Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _***ИУК «Информатика и управление»***______________

КАФЕДРА __***ИУК4 «Программное обеспечение ЭВМ и информационные технологии»***_____

ДОМАШНЯЯ РАБОТА

ДИСЦИПЛИНА: «ПРОЕКТИРОВАНИЕ СИСТЕМ ХРАНЕНИЯ И ОБРАБОТКИ ДАННЫХ»

Выполнил: студент гр. ИУК4-52Б ________________ (_____Губин Е.В._____)
(Подпись) (Ф.И.О.)

Проверил: ________________ (____Глебов С.А._____)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга , 2024

Цель: Разработка клиентского приложения (CRUD-операции с таблицами БД)

Сервер

Роутер для постов:

```
import { Router } from "express";
import postController from "../controllers/post-controller";
import authMiddleware from "../middlewares/auth-
middleware";
import { param } from "express-validator";

const postRouter = Router();

postRouter.post("/", authMiddleware,
postController.newPost);
postRouter.get("/all", authMiddleware,
postController.getAllPosts)
postRouter.get(
    "/repost/:childrenPostId",
    param("childrenPostId").isNumeric(),
    authMiddleware,
    postController.getPostsByChildrenPostId
);
postRouter.get(
    "/posts/:userId",
    param("userId").isNumeric(),
    authMiddleware,
    postController.getPostsByUserId
);
postRouter.get(
    "/:postId",
    param("postId").isNumeric(),
    authMiddleware,
    postController.getPostById
);
postRouter.delete(
    "/:postId",
    param("postId").isNumeric(),
    authMiddleware,
    postController.deletePost
);

export default postRouter;
```

Контроллер для постов:

```
import { NextFunction, Request, Response } from "express";
import INewPost from "../interfaces/INewPost";
import postService from "../services/post-service";
import { validationResult } from "express-validator";
import ApiError from "../exceptions/ApiError";

class PostController {
    async newPost(req: Request, res: Response, next:
NextFunction) {
        try {
            const newPost: INewPost = req.body;
            const authorId =
Number(res.locals.userData.userId);
            const newPostData = await
postService.newPost(newPost, authorId);
            res.json({ ...newPostData });
        } catch (error) {
            next(error);
        }
    }

    async getPostsByUserId(req: Request, res: Response,
next: NextFunction) {
        try {
            const errors = validationResult(req);
            if (!errors.isEmpty()) {
                throw ApiError.BadRequest("Incorrect
userId", errors.array());
            }
            const userId = Number(req.params.userId);
            const posts = await
postService.getPostsByUserId(userId);
            res.json({ posts });
        } catch (error) {
            next(error);
        }
    }
    async getPostById(req: Request, res: Response, next:
NextFunction) {
        try {
            const errors = validationResult(req);
            if (!errors.isEmpty()) {
                throw ApiError.BadRequest("Incorrect post
id", errors.array());
            }
            const postId = Number(req.params.postId);
            const postData = await
postService.getPostById(postId);
            res.json({ ...postData });
        } catch (error) {
            next(error);
        }
    }

    async deletePost(req: Request, res: Response, next:
NextFunction) {
        try {
            const errors = validationResult(req);
            if (!errors.isEmpty()) {
                throw ApiError.BadRequest("Incorrect
postId", errors.array());
            }
            const postId = Number(req.params.postId);
            const post = await
postService.deletePostById(postId);
```

```
            res.json({ ...post });
        } catch (error) {
            next(error);
        }
    }

    async getPostsByChildrenPostId(
        req: Request,
        res: Response,
        next: NextFunction
    ) {
        try {
            const errors = validationResult(req);
            if (!errors.isEmpty()) {
                throw ApiError.BadRequest(
                    "Incorrect children post id",
                    errors.array()
                );
            }
            const childrenPostId =
Number(req.params.childrenPostId);
            const posts = await
postService.getPostsByChildrenPostId(
                childrenPostId
            );
            res.json({ posts });
        } catch (error) {
            next(error);
        }
    }

    async getAllPosts(req: Request, res: Response, next:
NextFunction) {
        try {
            const posts = await postService.getAllPosts();
            res.json({ posts });
        } catch (error) {
            next(error);
        }
    }
}

export default new PostController();
```

Сервис для постов:

```
import INewPost from "../interfaces/INewPost";
import db from "../db";
import IPostFromDataBase from
"../interfaces/IPostFromDataBase";
import PostDto from "../dtos/post-dto";
import userService from "./user-service";
import ApiError from "../exceptions/ApiError";
import dateTimeService from "./dateTime-service";

class PostService {
    async newPost(newPost: INewPost, authorId: number) {
        if (!(await userService.userIsExistsById(authorId))) {
            throw ApiError.BadRequest("User with this id
aren't exist");
        }
        if (
            newPost.childrenPostId &&
            !(await
this.postIsExistsById(newPost.childrenPostId))
        ) {
            throw ApiError.BadRequest("Children post isn't
found");
        }

        const nowFormattedDateTime =
dateTimeService.getNowDate();

        const postFromDataBase: IPostFromDataBase = (
            await db.query(
                `INSERT INTO posts (content,
publication_date_time, children_post_id, post_author_id)
VALUES ($1, $2, $3, $4) RETURNING *`,
                [
                    newPost.content,
                    nowFormattedDateTime,
                    newPost.childrenPostId,
                    authorId,
                ]
            )
        ).rows[0];

        postFromDataBase.publication_date_time =
dateTimeService.formatDateTime(
            postFromDataBase.publication_date_time
        );

        return new PostDto(postFromDataBase);
    }

    async getPostsByUserId(userId: number) {
        if (!(await userService.userIsExistsById(userId))) {
            throw ApiError.BadRequest("User with this id
aren't exist");
        }
        if (await userService.userIsExistsById(userId)) {
            const posts: IPostFromDataBase[] = (
                await db.query(
                    `SELECT * FROM posts WHERE
post_author_id = $1`,
                    [userId]
                )
            ).rows;
            return posts.map((post) => {
                post.publication_date_time =
dateTimeService.formatDateTime(
                    post.publication_date_time
                );
                return new PostDto(post);
            });
        }
        throw ApiError.BadRequest("User with this id aren't
exists");
    }

    async getPostById(postId: number) {
        if (!(await this.postIsExistsById(postId))) {
            throw ApiError.ResourseNotFound();
        }
        const postData: IPostFromDataBase = (
            await db.query("SELECT * FROM posts WHERE
post_id = $1", [postId])
        ).rows[0];
        postData.publication_date_time =
dateTimeService.formatDateTime(
            postData.publication_date_time
        );
        return new PostDto(postData);
    }

    async deletePostById(postId: number) {
        if (await this.postIsExistsById(postId)) {
            const post: IPostFromDataBase = (
                await db.query(
                    "DELETE FROM posts WHERE post_id =
$1 RETURNING *",
                    [postId]
                )
            ).rows[0];
            post.publication_date_time =
dateTimeService.formatDateTime(
                post.publication_date_time
            );
            return new PostDto(post);
        }
        throw ApiError.BadRequest("Post with this id aren't
found");
    }

    async postIsExistsById(postId: number) {
        const postById: IPostFromDataBase[] = (
            await db.query(`SELECT * FROM posts WHERE
post_id = $1`, [postId])
        ).rows;
        if (postById.length == 0) {
            return false;
        }
        return true;
    }

    async getPostsByChildrenPostId(childrenPostId: number)
{
        if (!(await this.postIsExistsById(childrenPostId))) {
```

```
            throw ApiError.BadRequest("Children post isn't
found");
        }
        const posts: IPostFromDataBase[] = (
            await db.query("SELECT * FROM posts WHERE
children_post_id = $1", [
                childrenPostId,
            ])
        ).rows;
        return posts.map((post) => {
            post.publication_date_time =
dateTimeService.formatDateTime(
                post.publication_date_time
            );
            return new PostDto(post);
        });
    }

    async getPostIdsByAuthorId(authorId: number) {
        if (!(await userService.userIsExistsById(authorId))) {
            throw ApiError.BadRequest("Author id isn't
found");
        }
        const ids: number[] = (
            await db.query(
                "SELECT post_id FROM posts WHERE
post_author_id = $1",
                [authorId]
            )
        ).rows;
        return ids;
    }

    async getAllPosts() {
        const posts: IPostFromDataBase[] = (
            await db.query("SELECT * FROM posts", [])
        ).rows;
        return posts.map((post) => {
            post.publication_date_time =
dateTimeService.formatDateTime(
                post.publication_date_time
            );
            return new PostDto(post);
        });
    }
}

export default new PostService();
```

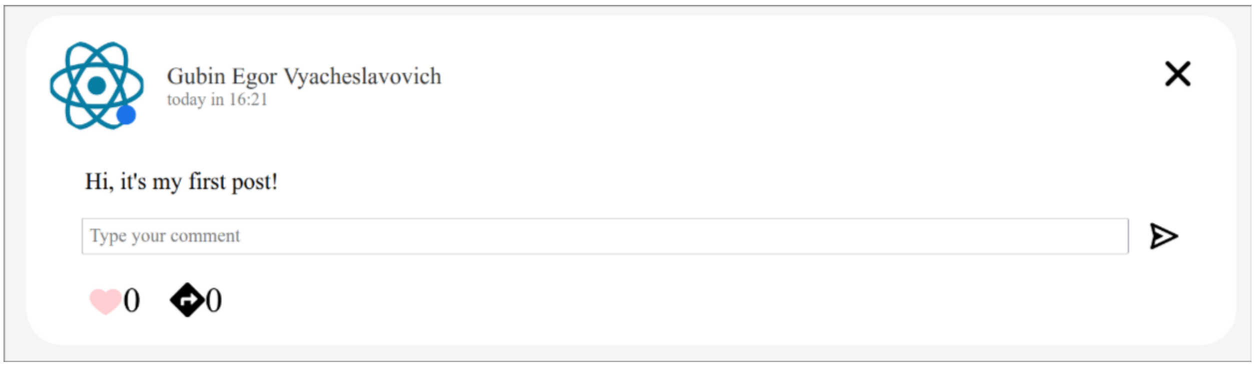
Клиент

Axios для работы с сервером над постами:

```
import $api from "../http";
import IGetPost from "../interfaces/IResponses/IGetPost";

export default class PostService {
    static async getPostsByUserId(userId: number) {
        return await $api.get<{ posts: IGetPost[]
}>(`/post/posts/${userId}`);
    }

    static async getPostById(postId: number) {
        return await $api.get<IGetPost>(`/post/${postId}`);
    }

    static async getPostsByChildrenPostId(childrenPostId:
number) {
        return await $api.get<{ posts: IGetPost[] }>(
            `/post/repost/${childrenPostId}`
        );
    }

    static async newPost(
        content: string,
        childrenPostId: number | null = null
    ) {
        return await $api.post<IGetPost>("/post", {
content, childrenPostId });
    }

    static async deletePost(postId: number) {
        return await
$api.delete<IGetPost>(`/post/${postId}`);
    }
    static async getALlPosts() {
        return await $api.get<{ posts: IGetPost[]
}>("/post/all");
    }
}
```

Пользовательский интерфейс

Компонент пост:


Gubin Egor Vyacheslavovich
today in 16:21

Hi, it's my first post!

Type your comment

0 0

Рис. 1: Пост

```
import React, { ChangeEvent, useContext, useEffect,
useState } from "react";
import s from "./Post.module.css";
import IPost from "../../interfaces/IProps/IPost";
import DateTimeService from "../../services/dateTime-
service";
import PostService from "../../services/post-service";
import IGetPost from "../../interfaces/IResponses/IGetPost";
import IUser from "../../interfaces/IResponses/IUser";
import UserService from "../../services/user-service";
import ProfileImageService from "../../services/profileImage-
service";
import classNames from "classnames";
import IGetComment from
"../../interfaces/IResponses/IGetComment";
import Comment from "../Comment/Comment";
import CommentService from "../../services/comment-
service";
import { BiSend } from "react-icons/bi";
import { FcLikePlaceholder } from "react-icons/fc";
import { FcLike } from "react-icons/fc";
import ReactionService from "../../services/reaction-service";
import IGetReaction from
"../../interfaces/IResponses/IGetReaction";
import { Context } from "../..";
import { FaDirections } from "react-icons/fa";
import { observer } from "mobx-react-lite";
import { IoClose } from "react-icons/io5";
import PostImageService from "../../services/postImage-
service";
import ImageSlider from "../ImageSlider/ImageSlider";
import { useNavigate } from "react-router-dom";
import io from "socket.io-client";
import { globalSocket } from "../../globalSocket";
import { GrStatusGoodSmall } from "react-icons/gr";

const Post: React.FC<IPost> = ({
    post,
    isChild,
    setCreatePostFormIsOpened,
    setRepost,
    setPosts,
}) => {
    const { store } = useContext(Context);

    const [postAvatarImage, setPostAvatarImage] =
useState(“”);
    const [author, setAuthor] = useState({} as IUser);
    const [postImages, setPostImages] =
useState<string[]>([]);
    const [comments, setComments] =
useState<IGetComment[]>([]);
    const [childPost, setChildPost] = useState<JSX.Element
| null>(null);
    const [showAllComments, setShowAllComments] =
useState<boolean>(false);
    const [newComment, setNewComment] = useState(“”);
    const [isReaction, setIsReaction] = useState(false);
    const [reactionsAmount, setReactionsAmount] =
useState(0);
    const [repostsAmount, setRepostsAmount] =
useState(0);
    const [isOnline, setIsOnline] = useState(false)

    const navigate = useNavigate();
    const openPageByAvatar = () => {
        if (store.user.userId !== post.postAuthorId) {
            navigate(`/profile/${post.postAuthorId}`);
        }
    };

    const loadPostAvatarImage = async () => {
        setPostAvatarImage(
            (await
ProfileImageService.getProfileImage(post.postAuthorId)).data
                .src
        );
    };

    const loadAuthor = async () => {
        setAuthor((await
UserService.getUserById(post.postAuthorId)).data);
    };

    const loadPostImages = async () => {
        setPostImages(
            (await
PostImageService.getPostImages(post.postId)).data.postImage
s
        );
    };

    const loadComments = async () => {
        setComments(
            (
                await
CommentService.getCommentsByPostId(post.postId)
            ).data.comments.sort((first, second) => {
                const dateFirst = new
Date(first.commentDateTime);
                const dateSecond = new
Date(second.commentDateTime);
                return dateSecond.getTime() -
dateFirst.getTime();
            })
        );
    };

    const loadChildPost = async () => {
        if (post.childrenPostId) {
            const childPostData = (
                await
PostService.getPostById(post.childrenPostId)
            ).data;
            setChildPost(<Post post={childPostData}
isChild={true} />);
        }
    };

    const laodReactions = async () => {
        const reactionsData = (
            await
ReactionService.getReactionsByPostId(post.postId)
        ).data.reactions;
        setReactionsAmount(reactionsData.length);
        for (let i = 0; i < reactionsData.length; i++) {
            if (reactionsData[i].reactionAuthorId ===
store.user.userId) {
                setIsReaction(true);
                return;
            }
        }
    };

    const loadReposts = async () => {
        setRepostsAmount(
            (await
PostService.getPostsByChildrenPostId(post.postId)).data.post
s
                .length
        );
    };

    const loadIsOnline = async () => {
        setIsOnline((await
UserService.getStatus(post.postAuthorId)).data.isOnline)
    }

    useEffect(() => {
        loadAuthor();
        loadPostImages();
        loadChildPost();
        loadPostAvatarImage();
        laodReactions();
        loadReposts();
        loadIsOnline()
        if (!isChild) {
            loadComments();
        }
        const socket = io(globalSocket);
        socket.emit("subscribe_image", {
            userId: post.postAuthorId,
        });
        socket.emit("subscribe_like", {
            postId: post.postId,
            authorId: store.user.userId,
        });
        socket.emit("subscribe_online", {userId:
post.postAuthorId });
        socket.on("set_image", () => {
            loadPostAvatarImage();
        });
        socket.on("set_like", ({ operation }: { operation:
number }) => {
            console.log(operation)
            setReactionsAmount((prev) => prev +
operation);
```

```
        });
        socket.on("set_status", ({ isOnline }: { isOnline:
boolean }) => {
            setIsOnline(isOnline);
        });
        return () => {
            socket.off("set_status");
            socket.off("set_like");
            socket.off("set_image");
            socket.disconnect();
        };
    }, []);

    return (
        <div className={classNames(s.post, { [s.left_border]:
isChild })}>
            <div className={s.post_header}>
                <div
                    className={s.profile_post_image}
                    onClick={openPageByAvatar}
                    style={{
                        backgroundImage:
`url(${postAvatarImage})`,
                    }}
                >
                {isOnline && (
                    <GrStatusGoodSmall
className={s.online} />
```

```
                    )}</div>
                    <div className={s.date_time_fio}>
                        <div className={s.post_fio}>
                            {[
                                author.lastName,
                                author.firstName,
                                author.patronymic,
                            ].join(" ")}
                        </div>
                        <div className={s.pub_post_date_time}>

{DateTimeService.formDate(post.publicationDateTime)}
                        </div>
                    </div>
                </div>
                <div
className={s.post_content}>{post.content}</div>
                <ImageSlider images={postImages} />
                {childPost}
                {comments.length !== 0 &&
                    comments.map((comment, index) => {
                        if (showAllComments ||
comments.length <= 2)
                            return (
                                <Comment
                                    isMyPost={
                                        store.user.userId ===
post.postAuthorId
```

```
                                }
                                comment={comment}
                                key={comment.commentId}

setComments={setComments}
                            />
                        );
                    if (index <= 1)
                        return (
                            <Comment
                                isMyPost={
                                    store.user.userId ===
post.postAuthorId
                                }
                                comment={comment}
                                key={comment.commentId}

setComments={setComments}
                            />
                        );
                    return null;
                })}
            {comments.length > 2 ? (
                showAllComments ? (
                    <span
                        className={s.manage_comments}
                        onClick={() =>
setShowAllComments(false)}
```

```
                    >
                        Hide comments
                    </span>
                ) : (
                    <span
                        className={s·manage_comments}
                        onClick={() =>
setShowAllComments(true)}
                    >
                        Show all commets···
                    </span>
                )
            ) : null}
            {!isChild ? (
                <div className={s·new_comment}>
                    <textarea
                        placeholder="Type your comment"
                        className={s·comment_area}
                        rows={1}
                        value={newComment}
                        onChange={(event) => {
                            const textarea = event·target;

                            textarea·style·height = "auto";

                            textarea·style·height =
                                textarea·scrollHeight + "px";
```

```
setNewComment(event·target·value);
                        }}
                    ></textarea>
                    <BiSend
                        onClick={() => {

CommentService·newComment(newComment, post·postId)
                                    ·then((response) =>
response·data)
                                    ·then((data) =>
                                        setComments((prev) =>
[data, ···prev])
                                    );
                            setNewComment("");
                        }}
                        className={s·send_comment}
                    />
                </div>
            ) : null}
            {isChild ? null : (
                <div className={s·reaction_repost}>
                    <div
                        className={s·reactions_amount}
                        onClick={
                            isReaction
                                ? () => {
```

```
ReactionService.deleteReaction(
                                        post.postId
                                    )
                                        .then((response)
=> response.data)
                                        .then((data) => {

setReactionsAmount(
                                                (prev) =>
prev - 1
                                            );

setIsReaction(false);
                                        });
                                     const socket =
io(globalSocket);

socket.emit("change_like", {
                                        postId:
post.postId,
                                        operation: -1,
                                        authorId:
store.user.userId,
                                    });
                                 }
                               : () => {
```

```
ReactionService.newReaction(post.postId)
                                                    .then((response)
=> response.data)
                                                    .then((data) => {

setReactionsAmount(
                                                              (prev) =>
prev + 1
                                                        );

setIsReaction(true);
                                                   });
                                               const socket =
io(globalSocket);

socket.emit("change_like", {
                                                    postId:
post.postId,
                                                    operation: 1,
                                                    authorId:
store.user.userId,
                                               });
                                           }
                                }
                          >
                              {isReaction ? (
                          <FcLike
className={s·reaction_button} />
                      ) : (
                          <FcLikePlaceholder
className={s·reaction_button} />
                      )}{" "}
                      {reactionsAmount}{" "}
                  </div>
                  <div
                      className={classNames({
                          [s·reposts_amount]:
                              post·postAuthorId !==
store·user·userId,
                          [s·repost_amount_unself]:
                              post·postAuthorId ===
store·user·userId,
                      })}
                      onClick={
                          post·postAuthorId ===
store·user·userId
                              ? () => {}
                              : () => {
                                    if (

setCreatePostFormIsOpened &&
                                          setRepost
                                    ) {
```

```
setCreatePostFormIsOpened(true);
                                                setRepost(post);
                                            }
                                        }
                            }
                        >
                            <FaDirections
className={s.repost_button} />{" "}
                            {repostsAmount}
                        </div>
                    </div>
                )}
                {isChild || post.postAuthorId !==
store.user.userId ? null : (
                    <IoClose
                        className={s.close}
                        onClick={() => {
                            PostService.deletePost(post.postId)
                                .then((response) =>
response.data)
                                .then((data) => {
                                    if (setPosts)
                                        setPosts((prev) =>
                                            prev.filter(
                                                (postData) =>

postData.postId !== post.postId
```

```
                                                    )
                                                );
                                            const socket =
io(globalSocket);
                                            socket.emit("delete_post", {
                                                postId: data.postId,
                                                authorId:
data.postAuthorId,
                                            });
                                        });
                                }}
                            />
                        )}
                    </div>
            );
};

export default observer(Post);
```

Форма для создания поста:

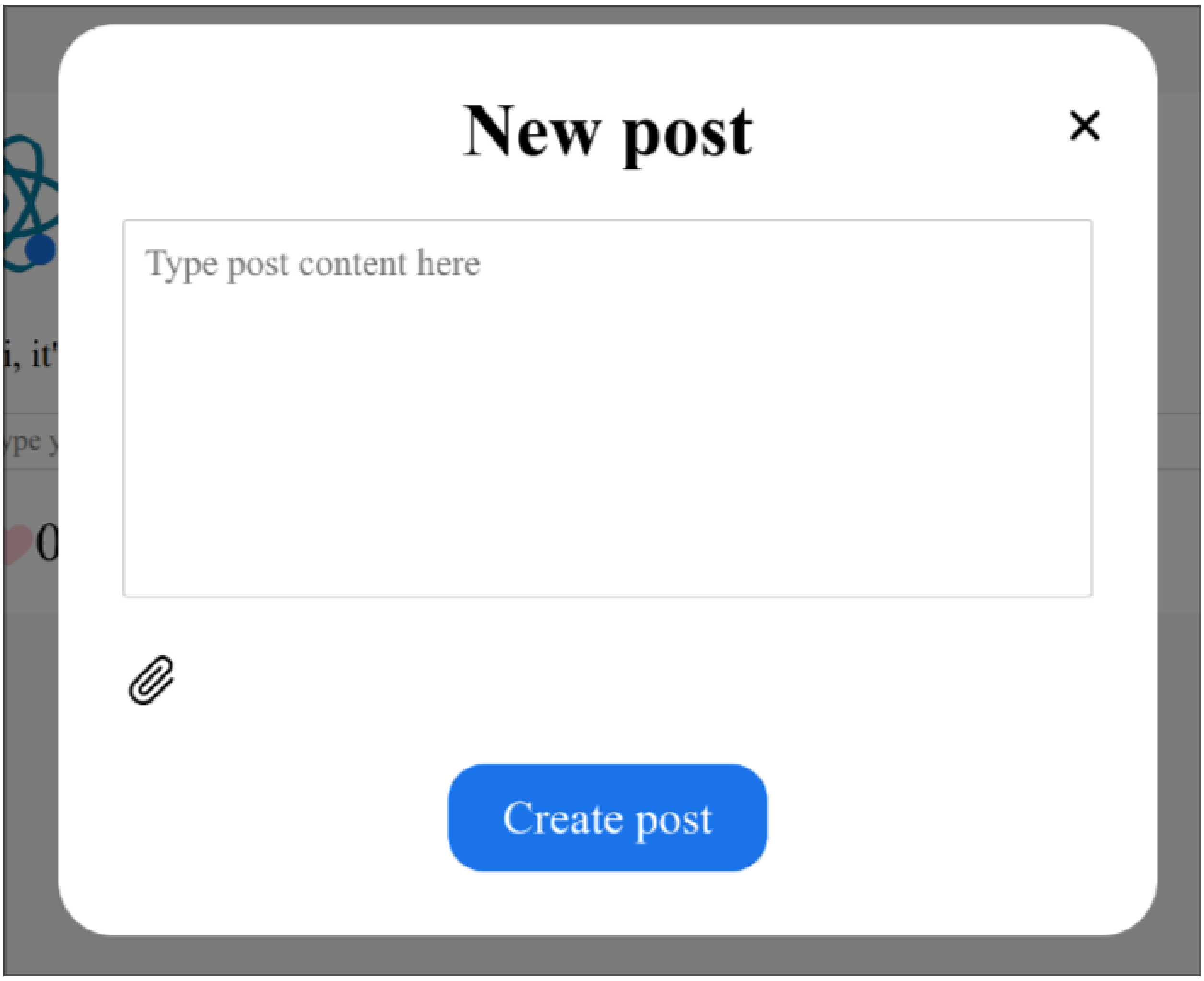


Рис. 2: Форма для создания поста

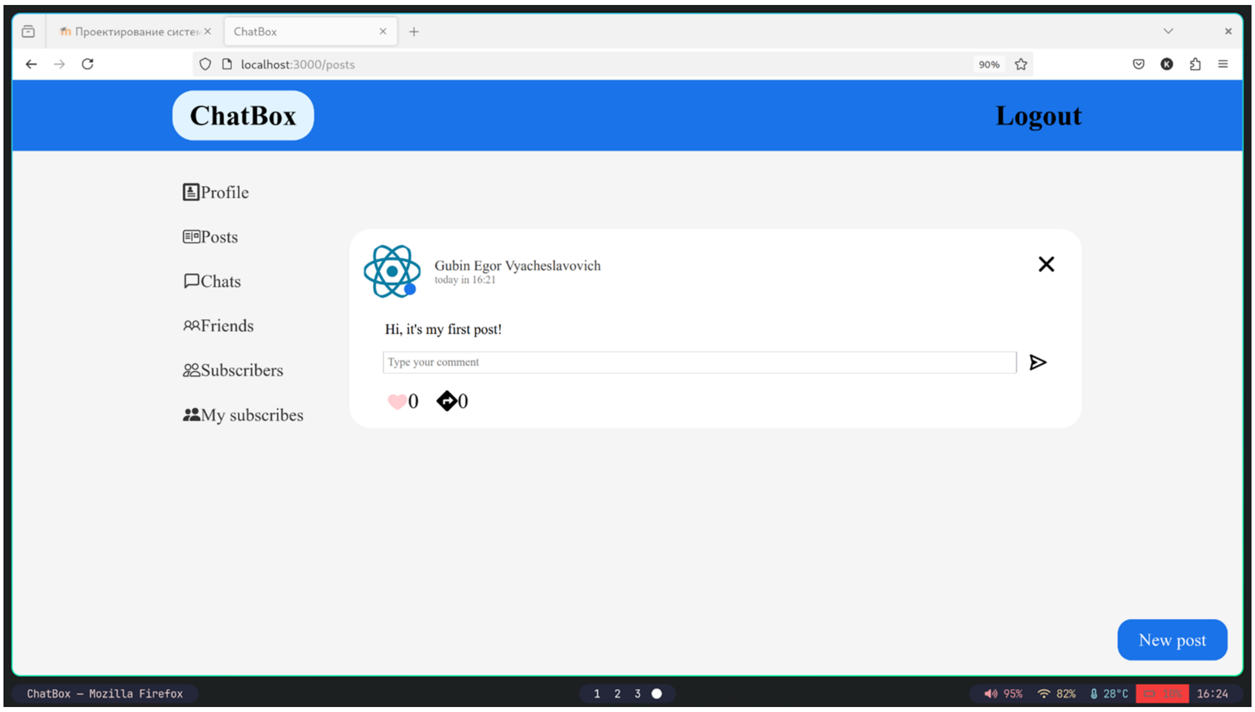
```
import React, { ChangeEvent, useRef, useState } from
"react";
import s from "./NewPostForm.module.css";
import { FormButton } from
"../UI/FormButton/FormButton";
import { GoPaperclip } from "react-icons/go";
import PostService from "../../services/post-service";
import PostImageService from "../../services/postImage-
service";
import INewPostForm from
"../../interfaces/IProps/INewPostForm";
```

```
import io from "socket.io-client";
import { globalSocket } from "../../globalSocket";

export const NewPostForm: React.FC<INewPostForm> = ({
    setPosts,
    repost,
    setRepost,
    setCreatePostFormIsOpened,
    isFromPostsPage,
}) => {
    const [content, setContent] = useState("");
    const fileInputRef = useRef<HTMLInputElement>(null);
    const [files, setFiles] = useState<File[]>([]);
    const [filesImages, setFilesImages] =
useState<string[]>([]);

    const setImages = (event:
ChangeEvent<HTMLInputElement>) => {
        const filesEvent = event.target.files;
        if (!filesEvent || filesEvent.length === 0) {
            return;
        }
        const permittedTypes = ["image/jpeg", "image/jpg",
"image/png"];
        for (let i = 0; i < filesEvent.length; i++) {
            if (!permittedTypes.includes(filesEvent[i].type))
{
                return;
            }
        }
        const filesArray = Array.from(filesEvent);
        setFiles(filesArray);
        setFilesImages(
            filesArray.map(
                (fileData) =>
`url(${URL.createObjectURL(fileData)})`
            )
        );
    };

    return (
        <div className={s.new_post_form}>
            <div className={s.container}>
                <textarea
                    rows={8}
                    placeholder="Type post content here"
                    className={s.content}
                    value={content}
                    onChange={(event) =>
setContent(event.target.value)}
                ></textarea>
                <div className={s.selected_images}>
                    <GoPaperclip
                        className={s.clip}
                        onClick={() =>
fileInputRef.current?.click()}
```

```
                    />
                    {filesImages.map((file) => (
                        <div
                            className={s.image}
                            style={{
                                backgroundImage: file,
                            }}
                        ></div>
                    ))}
                </div>
                <input
                    ref={fileInputRef}
                    type="file"
                    multiple={true}
                    accept=".jpg, .jpeg, .png"
                    onChange={setImages}
                    className={s.file_input}
                />
            </div>
            <FormButton
                onClick={
                    repost
                        ? async () => {
                              const post = (
                                  await
PostService.newPost(
                                      content,
                                      repost.postId
```

```
                                        )
                                    ).data;
                                    await
PostImageService.newPostImages(
                                        files,
                                        post.postId
                                    );
                                    setRepost(null);
                                    setContent("");
                                    setFiles([]);
                                    setFilesImages([]);

setCreatePostFormIsOpened(false);
                                    if (isFromPostsPage)
                                        setPosts((prev) => [post,
...prev]);
                                    const socket =
io(globalSocket);
                                    socket.emit("new_post", {
post });
                                }
                              : async () => {
                                    const post = (await
PostService.newPost(content))
                                        .data;
                                    await
PostImageService.newPostImages(
                                        files,
                                        post.postId
                                    );
                                    setContent("");
                                    setFiles([]);
                                    setFilesImages([]);

setCreatePostFormIsOpened(false);
                                    setPosts((prev) => [post,
...prev]);
                                    const socket =
io(globalSocket);
                                    socket.emit("new_post", {
post });
                                }
                            }
                            type="button"
                        >
                            Create post
                        </FormButton>
                    </div>
            );
};
```

Страница постов:



Hi, it's my first post!

Рис. 3: Страница постов

```
import React, { useEffect, useState } from "react";
import s from "./PostsPage.module.css";
import IGetPost from "../../interfaces/IResponses/IGetPost";
import PostService from "../../services/post-service";
import Post from "../Post/Post";
import { ModalWindow } from
"../ModalWindow/ModalWindow";
import { NewPostForm } from
"../NewPostForm/NewPostForm";
import { FormButton } from
"../UI/FormButton/FormButton";

export const PostsPage: React.FC = () => {
    const [posts, setPosts] = useState<IGetPost[]>([]);
    const [createPostFormIsOpened,
setCreatePostFormIsOpened] = useState(false);
```

```
    const [repost, setRepost] = useState<IGetPost |
null>(null);

    const loadPosts = async () => {
        const postsData = (await
PostService.getALlPosts()).data.posts;
        setPosts(
            postsData.sort((first, second) => {
                const dateFirst = new
Date(first.publicationDateTime);
                const dateSecond = new
Date(second.publicationDateTime);
                return dateSecond.getTime() -
dateFirst.getTime();
            })
        );
    };

    useEffect(() => {
        loadPosts();
    }, []);

    return (
        <div>
            {posts.map((post) => (
                <Post
                    key={post.postId}
                    post={post}
                    isChild={false}
                    setRepost={setRepost}

setCreatePostFormIsOpened={setCreatePostFormIsOpened}
                    setPosts={setPosts}
                />
            ))}
            <FormButton
                className={s.new_post}
                type="button"
                onClick={() =>
setCreatePostFormIsOpened(true)}
            >
                New post
            </FormButton>
            <ModalWindow
                isOpened={createPostFormIsOpened}
                setIsOpened={setCreatePostFormIsOpened}
                header="New post"
            >
                <NewPostForm

setCreatePostFormIsOpened={setCreatePostFormIsOpened}
                    setPosts={setPosts}
                    repost={repost}
                    setRepost={setRepost}
                    isFromPostsPage={true}
                />
                </ModalWindow>
            </div>
        );
    };
```

Вывод: в ходе лабораторной работы были реализованы CRUD операции для работы с постами.