue } from "vue-property-decorator";  
import { Modal } from "bootstrap";  
import { PostLoader, PostsLoader } from "@/util";  
  
*// Всплывающее окно добавления новости*@Component  
export default class PostModal extends Vue {  
 private modal!: Modal;  
 private text = "";  
 private loader: PostsLoader | null = null;  
  
 $refs!: {  
 form: HTMLFormElement;  
 files: HTMLInputElement;  
 };  
  
 private mounted() {  
 this.modal = new Modal(this.$refs.form);  
 }  
  
 show(loader: PostsLoader) {  
 this.loader = loader;  
 this.text = "";  
 if (this.$refs.files) this.$refs.files.value = "";  
 this.modal.show();  
 }  
  
 private async submit() {  
 this.$refs.files.files;  
 if (this.loader) {  
 let loader = this.loader;  
 this.loader = null;  
 loader.list.splice(  
 0,  
 0,  
 await PostLoader.makePost(this.text, this.$refs.files.files)  
 );  
 }  
 this.loader = null;  
 this.modal.hide();  
 }  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/components/forms/RedactModal.vue***

<template>  
 <form  
 class="modal fade"  
 data-bs-backdrop="static"  
 ref="form"  
 @submit.prevent="submit"  
 >  
 <div class="modal-dialog modal-dialog-centered modal-md">  
 <div class="modal-content">  
 <h3 class="modal-header">Редактировать Пользователя</h3>  
 <div class="modal-body" v-if="buffer">  
 <PersonalDataRedactor v-model="buffer" />  
 <StatusDataRedactor v-if="admin" v-model="buffer" />  
 </div>  
  
 <div class="modal-footer">  
 <button type="button" class="btn btn-secondary" @click="modal.hide()">  
 Отмена  
 </button>  
 <button  
 type="button"  
 class="btn btn-info"  
 @click="$refs.form.requestSubmit()"  
 >  
 Обновить  
 </button>  
 </div>  
 </div>  
 </div>  
 </form>  
</template>  
  
<script lang="ts">  
import { Component, Vue } from "vue-property-decorator";  
import { Modal } from "bootstrap";  
import { UserLoader } from "@/util";  
import { UserData } from "../../../api";  
import PersonalDataRedactor from "@/components/forms/PersonalDataRedactor.vue";  
import StatusDataRedactor from "@/components/forms/StatusDataRedactor.vue";  
  
*// Всплывающее окно редактирования пользователя*@Component({  
 components: { StatusDataRedactor, PersonalDataRedactor },  
})  
export default class RedactModal extends Vue {  
 private modal!: Modal;  
 private loader: UserLoader | null = null;  
 private buffer: UserData | null = null;  
 private admin = false;  
  
 $refs!: {  
 form: HTMLFormElement;  
 };  
  
 private mounted() {  
 this.modal = new Modal(this.$refs.form);  
 }  
  
 show(loader: UserLoader, admin = false) {  
 this.admin = admin;  
 this.loader = loader;  
 if (loader.data) {  
 this.buffer = { ...loader.data };  
 }  
 this.modal.show();  
 }  
  
 private submit() {  
 this.modal.hide();  
 if (this.loader?.data && this.buffer) {  
 this.loader.updatePersonal(this.buffer);  
 if (this.admin) this.loader.updateStatus(this.buffer);  
 this.loader = null;  
 }  
 }  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/components/forms/SignUpModal.vue***

<template>  
 <form  
 class="modal fade"  
 data-bs-backdrop="static"  
 ref="form"  
 @submit.prevent="submit"  
 >  
 <div class="modal-dialog modal-dialog-centered modal-md">  
 <div class="modal-content">  
 <h3 class="modal-header">Регистрация</h3>  
 <div class="modal-body" v-if="usud">  
 <PersonalDataRedactor v-model="usud" />  
  
 <div class="mb-3">  
 <label>Пароль</label>  
 <input  
 required  
 class="form-control"  
 type="password"  
 v-model="usud.password"  
 placeholder="Пароль"  
 />  
 </div>  
 </div>  
  
 <div class="modal-footer">  
 <button type="button" class="btn btn-secondary" @click="modal.hide()">  
 Отмена  
 </button>  
 <button  
 type="button"  
 class="btn btn-warning"  
 @click="$refs.form.requestSubmit()"  
 >  
 Подтвердить  
 </button>  
 </div>  
 </div>  
 </div>  
 </form>  
</template>  
  
<script lang="ts">  
import { Component, InjectReactive, Vue } from "vue-property-decorator";  
import { Modal } from "bootstrap";  
import { UserController } from "@/util";  
import { UserSignUpData } from "../../../api";  
import PersonalDataRedactor from "@/components/forms/PersonalDataRedactor.vue";  
import StatusDataRedactor from "@/components/forms/StatusDataRedactor.vue";  
import Toaster, { States } from "@/components/Toaster.vue";  
  
*// Всплывающее окно регистрации*@Component({  
 components: { StatusDataRedactor, PersonalDataRedactor },  
})  
export default class SignUpModal extends Vue {  
 @InjectReactive() readonly toaster!: Toaster | null;  
 private modal!: Modal;  
 private usud: UserSignUpData | null = null;  
  
 $refs!: {  
 form: HTMLFormElement;  
 };  
  
 private mounted() {  
 this.modal = new Modal(this.$refs.form);  
 }  
  
 show() {  
 this.usud = {  
 email: "",  
 name: "",  
 surname: "",  
 lastName: "",  
 birthDate: "",  
 password: "",  
 };  
  
 this.modal.show();  
 }  
  
 private async submit() {  
 if (this.usud)  
 if (await UserController.signUp(this.usud)) {  
 this.toaster?.show("Вы успешно зарегистрированы", States.SUCCESS);  
 this.modal.hide();  
 } else {  
 this.toaster?.show(  
 "Пользователь c таким email-ом уже существует",  
 States.WARNING  
 );  
 }  
 }  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/components/forms/StatusDataRedactor.vue***

<template>  
 <div v-if="sd">  
 <div class="mb-3">  
 <label class="d-block">Роль</label>  
 <div class="btn-group">  
 <input  
 type="radio"  
 class="btn-check"  
 name="role"  
 id="user"  
 v-model="sd.role"  
 :value="Role.USER"  
 />  
 <label class="btn btn-outline-secondary" for="user">Пользователь</label>  
  
 <input  
 type="radio"  
 class="btn-check"  
 name="role"  
 id="admin"  
 v-model="sd.role"  
 :value="Role.ADMIN"  
 />  
 <label class="btn btn-outline-info" for="admin">Администратор</label>  
 </div>  
 </div>  
  
 <div class="mb-3">  
 <label class="d-block">Статус</label>  
 <div class="btn-group">  
 <input  
 type="radio"  
 class="btn-check"  
 name="status"  
 id="active"  
 v-model="sd.status"  
 :value="Status.ACTIVE"  
 />  
 <label class="btn btn-outline-success" for="active">Активный</label>  
  
 <input  
 type="radio"  
 class="btn-check"  
 name="status"  
 id="unconfirmed"  
 v-model="sd.status"  
 :value="Status.UNCONFIRMED"  
 />  
 <label class="btn btn-outline-warning" for="unconfirmed">  
 Не подтверждённый  
 </label>  
  
 <input  
 type="radio"  
 class="btn-check"  
 name="status"  
 id="blocked"  
 v-model="sd.status"  
 :value="Status.BLOCKED"  
 />  
 <label class="btn btn-outline-danger" for="blocked">  
 Заблокированный  
 </label>  
 </div>  
 </div>  
 </div>  
</template>  
  
<script lang="ts">  
import { Component, Model, Vue } from "vue-property-decorator";  
  
import { Role, Status, UserStatusData } from "../../../api";  
  
*// Поля редактирования статусной информации*@Component({})  
export default class StatusDataRedactor extends Vue {  
 private Role = Role;  
 private Status = Status;  
 @Model("status-changed", { required: true }) sd!: UserStatusData;  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/components/forms/UpdateAvatarModal.vue***

<template>  
 <form  
 class="modal fade"  
 data-bs-backdrop="static"  
 ref="form"  
 @submit.prevent="update"  
 >  
 <div class="modal-dialog modal-dialog-centered modal-md">  
 <div class="modal-content">  
 <div class="modal-header">  
 <h4>Обновить фотографию профиля</h4>  
 </div>  
  
 <div class="modal-body">  
 <label for="formFile" class="form-label">Выберите файл</label>  
 <input  
 class="form-control"  
 required  
 type="file"  
 id="formFile"  
 ref="file"  
 />  
 </div>  
 <div class="modal-footer">  
 <button type="button" class="btn btn-secondary" @click="modal.hide()">  
 Отмена  
 </button>  
 <button  
 v-if="user?.loader?.data?.photoId"  
 type="button"  
 class="btn btn-danger"  
 @click="remove"  
 >  
 Удалить фото  
 </button>  
 <button  
 type="button"  
 class="btn btn-warning"  
 @click="$refs.form.requestSubmit()"  
 >  
 Загрузить  
 </button>  
 </div>  
 </div>  
 </div>  
 </form>  
</template>  
  
<script lang="ts">  
import { Component, Vue } from "vue-property-decorator";  
import { Modal } from "bootstrap";  
import { UserController } from "@/util";  
  
*// Всплывающее окно обновление фотографии профиля*@Component  
export default class UpdateAvatarModal extends Vue {  
 private modal!: Modal;  
 private user: UserController | null = null;  
  
 $refs!: {  
 form: HTMLFormElement;  
 file: HTMLInputElement;  
 };  
  
 private mounted() {  
 this.modal = new Modal(this.$refs.form);  
 }  
  
 show(user: UserController) {  
 this.user = user;  
 this.modal.show();  
 }  
  
 private async update() {  
 this.modal.hide();  
 if (this.$refs.file?.files?.length)  
 await this.user?.updateAvatar(this.$refs.file.files[0]);  
 this.$router.go(0);  
 }  
  
 private async remove() {  
 this.modal.hide();  
 await this.user?.deleteAvatar();  
 this.$router.go(0);  
 }  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/components/lists/List.ts***

import Vue from "vue";  
import Component from "vue-class-component";  
import { Prop } from "vue-property-decorator";  
  
*// Миксин списка объектов*@Component  
export default class List extends Vue {  
 @Prop({ required: true }) readonly list!: number[];  
}

***Файл src/components/lists/PhotosList.vue***

<template>  
 <section class="row gap-2 g-0">  
 <div v-for="photo of list" :key="photo" class="image">  
 <Photo :id="photo" class="w-100 h-100"> </Photo>  
 </div>  
 </section>  
</template>  
  
<script lang="ts">  
import { Component } from "vue-property-decorator";  
  
import Photo from "@/components/Photo.vue";  
import List from "@/components/lists/List";  
  
*// Список фотографий пользователя*@Component({  
 components: { Photo },  
})  
export default class PhotosList extends List {}  
</script>  
  
<style scoped lang="scss">  
.photos {  
 display: grid;  
 grid-template-columns: repeat(3, 1fr);  
}  
.image {  
 height: 400px;  
 width: auto;  
}  
</style>

***Файл src/components/lists/PostsList.vue***

<template>  
 <section class="m-auto">  
 <transition-group name="list">  
 <Post v-for="post of list" :key="post" :id="post" class="border mt-3"  
 /></transition-group>  
 </section>  
</template>  
  
<script lang="ts">  
import { Component } from "vue-property-decorator";  
import Post from "@/components/Post.vue";  
import List from "@/components/lists/List";  
  
*// Список постов*@Component({  
 components: { Post },  
})  
export default class PostsList extends List {}  
</script>  
  
<style scoped lang="scss">  
.list-enter-active,  
.list-leave-active {  
 transition: all 0.5s;  
 max-height: 100vh;  
}  
.list-enter,  
.list-leave-to {  
 opacity: 0;  
 max-height: 0;  
}  
</style>

***Файл src/components/lists/UsersList.vue***

<template>  
 <section class="m-auto">  
 <transition-group name="list" tag="div">  
 <div v-for="id of list" :key="id" class="border mt-3">  
 <UserCard :id="id" :configurable="user.isAdmin"></UserCard></div  
 ></transition-group>  
 </section>  
</template>  
  
<script lang="ts">  
import { Component, InjectReactive } from "vue-property-decorator";  
import UserCard from "@/components/UserCard.vue";  
import List from "@/components/lists/List";  
import { UserController } from "@/util";  
  
*// Список пользователей*@Component({  
 components: { UserCard },  
})  
export default class UsersList extends List {  
 @InjectReactive() readonly user!: UserController | null;  
}  
</script>  
  
<style scoped lang="scss">  
.list-enter-active,  
.list-leave-active {  
 transition: all 0.5s;  
 max-height: 30vh;  
}  
.list-enter,  
.list-leave-to {  
 opacity: 0;  
 max-height: 0;  
}  
</style>

***Файл src/components/lists/Carousel.ts***

<template>  
 <section :id="'carousel' + id" class="carousel slide">  
 <div class="carousel-indicators" v-show="list.length > 1">  
 <button  
 v-for="(photo, index) of list"  
 :key="index"  
 type="button"  
 :data-bs-target="'#carousel' + id"  
 :data-bs-slide-to="index"  
 ref="indicators"  
 ></button>  
 </div>  
 <div class="carousel-inner">  
 <div  
 v-for="(photo, index) of list"  
 :key="index"  
 ref="images"  
 class="carousel-item w-100"  
 >  
 <div class="w-100 h-100 d-flex justify-content-center">  
 <Photo :id="photo" class=""></Photo>  
 </div>  
 </div>  
 </div>  
 </section>  
</template>  
  
<script lang="ts">  
import { Component } from "vue-property-decorator";  
import Photo from "@/components/Photo.vue";  
import List from "@/components/lists/List";  
  
*// Карусель фотографий в посте*@Component({  
 components: { Photo },  
})  
export default class Carousel extends List {  
 private id = 0;  
  
 $refs!: {  
 indicators: HTMLButtonElement[];  
 images: HTMLDivElement[];  
 };  
  
 private mounted() {  
 if (this.list.length) {  
 this.id = Date.now();  
 this.$refs.indicators[0].classList.add("active");  
 this.$refs.images[0].classList.add("active");  
 }  
 }  
}  
</script>  
  
<style scoped lang="scss">  
@import "@/styles/main.scss";  
.carousel-item {  
 background: $gray-600;  
 height: 400px;  
 width: auto;  
}  
</style>

***Файл src/components/BanSwitcher.vue***

<template>  
 <div v-if="user.isAdmin">  
 <font-awesome-icon  
 v-if="status === Status.ACTIVE"  
 icon="fa-solid fa-ban"  
 class="text-danger"  
 @click="switchStatus(Status.BLOCKED)"  
 role="button"  
 />  
 <font-awesome-icon  
 v-else  
 icon="fa-solid fa-clock-rotate-left"  
 class="text-success"  
 @click="switchStatus(Status.ACTIVE)"  
 role="button"  
 />  
 </div>  
</template>  
  
<script lang="ts">  
import {  
 Component,  
 Emit,  
 InjectReactive,  
 Prop,  
 Vue,  
} from "vue-property-decorator";  
import { Status } from "@/../api";  
import { UserController } from "@/util";  
  
*// Компонент блокирования/разблокирования ресурсов*@Component({})  
export default class BanSwitcher extends Vue {  
 @InjectReactive() readonly user!: UserController;  
  
 private Status = Status;  
 @Prop({ required: true }) status!: Status;  
  
 @Emit("switched")  
 private switchStatus(s: Status): Status {  
 return s;  
 }  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/components/Photo.vue***

<template>  
 <div class="position-relative" v-if="photo && photoInfo?.data">  
 <img  
 :src="photo.url.toString()"  
 :alt="id"  
 class="h-100 w-100"  
 :class="{  
 banned:  
 photoInfo.data.status === Status.BLOCKED &&  
 user.loader?.data?.role === Role.ADMIN,  
 }"  
 />  
 <BanSwitcher  
 :status="photoInfo.data.status"  
 class="position-absolute fs-2 pt-1 pe-2 icon"  
 @switched="switchStatus"  
 />  
 </div>  
</template>  
  
<script lang="ts">  
import { Component, InjectReactive, Prop, Vue } from "vue-property-decorator";  
import { PhotoInfoLoader, PhotoLoader, UserController } from "@/util";  
import { Status, Role } from "@/../api";  
import BanSwitcher from "@/components/BanSwitcher.vue";  
import Toaster, { States } from "@/components/Toaster.vue";  
  
*// Фотография в галерее/постах*@Component({  
 components: { BanSwitcher },  
})  
export default class Photo extends Vue {  
 private Status = Status;  
 private Role = Role;  
 @InjectReactive() readonly toaster!: Toaster | null;  
 @InjectReactive() readonly user!: UserController;  
 @Prop({ required: true }) readonly id!: number;  
  
 private photo: PhotoLoader | null = null;  
 private photoInfo: PhotoInfoLoader | null = null;  
  
 private async mounted() {  
 this.photo = new PhotoLoader(this.id);  
 this.photoInfo = new PhotoInfoLoader(this.id);  
 await this.photoInfo?.fetch();  
 }  
  
 private switchStatus(s: Status) {  
 this.toaster?.show(  
 s === Status.ACTIVE ? "Разблокировано" : "Заблокировано",  
 s === Status.ACTIVE ? States.SUCCESS : States.DANGER  
 );  
 this.photoInfo?.updateStatus(s);  
 }  
}  
</script>  
  
<style scoped lang="scss">  
.banned {  
 opacity: 0.5;  
}  
  
img {  
 object-fit: contain;  
 transition: all 0.3s ease-in-out;  
}  
  
.icon {  
 top: 0;  
 right: 0;  
}  
</style>

***Файл src/components/Post.vue***

<template>  
 <article  
 v-if="post?.data"  
 class="p-3 post"  
 :class="{  
 banned: post.data.status === Status.BLOCKED,  
 }"  
 >  
 <section v-if="userLoader?.data" class="d-flex">  
 <div  
 class="col-2 col-md-1 d-flex align-items-center justify-content-center card-image"  
 >  
 <UserThumbnail class="w-100 h-100" :loader="userLoader"></UserThumbnail>  
 </div>  
 <div  
 class="col-9 col-md-10 d-flex flex-column justify-content-between ps-3 card-content"  
 >  
 <h6>{{ userLoader.fullName }}</h6>  
 <h6 class="text-muted fw-normal">  
 {{ new Date(post.data.time).toLocaleString() }}  
 </h6>  
 </div>  
 <BanSwitcher  
 :status="post.data.status"  
 class="col-1 text-danger icon fs-2 p-1"  
 @switched="switchStatus"  
 />  
 </section>  
 <section class="mt-2">  
 {{ post.data.text }}  
 </section>  
 <section>  
 <Carousel  
 v-if="post.data?.photosId?.length"  
 :list="post.data.photosId"  
 class="mt-3"  
 ></Carousel>  
 </section>  
 </article>  
</template>  
  
<script lang="ts">  
import { Component, InjectReactive, Prop, Vue } from "vue-property-decorator";  
import { PostLoader, UserController, UserLoader } from "@/util";  
import Carousel from "@/components/lists/Carousel.vue";  
import UserThumbnail from "@/components/UserThumbnail.vue";  
import BanSwitcher from "@/components/BanSwitcher.vue";  
import Toaster, { States } from "@/components/Toaster.vue";  
import { Status, Role } from "@/../api";  
  
*// Компонент поста*@Component({  
 components: { BanSwitcher, UserThumbnail, Carousel },  
})  
export default class Post extends Vue {  
 private Status = Status;  
 private Role = Role;  
 @InjectReactive() readonly user!: UserController;  
 @InjectReactive() readonly toaster!: Toaster | null;  
 @Prop({ type: Number, required: true }) readonly id!: number;  
  
 private post: PostLoader | null = null;  
 private userLoader: UserLoader | null = null;  
  
 private async mounted() {  
 this.post = new PostLoader(this.id);  
 await this.post.fetch();  
 if (this.post.data) this.userLoader = new UserLoader(this.post.data.userId);  
 this.userLoader?.fetch();  
 }  
  
 private switchStatus(s: Status) {  
 this.toaster?.show(  
 s === Status.ACTIVE ? "Разблокировано" : "Заблокировано",  
 s === Status.ACTIVE ? States.SUCCESS : States.DANGER  
 );  
 this.post?.updateStatus(s);  
 }  
}  
</script>  
  
<style scoped lang="scss">  
@import "@/styles/main.scss";  
  
.post {  
 transition: all 0.3s ease-in-out;  
}  
.banned {  
 background-color: $red-100;  
}  
</style>

***Файл src/components/Toaster.uve***

<template>  
 <Transition name="toast">  
 <section class="toast-container text-center" v-if="visible">  
 <div class="toast d-block m-0">  
 <div  
 class="toast-body alert m-0"  
 :class="  
 'alert-' +  
 ['primary', 'secondary', 'success', 'warning', 'danger'][state]  
 "  
 >  
 {{ this.msg }}  
 </div>  
 </div>  
 </section>  
 </Transition>  
</template>  
  
<script lang="ts">  
import { Component, Vue } from "vue-property-decorator";  
import { Status } from "@/../api";  
  
*// Состояние тостера (цвет алёрта)*export enum States {  
 *PRIMARY*,  
 *SECONDARY*,  
 *SUCCESS*,  
 *WARNING*,  
 *DANGER*,  
}  
  
*// Компонент вывода всплывающих сообщений*@Component({  
 components: {},  
})  
export default class Toaster extends Vue {  
 private Status = Status;  
  
 private msg = "";  
 private state: States = States.PRIMARY;  
 private visible = false;  
 private timerId = 0;  
  
 show(  
 msg: string,  
 state: States = States.PRIMARY,  
 delay = 1500,  
 sound?: HTMLAudioElement  
 ) {  
 this.state = state;  
 this.msg = msg;  
 this.visible = true;  
 clearTimeout(this.timerId);  
 this.timerId = setTimeout(() => {  
 this.visible = false;  
 }, delay);  
 if (sound)  
 try {  
 sound.play();  
 } catch (e) {  
 this.$rollbar.error(`Error during playing sound: ${e}`);  
 }  
 }  
}  
</script>  
  
<style scoped lang="scss">  
.toast-container {  
 z-index: 5000;  
}  
  
.toast-enter-active,  
.toast-leave-active,  
.alert {  
 transition: all 0.3s ease-in-out;  
}  
.toast-enter,  
.toast-leave-to {  
 opacity: 0;  
 transform: translateY(100%);  
}  
</style>

***Файл src/components/UserCard.vue***

<template>  
 <article  
 class="d-flex p-3 flex-column flex-md-row align-items-center align-items-md-stretch"  
 v-if="loader?.data"  
 >  
 <div  
 class="col-6 col-sm-5 col-md-3 p-2 d-flex align-items-center justify-content-center"  
 >  
 <UserThumbnail :loader="loader" class="col-12" />  
 </div>  
 <div  
 class="col-12 col-md-9 ps-0 ps-md-3 d-flex flex-column justify-content-between"  
 >  
 <h4 class="row justify-content-between g-0 align-items-start">  
 <span class="col-8">  
 {{ loader.fullName }}  
 </span>  
 <span class="badge bg-secondary rounded-pill col-2">#{{ id }}</span>  
 <font-awesome-icon  
 v-if="configurable"  
 icon="fa-solid fa-user-gear"  
 class="text-primary col-1 fs-3"  
 role="button"  
 @click="settings"  
 />  
 </h4>  
 <h5 class="text-primary">  
 {{ loader.data.email }}  
 </h5>  
 <h6 class="text-muted">  
 Дата рождения:  
 {{ new Date(loader.data.birthDate).toLocaleDateString() }}  
 </h6>  
 <h6>  
 <span  
 class="badge rounded-pill"  
 :class="['bg-secondary', 'bg-info'][loader.data.role]"  
 >  
 {{ ["Пользователь", "Администратор"][loader.data.role] }}  
 </span>  
 </h6>  
 <h6>  
 <span  
 class="badge rounded-pill"  
 :class="['bg-success', 'bg-warning', 'bg-danger'][loader.data.status]"  
 >  
 {{  
 ["Активный", "Не подтверждённый", "Заблокированный"][  
 loader.data.status  
 ]  
 }}  
 </span>  
 </h6>  
 <div  
 v-if="user.loader?.data && loader.data && user.id !== id"  
 class="text-end"  
 >  
 <div  
 v-if="user.loader.data.friendsRequests.includes(id)"  
 class="btn-group"  
 role="group"  
 >  
 <button class="btn btn-primary" @click="requestFriend">  
 Принять заявку  
 </button>  
 <button class="btn btn-warning" @click="declineFriend">  
 Отклонить  
 </button>  
 </div>  
 <div v-else>  
 <button  
 class="btn btn-danger"  
 v-if="user.loader.data.friends.includes(id)"  
 @click="declineFriend"  
 >  
 Удалить из друзей  
 </button>  
  
 <button  
 class="btn btn-secondary"  
 v-else-if="loader.data.friendsRequests.includes(user.id)"  
 @click="declineFriend"  
 >  
 Отменить заявку  
 </button>  
  
 <button v-else class="btn btn-info" @click="requestFriend">  
 Добавить в друзья  
 </button>  
 </div>  
 </div>  
 </div>  
 </article>  
</template>  
  
<script lang="ts">  
import { Component, InjectReactive, Prop, Vue } from "vue-property-decorator";  
import { UserController, UserLoader } from "@/util";  
import { Views } from "@/router";  
import UserThumbnail from "@/components/UserThumbnail.vue";  
import RedactModal from "@/components/forms/RedactModal.vue";  
import { Role } from "../../api";  
  
*// Карточка Пользователя*@Component({  
 components: { UserThumbnail },  
})  
export default class UserCard extends Vue {  
 private Views = Views;  
 @InjectReactive() readonly redact!: RedactModal | null;  
 @InjectReactive() readonly user!: UserController;  
 @Prop({ required: true }) readonly id!: number;  
 @Prop({ default: false }) readonly configurable!: boolean;  
  
 private loader: UserLoader | null = null;  
  
 private async mounted() {  
 if (this.id === this.user?.id) this.loader = this.user.loader;  
 else {  
 this.loader = new UserLoader(this.id);  
 await this.loader.fetch();  
 }  
 }  
  
 private settings() {  
 if (this.loader?.data)  
 this.redact?.show(  
 this.loader,  
 this.user?.loader?.data?.role === Role.ADMIN  
 );  
 }  
  
 private requestFriend() {  
 if (this.loader) this.user?.requestFriend(this.loader);  
 }  
  
 private declineFriend() {  
 if (this.loader) this.user?.declineFriend(this.loader);  
 }  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/components/UserThumbnail.vue***

<template>  
 <router-link  
 :to="{  
 name: Views.USER\_POSTS,  
 params: { user\_id: this.loader.id.toString() },  
 }"  
 class="position-relative card-image"  
 :class="{ 'self-profile': $route.name === Views.PROFILE }"  
 >  
 <font-awesome-icon  
 icon="fa-circle-arrow-left"  
 class="position-absolute text-info open w-100 h-100 bg-white hover"  
 />  
 <font-awesome-icon  
 v-if="!loader.data.photoId"  
 icon="fa-circle-user"  
 class="rounded-circle img-fluid text-primary"  
 />  
 <img v-else :src="loader.avatarUrl" class="rounded-circle img-fluid" />  
 </router-link>  
</template>  
  
<script lang="ts">  
import { Component, Prop, Vue } from "vue-property-decorator";  
import { UserLoader } from "@/util";  
import { Views } from "@/router";  
  
*// Иконка пользователя*@Component({})  
export default class UserThumbnail extends Vue {  
 private Views = Views;  
 @Prop({ required: true }) readonly loader!: UserLoader;  
}  
</script>  
  
<style scoped lang="scss">  
.card-image {  
 .open {  
 transition: all 0.3s ease-in-out;  
 top: 0;  
 bottom: 0;  
 left: 0;  
 right: 0;  
 opacity: 0;  
 }  
  
 img,  
 svg {  
 object-fit: cover;  
 aspect-ratio: 1/1;  
 }  
  
 &:hover {  
 .open {  
 opacity: 1;  
 }  
 }  
}  
  
.self-profile {  
 cursor: default;  
 .hover {  
 display: none;  
 }  
}  
</style>

***Файл src/views/Friends.vue***

<template>  
 <section v-if="user.loader?.data">  
 <h3 v-if="user.loader.data.friendsRequests.length" class="mt-3">Заявки</h3>  
 <UsersList :list="user.loader.data.friendsRequests" />  
 <h3 class="mt-3">Друзья</h3>  
 <UsersList :list="user.loader.data.friends" />  
 </section>  
</template>  
  
<script lang="ts">  
import { Component, InjectReactive, Vue } from "vue-property-decorator";  
import { UserController } from "@/util";  
import UsersList from "@/components/lists/UsersList.vue";  
  
*// Список друзей*@Component({  
 components: { UsersList },  
})  
export default class Friends extends Vue {  
 @InjectReactive() readonly user!: UserController;  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/views/LoginView.vue***

<template>  
 <section>  
 <header  
 class="header container-fluid py-2 px-5 bg-primary text-light fw-bold fs-2"  
 >  
 <font-awesome-icon icon="fa-solid fa-users-rays" />  
 </header>  
 <form  
 class="login container border shadow rounded-3 col-md-8 col-xl-6 mt-5 p-5"  
 @submit.prevent="submit"  
 >  
 <div class="fs-3">  
 <label for="login" class="form-label">Email</label>  
 <input  
 v-model="user.email"  
 type="text"  
 class="form-control"  
 :class="{ 'is-invalid': !valid }"  
 @keydown="reset"  
 id="login"  
 required  
 />  
  
 <label for="pwd" class="mt-3 form-label">Пароль</label>  
 <input  
 v-model="user.password"  
 type="password"  
 class="form-control"  
 :class="{ 'is-invalid': !valid }"  
 @keydown="reset"  
 id="pwd"  
 required  
 />  
 </div>  
 <div  
 class="mt-4 d-flex flex-column align-items-center justify-content-center fs-3 fw-bold gap-2"  
 >  
 <button type="submit" class="btn btn-primary col-8 col-sm-6">  
 Вход  
 </button>  
  
 <button  
 type="button"  
 @click="register"  
 class="btn btn-info col-8 col-sm-6"  
 >  
 Регистрация  
 </button>  
 </div>  
 </form>  
 </section>  
</template>  
  
<script lang="ts">  
import { Component, InjectReactive, Vue } from "vue-property-decorator";  
import { UserAuthData } from "@/../api";  
import Toaster, { States } from "@/components/Toaster.vue";  
import { Views } from "@/router";  
import SignUpModal from "@/components/forms/SignUpModal.vue";  
import { UserController } from "@/util";  
  
*// Страница входа*@Component({})  
export default class LoginView extends Vue {  
 @InjectReactive() readonly toaster!: Toaster | null;  
 @InjectReactive() readonly signUp!: SignUpModal | null;  
 private user: UserAuthData = { email: "", password: "" };  
 private valid = true;  
  
 private async submit() {  
 if (await UserController.login(this.user))  
 await this.$router.push({ name: Views.PROFILE });  
 else {  
 this.toaster?.show("Неверные данные для входа", States.DANGER);  
 this.valid = false;  
 }  
 }  
  
 private reset() {  
 this.valid = true;  
 }  
  
 register() {  
 this.signUp?.show();  
 }  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/views/MyProfile.vue***

<template>  
 <section>  
 <UserCard :id="user.id" configurable="true" />  
 <button  
 class="d-block btn btn-primary my-2"  
 @click="avatarUpdate.show(user)"  
 >  
 Сменить фото профиля  
 </button>  
 <button class="d-block btn btn-primary my-2" @click="makePost">  
 Новая запись  
 </button>  
 <PostsList :list="posts.list" class="mb-3"></PostsList>  
 </section>  
</template>  
  
<script lang="ts">  
import { *Component*, *InjectReactive*, Vue } from "vue-property-decorator";  
import { PostsLoader, UserController } from "@/util";  
import UserCard from "@/components/UserCard.vue";  
import PostsList from "@/components/lists/PostsList.vue";  
import UpdateAvatarModal from "@/components/forms/UpdateAvatarModal.vue";  
import PostModal from "@/components/forms/PostModal.vue";  
  
*// Собственный профиль*@Component({  
 components: { PostsList, UserCard },  
})  
export default class MyProfile extends Vue {  
 @InjectReactive() readonly user!: UserController;  
 @InjectReactive() readonly avatarUpdate!: UpdateAvatarModal;  
 @InjectReactive() readonly postModal!: PostModal;  
 posts: PostsLoader | null = null;  
  
 private created() {  
 if (this.user.id) {  
 this.posts = new PostsLoader(this.user.id);  
 this.posts.fetch();  
 }  
 }  
  
 private makePost() {  
 if (this.posts) this.postModal.show(this.posts);  
 }  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/views/Network.vue***

<template>  
 <section>  
 <nav class="navbar navbar-expand-sm px-5 bg-light">  
 <font-awesome-icon  
 class="text-primary navbar-brand fs-2"  
 icon="fa-solid fa-users-rays"  
 />  
 <button  
 class="navbar-toggler text-info"  
 type="button"  
 data-bs-toggle="collapse"  
 data-bs-target="#navbarContent"  
 >  
 <span class="navbar-toggler-icon"></span>  
 </button>  
 <div class="collapse navbar-collapse" id="navbarContent">  
 <ul class="navbar-nav">  
 <li class="nav-item" v-for="route of routes" :key="route[1]">  
 <router-link  
 class="nav-link"  
 :to="{ name: route[1] }"  
 exact-active-class="text-primary"  
 >  
 {{ route[0] }}  
 </router-link>  
 </li>  
 </ul>  
 </div>  
 </nav>  
 <main v-if="user.id" class="container">  
 <transition name="network-view" mode="out-in">  
 <router-view class="col-lg-8 m-auto"></router-view>  
 </transition>  
 </main>  
  
 <RedactModal ref="redact" />  
 <UpdateAvatarModal ref="avatarUpdate" />  
 <PostModal ref="postModal" />  
 </section>  
</template>  
  
<script lang="ts">  
import {  
 Component,  
 InjectReactive,  
 ProvideReactive,  
 Vue,  
} from "vue-property-decorator";  
import { Views } from "@/router";  
import { SocketManager, UserController } from "@/util";  
import RedactModal from "@/components/forms/RedactModal.vue";  
import UpdateAvatarModal from "@/components/forms/UpdateAvatarModal.vue";  
import PostModal from "@/components/forms/PostModal.vue";  
import { UserData } from "../../api";  
import Toaster, { States } from "@/components/Toaster.vue";  
  
*// Главный view соцсети для авторизованных пользователей*@Component({  
 components: { PostModal, UpdateAvatarModal, RedactModal },  
})  
export default class Network extends Vue {  
 @InjectReactive() readonly toaster!: Toaster | null;  
  
 private Views = Views;  
 private get routes() {  
 const requests = this.user.loader?.data?.friendsRequests.length;  
 return [  
 ["Моя страница", Views.PROFILE],  
 ["Новости", Views.NEWS],  
 ["Друзья " + (requests ? `(${requests})` : ""), Views.FRIENDS],  
 ["Фотографии", Views.PHOTOS],  
 ["Пользователи", Views.USERS\_LIST],  
 ];  
 }  
  
 $refs!: {  
 avatarUpdate: UpdateAvatarModal;  
 redact: RedactModal;  
 postModal: PostModal;  
 };  
  
 private async created() {  
 await this.user.fetchId();  
 await this.user.loader?.fetch();  
 this.redact = this.$refs.redact;  
 this.avatarUpdate = this.$refs.avatarUpdate;  
 this.postModal = this.$refs.postModal;  
 this.socket.addListener(this.postNotification);  
 }  
  
 private beforeDestroy() {  
 this.socket.removeListener(this.postNotification);  
 }  
  
 private notification = require("../assets/notification.mp3");  
  
 *// Уведомление о новости* private postNotification(user: UserData) {  
 this.toaster?.show(  
 `Пользователь ${user.surname} ${user.name} опубликовал новую запись`,  
 States.PRIMARY,  
 2000,  
 new Audio(this.notification)  
 );  
 }  
  
 *// Объект текущего пользователя* @ProvideReactive() user: UserController = new UserController();  
 *// Менеджер сокета* @ProvideReactive() socket: SocketManager = new SocketManager();  
 @ProvideReactive() redact: RedactModal | null = null;  
 @ProvideReactive() avatarUpdate: UpdateAvatarModal | null = null;  
 @ProvideReactive() postModal: PostModal | null = null;  
}  
</script>  
  
<style scoped lang="scss">  
.network-view-enter-active,  
.network-view-leave-active {  
 transition: all 0.3s;  
}  
.network-view-enter,  
.network-view-leave-to {  
 opacity: 0;  
}  
</style>

***Файл src/views/News.vue***

<template>  
 <section>  
 <PostsList :list="posts.list" class="mb-2"></PostsList>  
 </section>  
</template>  
  
<script lang="ts">  
import { Component, InjectReactive, Vue } from "vue-property-decorator";  
  
import { FriendsPostsLoader, SocketManager, UserController } from "@/util";  
import PostsList from "@/components/lists/PostsList.vue";  
import { UserData } from "../../api";  
  
*// Страница новостей*@Component({ components: { PostsList } })  
export default class News extends Vue {  
 @InjectReactive() readonly user!: UserController;  
 @InjectReactive() readonly socket!: SocketManager;  
 posts: FriendsPostsLoader | null = null;  
  
 private created() {  
 if (this.user.id) {  
 this.posts = new FriendsPostsLoader(this.user.id);  
 this.posts.fetch();  
 }  
 this.socket.addListener(this.postsUpdate);  
 }  
  
 private beforeDestroy() {  
 this.socket.removeListener(this.postsUpdate);  
 }  
  
 private postsUpdate(user: UserData) {  
 this.posts?.fetch();  
 }  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/views/Photos.vue***

<template>  
 <section>  
 <PhotosList :list="photos.list" class="mt-3"></PhotosList>  
 </section>  
</template>  
  
<script lang="ts">  
import { Component, InjectReactive, Vue } from "vue-property-decorator";  
import PhotosList from "@/components/lists/PhotosList.vue";  
import { PhotosLoader, UserController } from "@/util";  
  
*// Страница своих фотографий*@Component({  
 components: { PhotosList },  
})  
export default class Photos extends Vue {  
 @InjectReactive() readonly user!: UserController;  
 photos: PhotosLoader | null = null;  
  
 private created() {  
 if (this.user.id) {  
 this.photos = new PhotosLoader(this.user.id);  
 this.photos.fetch();  
 }  
 }  
}  
</script>  
  
<style scoped lang="scss"></style>

***Файл src/views/Profile.vue***

<template>  
 <section>  
 <UserCard  
 :id="id"  
 class="m-auto"  
 :key="id"  
 :configurable="user.isAdmin"  
 ></UserCard>  
  
 <nav class="col-lg-8 m-auto mt-2 btn-group w-100">  
 <router-link  
 v-for="link of categories"  
 :key="link[0]"  
 :to="{ name: link[0] }"  
 class="btn btn-outline-primary"  
 exact-active-class="active"  
 >  
 {{ link[1] }}  
 </router-link>  
 </nav>  
  
 <div class="my-3">  
 <transition name="user-view" mode="out-in">  
 <router-view v-if="loader?.list" :list="loader.list"></router-view>  
 </transition>  
 </div>  
 </section>  
</template>  
  
<script lang="ts">  
import { Component, InjectReactive, Vue, Watch } from "vue-property-decorator";  
import UserCard from "@/components/UserCard.vue";  
import { Route } from "vue-router";  
import { Views } from "@/router";  
import {  
 FriendsLoader,  
 FriendsPostsLoader,  
 ListLoader,  
 PhotosLoader,  
 PostsLoader,  
 UserController,  
} from "@/util";  
  
*// Страница пользователя*@Component({  
 components: { UserCard },  
})  
export default class Profile extends Vue {  
 private id = 0;  
 private loader: ListLoader | null = null;  
 @InjectReactive() readonly user!: UserController;  
  
 readonly categories = [  
 [Views.USER\_POSTS, "Новости"],  
 [Views.USER\_PHOTOS, "Фото"],  
 [Views.USER\_FRIENDS, "Друзья"],  
 [Views.USER\_FRIENDSPOSTS, "Новости друзей"],  
 ];  
  
 private mounted() {  
 this.onRouteChange(this.$route);  
 }  
  
 @Watch("$route")  
 onRouteChange(to: Route) {  
 const id = parseInt(this.$route.params.user\_id);  
 if (id === this.user.id) {  
 this.$router.push({ name: Views.PROFILE });  
 }  
 this.id = id;  
  
 (this.loader =  
 ((): ListLoader | undefined => {  
 switch (to.name as Views) {  
 case Views.USER\_POSTS:  
 return new PostsLoader(this.id);  
 case Views.USER\_PHOTOS:  
 return new PhotosLoader(this.id);  
 case Views.USER\_FRIENDS:  
 return new FriendsLoader(this.id);  
 case Views.USER\_FRIENDSPOSTS:  
 return new FriendsPostsLoader(this.id);  
 }  
 })() ?? null)?.fetch();  
 }  
}  
</script>  
  
<style scoped lang="scss">  
.user-view-enter-active,  
.user-view-leave-active {  
 transition: all 0.3s;  
}  
.user-view-enter,  
.user-view-leave-to {  
 opacity: 0;  
}  
</style>

***Файл src/views/UsersListView.vue***

<template>  
 <section class="container mt-3 mb-3">  
 <UsersList :list="users.list"></UsersList>  
 </section>  
</template>  
  
<script lang="ts">  
import { Component, Vue } from "vue-property-decorator";  
import UsersList from "@/components/lists/UsersList.vue";  
import { UsersListLoader } from "@/util";  
  
*// Страница списка всех пользователей*@Component({  
 components: { UsersList },  
})  
export default class UsersListView extends Vue {  
 private users: UsersListLoader = new UsersListLoader();  
  
 private mounted() {  
 this.users.fetch();  
 }  
}  
</script>  
  
<style scoped lang="scss"></style>