Минобрнауки России

федеральное государственное бюджетное образовательное учреждение высшего образования «Санкт-Петербургский государственный технологический институт (технический университет)»

СПбГТИ(ТУ)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | | |
| Направленность программы | |  | Автоматизированные системы обработки информации и управления | | | |
|  | |  |  | | | |
| Факультет | |  | Информационных технологий и управления | | | |
| Кафедра | |  | Систем автоматизированного  проектирования и управления | | | |
| Учебная дисциплина | |  | Интернет-технологии | | | |
| Курс 3 |  | | | Группа | 404 |

**Отчет по контрольной работе № 3**

|  |  |
| --- | --- |
| **Тема:** | **Создание серверной части веб-сайта** |

Студент гр. 404 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. К. Азаров \_

(подпись, дата) (инициалы, фамилия)

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Иванов \_

(подпись, дата) (инициалы, фамилия)

Отметка о зачете \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись преподавателя)

Санкт-Петербург

2023

#### **1 Описание идеи проекта. Формирование функциональных требований к проекту**

В данной контрольной работе осуществляется продолжение развитие идеи проекта, описанной в контрольной работе №1.

**Цель проекта:** разработать интерфейса веб-приложения для шифрования текста.

**Функциональные требования к разрабатываемому веб-приложению:**

1) Предоставлять доступ к сайту.

2) Шифровать текст алгоритмами: AES, BLOWFISH, CAST, DES.

#### **2 Обзор и обоснование выбора архитектуры информационной системы, способа клиент-серверного взаимодействия**

Наиболее известные подходы к архитектуре:

– Многослойная архитектура (Layered Architecture);

– Многоуровневая архитектура (Tiered Architecture);

– Сервис-ориентированная архитектура (Service Oriented Architecture — SOA);

– Микросервисная архитектура (Microservice Architecture) [1].

Многослойная архитектура работает по принципу разделения ответственностей. Программное обеспечение разделено на слои, лежащие друг на друге, и каждый из них выполняет определенную обязанность. Преимущества: более простая реализация по сравнению с другими подходами; предлагает абстракцию благодаря разделению ответственностей между уровнями; изолирование защищает одни слои от изменений других; повышает управляемость программного обеспечения за счет слабой связанности. Недостатки: не предлагает большой масштабируемости; программное обеспечение, созданное с таким подходом, будет иметь монолитную структуру, усложняющую внесение модификаций; данные должны проходить по каждому слою, даже если нет необходимости передавать их с определенных слоев.

Многоуровневая архитектура разделяет комплекс программного обеспечения на уровни по принципу взаимодействия “клиент-сервер”. Архитектура может иметь один, два и больше уровней, разделяющих ответственности между поставщиком данных и потребителем. Этот подход использует шаблон Request Response для связи между уровнями. В отличие от многослойной архитектуры, он предлагает масштабируемость, которая может быть как горизонтальной (масштабирование сети с помощью высокопроизводительных узлов), так и вертикальной (масштабирование каждого узла путем повышения его производительности).

Сервис-ориентированная архитектура состоит из компонентов и приложений, которые связываются друг с другом с помощью четко определенных сервисов. Клиент отправляет запрос с использованием стандартного протокола и формата данных по сети. Этот запрос обрабатывается ESB (enterprise service bus — сервисная шина предприятия), которая считается сердцем сервис-ориентированной архитектуры и отвечает за оркестровку и маршрутизацию. С помощью сервисного репозитория ESB направляет запрос в специальный сервис, который может взаимодействовать с другими сервисами и базами данных, чтобы составить полезную нагрузку (данные) ответа.

Mикросервисная архитектура разрабатывается как набор небольших сервисов, каждый из которых работает в собственном процессе и связывается с легковесными механизмами, обычно API для HTTP-ресурса. Преимущества:

предлагает слабую связанность благодаря высокой степени изоляции; повышает модульность; cбой в одном сервисе не затронет всю систему, поскольку они изолированы; предлагает высокую гибкость и масштабируемость; простота модификации может ускорить итерации; позволяет реализовать улучшенную систему обработки ошибок; решает проблемы с потоками данных, которые бывают у многослойной архитектуры. Недостатки: повышенный риск сбоя при обмене данными между сервисами; большим количеством сервисов трудно управлять; требует решения таких проблем, как задержки в сети, балансировка нагрузки и прочих трудностей, свойственных распределенной архитектуре; нуждается в комплексном тестировании в распределенной среде; на реализацию потребуется гораздо больше времени.

Для реализации проекта выбрана трёхуровневая архитектура приложения, представленная на рисунке 1.

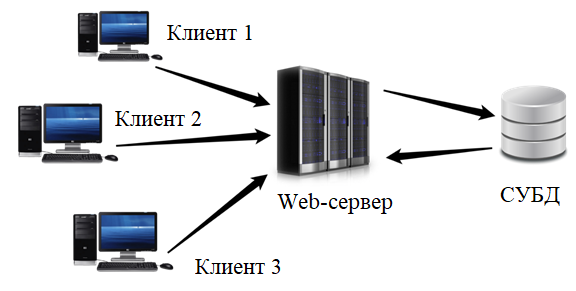


Рисунок 1 – Схема трехуровневой архитектура

Расчёт и визуализация результата будет производиться на стороне клиента, так как cхема трехуровневой архитектуры результатов на стороне сервера вносит дополнительную задержку ответа на действия пользователя.

Хранение данных будет осуществляться на стороне севера в серверной системе управления базами данных – это снизит требования к клиентскому оборудованию (требуется меньший объем накопителя).

Обработка запросов на получение статических файлов и данных из системы управления базами данных будет производиться на сервере.

Клиент и сервер осуществляют запросы посредством протокола HTTP. В целях обеспечения большей безопасности рекомендуется использовать расширение протокола HTTPS для поддержки шифрования данных.

Далее, необходимо выбрать систему управления базами данных. Для сравнения будут приведены PostgreSQL и MySQL, так как они являются наиболее популярными среди разработчиков.

Таблица 1 – Сравнение систем управления базами данных MySQL и PostgreSQL

|  |  |  |
| --- | --- | --- |
| Критерий | MySQL | PostgreSQL |
| Открытый код | + | + |
| ACID | Частично | + |
| Высокая производительность | + | Только при чтении данных |
| Поддержка NoSQL функции | - | + |
| Хранимые процедуры | - | + |

Для реализации в проекте хранения данных выбрана СУБД MongoDB обладающая весомыми преимуществами в рамках, а именно:

– PostgreSQL имеет поддержку хранимых процедур, которые повысят скорость выполнения запросов, в отличие от MySQL;

– PostgreSQL является более производительным и имеет полную поддержку ACID (atomicity, consistency, isolation, durability) [2].

#### **3 Разработка функциональной структуры системы с учётом клиент-серверной архитектуры. Постановка задачи разработки**

Функциональная структура веб-приложения для анализа данных с данными и пользователях, подписках, постах и переписках представлена на рисунке 2. Функциональная структура представлена в виде модели клиент-сервер. Клиент может отправлять на сервер HTTP запросы Ruquest и получать от сервера HTTP ответы Response.

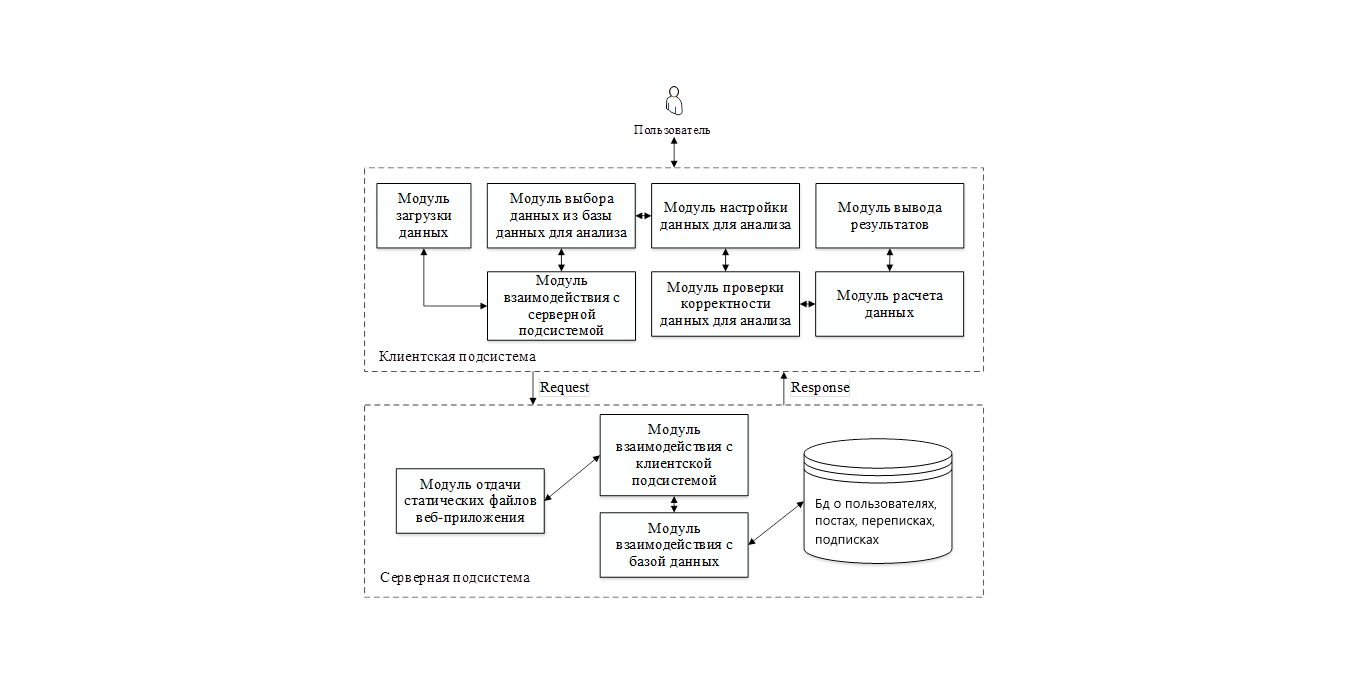


Рисунок 2 – Функциональная структура веб-приложения

Постановку задачи разработки:

Требуется разработать веб-приложение, позволяющее пользователю:

– загружать в приложение данные о измерениях толщины материнских рулонов и сохранять их в базу данных;

– по данным профилей толщины материнского рулона из базы данных о измерениях толщины материнских рулонов произвести расчёт индекса потенциала процесса Cp и индекса воспроизводимости процесса Cpk для материнского рулона или для рулона готовой продукции.

#### **4 Разработка UML-диаграммы вариантов использования программы**

На рисунке 3 представлена UML-диаграммы вариантов использования веб-приложения для пользователя.

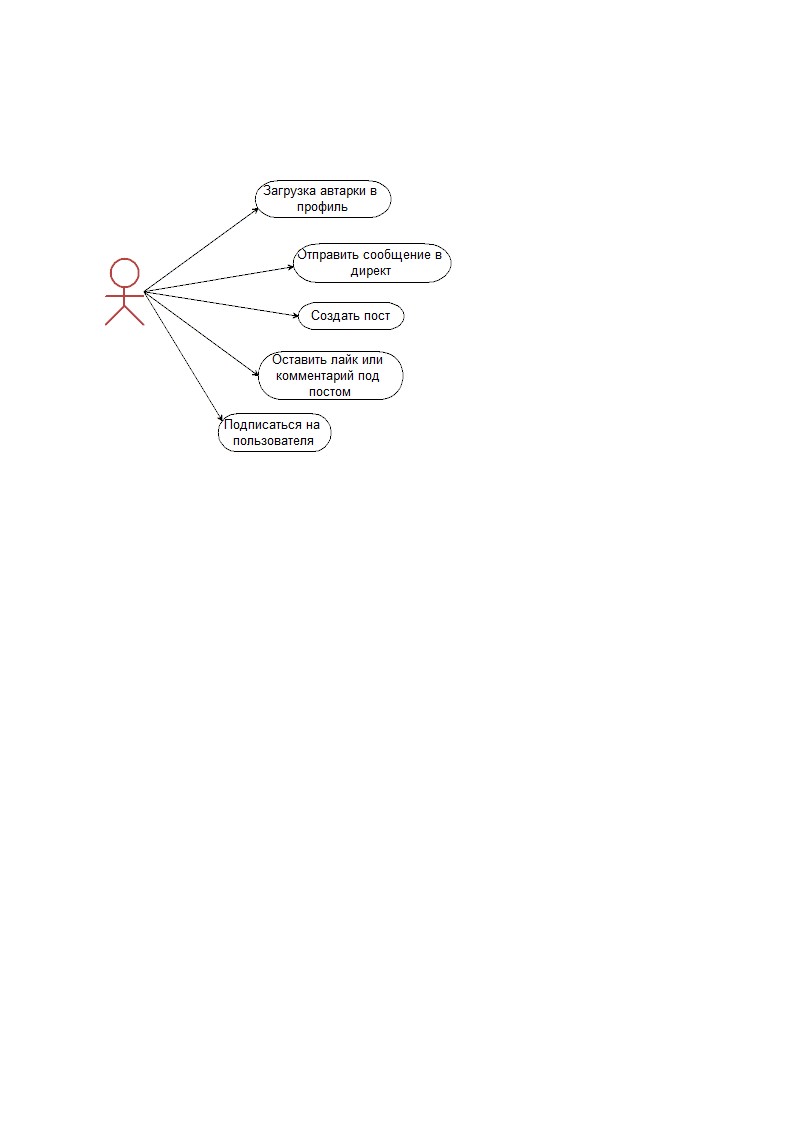


Рисунок 3 – UML-диаграммы вариантов использования веб-приложения

#### **5 Разработка блок-схемы алгоритма решения задачи**

На рисунке 4 представлена блок-схема алгоритма загрузки данных о профилях толщины в базу данных веб-приложения.

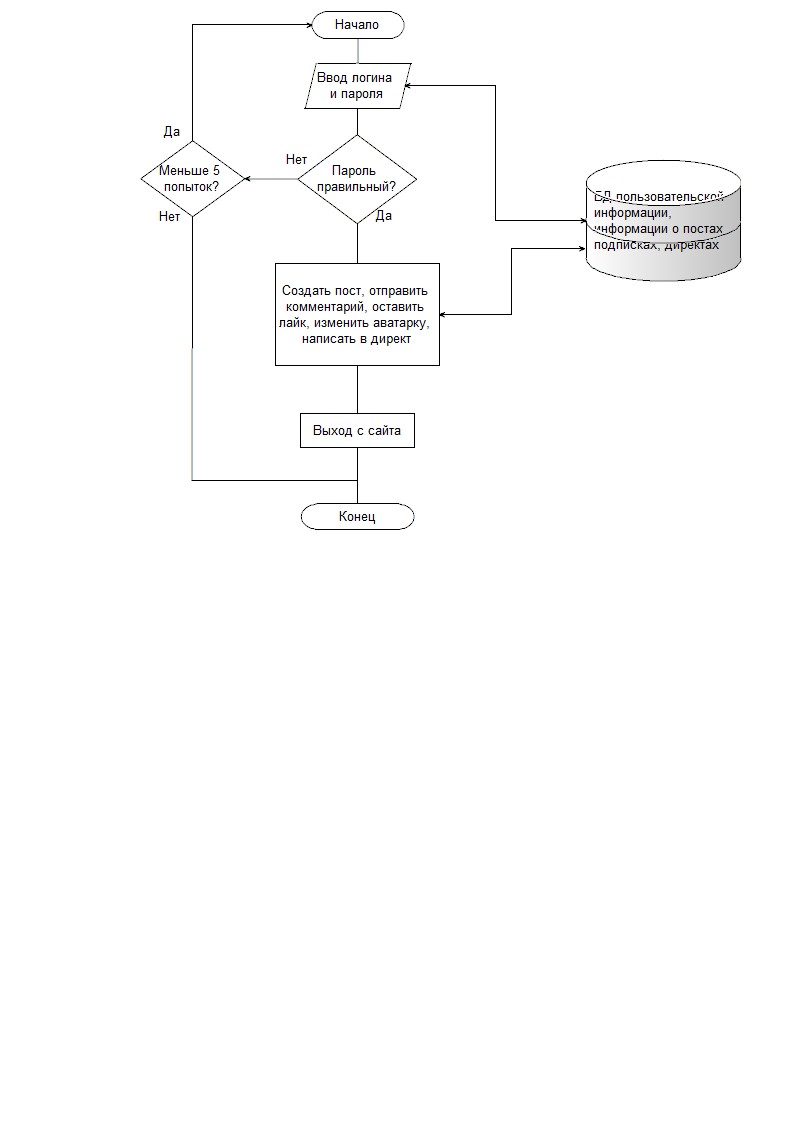


Рисунок 4 – Блок-схема алгоритма загрузки данных

#### **6 Проектирование серверного интерфейса взаимодействия с клиентом и СУБД**

В таблице 2 представлено описание API обращения клиента к серверу по основным запросам.

Таблица 2 – Описание API обращения клиента к серверу по основным запросам

|  |  |  |
| --- | --- | --- |
| Метод HTTP запроса | Относительный путь запроса | Назначение запроса |
| POST | /file | Создание новых данных о толщине материнского рулона в базе данных |
| GET | /filenames | Получение массива названий файлов и идентификаторов доступных для скачивания данных о толщине материнских рулонов |
| GET | /file/:id | Получение из базы данных документа с идентификатором id с описанием данных о толщине материнских рулонов |

Для взаимодействия с выбранной СУБД используется Entity Framework (EF), с помощью которого осуществляется чтение, обновление и создание записей в таблицах.

Далее приведен листинг сервера, который с СУБД.

[DirectService]

using AutoMapper;

using DDApp.API.Models.Direct;

using DDApp.Common.Exceptions;

using DDApp.Common.Exceptions.Forbidden;

using DDApp.Common.Exceptions.NotFound;

using DDApp.Common.Exceptions.UnprocessableEntity;

using DDApp.DAL;

using DDApp.DAL.Entites;

using DDApp.DAL.Entites.DirectDir;

using Microsoft.EntityFrameworkCore;

using DDApp.Common.Consts;

using DDApp.Common.Exceptions.Authorization.SpecificExceptions;

namespace DDApp.API.Services

{

public class DirectService

{

private readonly DataContext \_context;

private readonly IMapper \_mapper;

private readonly AttachService \_attachService;

private readonly GooglePushService \_pushService;

private readonly AttachService \_attchService;

public DirectService(DataContext context, IMapper mapper, AttachService attachService, GooglePushService pushService, AttachService attchService)

{

\_context = context;

\_mapper = mapper;

\_attachService = attachService;

\_pushService = pushService;

\_attchService = attchService;

}

/// <summary>

/// Создают директ группу

/// </summary>

public async Task CreateDirectGroup(CreateDirectGroupModel model, Guid userId)

{

model.Members.Add(userId);

if (model.Members == null || model.Members.Count < 2)

{

throw new MembresCountUnprocessableEntityException();

}

model.Members.ForEach(async u =>

{

if (!(await CheckUserExistById(u)))

{

throw new UserNotFoundException();

}

});

if(model.Title == default || model.Title.Length == 0)

{

throw new GroupTitleUnprocessableEntityException();

}

var direct = (await \_context.Directs.AddAsync(new Direct

{

DirectId = model.Id ?? Guid.NewGuid(),

DirectTitle = model.Title,

IsDirectGroup = true,

Created = model.Created ?? DateTimeOffset.UtcNow,

})).Entity;

if(model.GroupImage != null)

{

var filePath = \_attachService.CopyImageFile(model.GroupImage);

await \_context.DirectImages.AddAsync(new DirectImages

{

DirectId = direct.DirectId,

Author = \_context.Users.First(y => y.Id == userId),

Name = model.GroupImage.Name,

MimeType = model.GroupImage.MimeType,

FilePath = filePath,

Size = model.GroupImage.Size,

});

}

model.Members.ForEach(async x =>

{

await \_context.DirectMembers.AddAsync(new DirectMembers

{

DirectId = direct.DirectId,

UserId = x,

});

});

await \_context.SaveChangesAsync();

}

/// <summary>

/// Возвращает дерект пользователя

/// </summary>

public async Task<DirectModel?> GetUserDirect(Guid directId, Guid senderId)

{

var direct = await \_context.Directs

.AsNoTracking()

.Include(x => x.DirectMembers)

.Include(x => x.DirectMembers).ThenInclude(x => x.User)

.Include(x => x.DirectMembers).ThenInclude(x => x.User).ThenInclude(x => x.Avatar)

.Include(x => x.DirectImage)

.FirstOrDefaultAsync(x => x.DirectId == directId);

if (direct == null)

{

throw new DirectNotFoundException();

}

if (!(await CheckUserExistById(senderId)))

{

throw new UserNotFoundException();

}

var directMembers = await \_context.DirectMembers.Where(x => x.DirectId == directId).ToListAsync();

if (directMembers.FirstOrDefault(x => x.UserId == senderId) == null)

{

throw new AccessAuthorizationException();

}

return \_mapper.Map<DirectModel>(direct);

}

public async Task<List<DirectMessageModel>?> GetDirectMessage(Guid currentUser, Guid directId, int skip, int take, DateTimeOffset? lastDirectMessageCreated = null)

{

var direct = (await \_context.Directs

.AsNoTracking()

.Include(x => x.DirectMembers)

.Include(x => x.DirectMembers).ThenInclude(x => x.User)

.FirstOrDefaultAsync(x => x.DirectId == directId));

if (direct == null)

{

throw new DirectNotFoundException();

}

var directMembers = await \_context.DirectMembers.Where(x => x.DirectId == directId).ToListAsync();

if (directMembers.FirstOrDefault(x => x.UserId == currentUser) == null)

{

throw new AccessAuthorizationException();

}

return await \_context.DirectMessages

.AsNoTracking()

.Where(x => x.DirectId == directId && (lastDirectMessageCreated == null ? true : x.Sended > lastDirectMessageCreated))

.OrderByDescending(x => x.Sended)

.Skip(skip)

.Take(take)

.Include(x => x.Direct)

.Include(x => x.DirectFiles)

.Include(x => x.User)

.Select(x => \_mapper.Map<DirectMessages, DirectMessageModel>(x))

.ToListAsync();

}

public async Task<DirectModel?> GetDirectWithUser(Guid currentUserId, Guid userId)

{

var direct = await \_context.Directs

.AsNoTracking()

.Include(x => x.DirectImage)

.Include(x => x.DirectMembers)

.Include(x => x.DirectMembers).ThenInclude(x => x.User)

.Include(x => x.DirectMembers).ThenInclude(x => x.User).ThenInclude(x => x.Avatar)

.FirstOrDefaultAsync(x => x.DirectMembers

.Contains(new DirectMembers { DirectId = x.DirectId, UserId = currentUserId})

&& x.DirectMembers

.Contains(new DirectMembers { DirectId = x.DirectId, UserId = userId})

&& x.DirectMembers.Count == 2);

if(direct == null)

{

return null;

}

var member = await \_context.DirectMembers

.AsNoTracking()

.FirstOrDefaultAsync(x => x.DirectId == direct.DirectId && x.UserId == userId);

if (member != null)

{

var avatar = await \_context.Avatars

.AsNoTracking()

.FirstOrDefaultAsync(x => x.UserId == member.UserId);

if (avatar != null)

{

var res = \_mapper.Map<Avatar, DirectImageModel>(avatar);

var dirModel = \_mapper.Map<DirectModel>(direct);

dirModel.DirectImage = res;

return dirModel;

}

}

return \_mapper.Map<DirectModel>(direct);

}

/// <summary>

/// Возвращает список директов пользователя

/// </summary>

public async Task<List<DirectModel>?> GetUserDirects(Guid userId, int take, int skip)

{

var directs = await \_context.Directs

.AsNoTracking()

.Include(x => x.DirectImage)

.Include(x => x.DirectMembers)

.Include(x => x.DirectMembers).ThenInclude(x => x.User)

.Include(x => x.DirectMembers).ThenInclude(x => x.User).ThenInclude(x => x.Avatar)

.Where(x => x.DirectMembers.Contains(new DirectMembers { DirectId = x.DirectId, UserId = userId }))

.OrderByDescending(x => x.DirectTitle)

.Skip(skip)

.Take(take)

.Select(x => \_mapper.Map<Direct, DirectModel>(x))

.ToListAsync();

for (int i = 0; i < directs.Count; i++)

{

var member = await \_context.DirectMembers

.AsNoTracking()

.FirstOrDefaultAsync(x => x.DirectId == directs[i].DirectId && x.UserId != userId);

if(member != null)

{

var avatar = await \_context.Avatars

.AsNoTracking()

.FirstOrDefaultAsync(x => x.UserId == member.UserId);

if(avatar != null)

{

var res = \_mapper.Map<Avatar, DirectImageModel>(avatar);

directs[i].DirectImage = res;

}

}

}

return directs;

}

/// <summary>

/// Создает директ (один на один) с указанным пользователем

/// </summary>

public async Task CreateDirectWithUser(Guid currentUserId, CreateDirectModel model)

{

if (await CheckUserExistById(currentUserId) == false || await CheckUserExistById(model.UserId) == false)

{

throw new UserNotFoundException();

}

if((await \_context.Directs

.AsNoTracking()

.Include(x => x.DirectMembers)

.FirstOrDefaultAsync(x => x.DirectMembers.Count == 2

&& x.DirectMembers.Contains(new DirectMembers { UserId = currentUserId, DirectId = x.DirectId })

&& x.DirectMembers.Contains(new DirectMembers { UserId = model.UserId, DirectId = x.DirectId })))

!= null)

{

throw new DirectExistsForbiddenException();

}

var directId = model.Id ?? Guid.NewGuid();

var userAvatar = await \_context.Avatars

.AsNoTracking()

.Include(x => x.Author)

.FirstOrDefaultAsync(x => x.UserId == model.UserId);

DirectImages? directImage;

if (userAvatar != null)

{

var attach = await \_context.Attaches.AddAsync(new Attach

{

Author = await \_context.Users.FirstAsync(x => x.Id == currentUserId),

FilePath = userAvatar.FilePath,

Id = Guid.NewGuid(),

MimeType = userAvatar.MimeType,

Name = userAvatar.Name,

Size = userAvatar.Size,

});

directImage = new DirectImages

{

Author = attach.Entity.Author,

DirectId = directId,

FilePath = attach.Entity.FilePath,

Id = Guid.NewGuid(),

MimeType = attach.Entity.MimeType,

Name = attach.Entity.Name,

Size = attach.Entity.Size,

};

await \_context.Directs.AddAsync(new Direct

{

DirectId = directId,

DirectTitle = model.Title ?? (await \_context.Users.AsNoTracking().FirstOrDefaultAsync(x => x.Id == model.UserId))!.Name,

DirectImage = directImage,

});

}

else

{

var direct = await \_context.Directs.AddAsync(new Direct

{

DirectId = directId,

DirectTitle = model.Title ?? (await \_context.Users.AsNoTracking().FirstOrDefaultAsync(x => x.Id == model.UserId))!.Name,

});

}

\_context.DirectMembers.Add(new DirectMembers

{

DirectId = directId,

UserId = currentUserId,

});

\_context.DirectMembers.Add(new DirectMembers

{

DirectId = directId,

UserId = model.UserId,

});

await \_context.SaveChangesAsync();

}

/// <summary>

/// Отправляет сообщение в выбранный директ

/// </summary>

public async Task CreateDirectMessage(CreateDirectMessageModel model, Guid sender)

{

if((model.Files == null || model.Files?.Count == null) && (model.Message == null || model.Message == String.Empty))

{

throw new MessageOrFilesUnprocessableEntityException();

}

var message = (await \_context.DirectMessages.AddAsync(new DirectMessages

{

DirectMessageId = model.DirectMessageId ?? Guid.NewGuid(),

DirectId = model.DirectId,

DirectMessage = model.Message,

SenderId = sender,

Sended = model.Sended ?? DateTimeOffset.UtcNow,

})).Entity;

if (model.Files != null)

{

model.Files.ForEach(x =>

{

var filePath = \_attachService.CopyImageVideoFile(x);

\_context.DirectFiles.Add(new DirectFiles

{

DirectMessagesId = message.DirectMessageId,

Author = \_context.Users.First(y => y.Id == sender),

Name = x.Name,

MimeType = x.MimeType,

FilePath = filePath,

Size = x.Size,

});

});

}

await \_context.SaveChangesAsync();

}

private async Task<bool> CheckUserExistById(Guid userId)

{

var user = await \_context.Users.AsNoTracking().FirstOrDefaultAsync(x => x.Id == userId);

if (user == null || user.IsActive == false || user == default)

{

return false;

}

return true;

}

}

}

[DirectService]

[PostCommentService]

using AutoMapper;

using AutoMapper.QueryableExtensions;

using DDApp.API.Models;

using DDApp.Common.Exceptions;

using DDApp.Common.Exceptions.NotFound;

using DDApp.DAL;

using DDApp.DAL.Entites;

using Microsoft.EntityFrameworkCore;

using DDApp.Common.Exceptions.Authorization;

namespace DDApp.API.Services

{

public class PostCommentService

{

private readonly DataContext \_context;

private readonly IMapper \_mapper;

private readonly AttachService \_attachmentsService;

public PostCommentService(DataContext context, IMapper mapper, AttachService attachmentsService)

{

\_context = context;

\_mapper = mapper;

\_attachmentsService = attachmentsService;

}

/// <summary>

/// Возвращает все комментарии к определенному посту.

/// </summary>

public async Task<List<PostCommentModel>?> GetPostCommentsByPostId(DateTimeOffset? lastPostCreated, Guid postId, int take, int skip)

{

var post = await \_context.Posts.FirstOrDefaultAsync(x => x.Id == postId);

if (post == null || post == default)

{

throw new PostNotFoundException();

}

return await \_context.PostComments

.AsNoTracking()

.Where(x => x.Post.Id == postId && (lastPostCreated == null ? true : x.Created < lastPostCreated))

.OrderByDescending(x => x.Created)

.Skip(skip)

.Take(take)

.ProjectTo<PostCommentModel>(\_mapper.ConfigurationProvider)

.ToListAsync();

}

/// <summary>

/// Создает пост привязанный к определенному пользователю.

/// </summary>

public async Task CreatePostComment(Guid userId, CreatePostCommentModel model)

{

var user = await \_context.Users.FirstOrDefaultAsync(x => x.Id == userId);

if (user == null || user.IsActive == false)

{

throw new UserNotFoundException();

}

var post = await \_context.Posts.FirstOrDefaultAsync(x => x.Id == model.PostId);

if (post == null || post.IsActive == false)

{

throw new PostNotFoundException();

}

var comment = await \_context.PostComments.AddAsync(new PostComments

{

Id = model.Id,

Text = model.Text,

Created = model.Created,

Author = user,

Post = post,

});

await \_context.SaveChangesAsync();

}

/// <summary>

/// Удаляет комментарий к посту по его id

/// </summary>

public async Task DeletePostComment(Guid postCommentId, Guid userId)

{

var postComment = await \_context.PostComments

.Include(x => x.Author)

.Include(x => x.Post)

.ThenInclude(x => x.Author)

.FirstOrDefaultAsync(x => x.Id == postCommentId);

if(postComment == null || postComment == default || !postComment.IsActive)

{

throw new PostCommentNotFoundException();

}

if (!postComment.Post.IsActive)

{

throw new PostNotFoundException();

}

if(postComment.Author.Id != userId || postComment.Post.Author.Id != userId)

{

throw new UserAuthorizationException();

}

postComment.IsActive = false;

await \_context.SaveChangesAsync();

}

/// <summary>

/// Если лайк был он удаляется, если его не было, то он доавляется.

/// </summary>

public async Task ChangePostCommentLikeState(Guid postCommentId, Guid userId)

{

if((await \_context.PostComments.FirstOrDefaultAsync(x => x.Id == postCommentId)) == null)

{

throw new PostCommentNotFoundException();

}

var like = new PostCommentLikes

{

UserId = userId,

PostCommentId = postCommentId,

};

if (\_context.PostCommentsLikes.Contains(like))

{

\_context.PostCommentsLikes.Remove(like);

}

else

{

\_context.PostCommentsLikes.Add(like);

}

await \_context.SaveChangesAsync();

}

}

}

[PostCommentService]

[PostService]

using AutoMapper;

using AutoMapper.QueryableExtensions;

using DDApp.API.Models;

using DDApp.Common.Exceptions;

using DDApp.Common.Exceptions.NotFound;

using DDApp.DAL;

using DDApp.DAL.Entites;

using Microsoft.EntityFrameworkCore;

using DDApp.API.Models.Subscription;

using DDApp.Common.Exceptions.Authorization;

using FileNotFoundException = DDApp.Common.Exceptions.NotFound.FileNotFoundException;

namespace DDApp.API.Services

{

public class PostService

{

private readonly DataContext \_context;

private readonly IMapper \_mapper;

private readonly AttachService \_attachmentsService;

public PostService(DataContext context, IMapper mapper, AttachService attachmentsService)

{

\_context = context;

\_mapper = mapper;

\_attachmentsService = attachmentsService;

}

/// <summary>

/// Возвращает по идентефикатору.

/// </summary>

public async Task<PostModel> GetPost(Guid id)

{

var post = await \_context.Posts

.AsNoTracking()

.Include(x => x.Author).ThenInclude(x => x.Avatar)

.Include(x => x.PostFiles)

.Include(x => x.Comments)

.Include(x => x.PostLikes)

.Where(x => x.IsActive)

.FirstOrDefaultAsync(x => x.Id == id);

if (post == null || post.IsActive == false)

{

throw new PostNotFoundException();

}

return \_mapper.Map<PostModel>(post);

}

/// <summary>

/// Создает пост и прикрепляет к нему польщователя и изображения, последние

/// если имеются.

/// </summary>

public async Task CreatePost(Guid userId, CreatePostModel model)

{

var user = await \_context.Users.Include(x => x.Posts).FirstOrDefaultAsync(x => x.Id == userId);

if (user == null || user.IsActive == false)

{

throw new UserNotFoundException();

}

if(model.Files == null || model.Files.Count == 0)

{

throw new FileNotFoundException();

}

else

{

var filePath = string.Empty;

var post = await \_context.Posts.AddAsync(new Posts

{

Id = model.Id ?? Guid.NewGuid(),

Author = user,

Text = model.Text,

Created = model.Created ?? DateTimeOffset.UtcNow,

}) ;

if (model.Files != null)

{

foreach (var meta in model.Files)

{

filePath = \_attachmentsService.CopyImageVideoFile(meta);

var postFile = await \_context.PostFiles.AddAsync(new PostFiles

{

Id = meta.TempId ?? Guid.NewGuid(),

PostId = post.Entity.Id,

Name = meta.Name,

Author = user,

MimeType = meta.MimeType,

Size = meta.Size,

FilePath = filePath,

});

}

}

await \_context.SaveChangesAsync();

}

}

/// <summary>

/// Удаляет выбранный пост

/// </summary>

public async Task DeletePost(Guid postId, Guid userId)

{

var post = await \_context.Posts.Include(x => x.Author).FirstOrDefaultAsync(x => x.Id == postId);

if(post == default || post == null || post.IsActive == false)

{

throw new PostNotFoundException();

}

if(post.Author.Id != userId)

{

throw new UserAuthorizationException();

}

post.IsActive = false;

await \_context.SaveChangesAsync();

}

public async Task<int> GetUserPostAmount(Guid userId)

=> await \_context.Posts.CountAsync(x => x.Author.Id == userId && x.IsActive == true);

/// <summary>

/// Возвращает посты, созданные подписками данного пользователя

/// </summary>

public async Task<List<PostModel>?> GetSubscriptionsPosts(Guid userId, int skip, int take, DateTimeOffset? lastPostCreated)

{

var subscriptions = await \_context.Subscriptions

.AsNoTracking()

.Where(x => x.SubscriberId == userId && x.UserSubscription.IsActive)

.Select(x => \_mapper.Map<Subscriptions, OnlySubscriptionModel>(x))

.ToListAsync();

List<Guid> subscriptionsGuids = new List<Guid>();

foreach(var sub in subscriptions)

{

subscriptionsGuids.Add(sub.SubscriptionId);

}

var posts = await \_context.Posts

.AsNoTracking()

.Where(x => x.IsActive && subscriptionsGuids.Contains(x.Author.Id) && (lastPostCreated == null ? true : x.Created < lastPostCreated))

.OrderByDescending(x => x.Created)

.Skip(skip)

.Take(take)

.Include(x => x.PostFiles)

.Include(x => x.Comments)

.Include(x => x.PostLikes)

.Include(x => x.Author)

.Include(x => x.Author).ThenInclude(x => x.Avatar)

.Select(x => \_mapper.Map<Posts, PostModel>(x))

.ToListAsync();

return posts;

}

/// <summary>

/// Возвращает посты текущего пользвателя.

/// </summary>

public async Task<List<PostModel>> GetCurrentUserPosts(DateTimeOffset? lastPostCreated, int skip, int take, Guid userId)

{

var posts = await \_context.Posts

.AsNoTracking()

.Where(x => x.IsActive && x.Author.Id == userId && (lastPostCreated == null ? true : x.Created > lastPostCreated))

.OrderByDescending(x => x.Created)

.Skip(skip)

.Take(take)

.Include(x => x.PostFiles)

.Include(x => x.Comments)

.Include(x => x.PostLikes)

.Include(x => x.Author)

.Include(x => x.Author).ThenInclude(x => x.Avatar)

.Select(x => \_mapper.Map<Posts, PostModel>(x))

.ToListAsync();

if (posts == null || posts == default)

{

throw new PostNotFoundException();

}

return posts;

}

/// <summary>

/// Возвращает все посты.

/// </summary>

public async Task<List<PostModel>> GetPosts(DateTimeOffset? lastPostCreated, int skip, int take, Guid userId) {

var user = await \_context.Users.AsNoTracking().Include(x => x.Posts).FirstOrDefaultAsync(x => x.Id == userId);

if(user == default || user == null || user.IsActive == false)

{

throw new UserNotFoundException();

}

var posts = await \_context.Posts

.AsNoTracking()

.Where(x => x.IsActive && x.Author.Id == userId && (lastPostCreated == null ? true : x.Created > lastPostCreated))

.OrderByDescending(x => x.Created)

.Skip(skip)

.Take(take)

.Include(x => x.PostFiles)

.Include(x => x.Comments)

.Include(x => x.PostLikes)

.Include(x => x.Author)

.Include(x => x.Author).ThenInclude(x => x.Avatar)

.Select(x => \_mapper.Map<Posts, PostModel>(x))

.ToListAsync();

if(posts == null || posts == default)

{

throw new PostNotFoundException();

}

return posts;

}

/// <summary>

/// Если лайк на посте был, то он удаляется,

/// в противном случае добавляется.

/// </summary>

public async Task ChangePostLikeState(Guid postId, Guid userId)

{

var like = new PostLikes

{

UserId = userId,

PostId = postId,

};

if (\_context.PostLikes.Contains(like))

{

\_context.PostLikes.Remove(like);

}

else

{

\_context.PostLikes.Add(like);

}

await \_context.SaveChangesAsync();

}

public async Task<bool> GetPostLikeState(Guid postId, Guid userId)

{

if (await \_context.PostLikes.FirstOrDefaultAsync(x => x.PostId == postId && x.UserId == userId) == null)

{

return false;

}

return true;

}

}

}

[PostService]

[SubscriptionService]

using AutoMapper;

using DDApp.API.Models.Subscription;

using DDApp.Common.Exceptions;

using DDApp.Common.Exceptions.Forbidden;

using DDApp.DAL;

using DDApp.DAL.Entites;

using Microsoft.EntityFrameworkCore;

namespace DDApp.API.Services

{

public class SubscriptionService

{

private readonly DataContext \_context;

private readonly IMapper \_mapper;

public SubscriptionService(DataContext context, IMapper mapper)

{

\_context = context;

\_mapper = mapper;

}

/// <summary>

/// Получить всех подписчиков пользователя

/// </summary>

public async Task<List<SubscriberModel>> GetSubscribers(Guid userId, int skip, int take)

{

if(!(await CheckUserExistById(userId)))

{

throw new UserNotFoundException();

}

return await \_context.Subscriptions

.AsNoTracking()

.Skip(skip)

.Take(take)

.Where(x => x.SubscriptionId == userId)

.Include(x => x.UserSubscriber).ThenInclude(x => x.Avatar)

.Include(x => x.UserSubscriber).ThenInclude(x => x.Session)

.Select(x => \_mapper.Map<SubscriberModel>(x))

.ToListAsync();

}

/// <summary>

/// Подписаться или отписать от пользователя в зависимости от текущего состояния

/// </summary>

public async Task ChangeSubscriptionStateOnUserById(Guid subscriberId, Guid subscriptionId)

{

if(subscriberId == subscriptionId)

{

throw new UserSubscriptionForbiddentException();

}

if(await CheckUserExistById(subscriberId) == false || await CheckUserExistById(subscriptionId) == false)

{

throw new UserNotFoundException();

}

var subscription = new Subscriptions

{

SubscriberId = subscriberId,

SubscriptionId = subscriptionId,

};

if (\_context.Subscriptions.Contains(subscription))

{

\_context.Subscriptions.Remove(subscription);

}

else

{

await \_context.Subscriptions.AddAsync(subscription);

}

await \_context.SaveChangesAsync();

}

/// <summary>

/// Возвращает все подписки пользвователя

/// </summary>

public async Task<List<SubscriptionModel>> GetSubscriptions(Guid userId, int skip, int take)

{

if (!(await CheckUserExistById(userId)))

{

throw new UserNotFoundException();

}

return await \_context.Subscriptions

.AsNoTracking()

.Skip(skip)

.Take(take)

.Where(x => x.SubscriberId == userId)

.Include(x => x.UserSubscription).ThenInclude(x => x.Avatar)

.Include(x => x.UserSubscription).ThenInclude(x => x.Session)

.Select(x => \_mapper.Map<SubscriptionModel>(x))

.ToListAsync();

}

public async Task<int> GetUserSubscribersAmount(Guid userId)

=> await \_context.Subscriptions.CountAsync(x => x.UserSubscription.Id == userId && x.UserSubscription.IsActive == true);

public async Task<int> GetUserSubscriptionsAmount(Guid userId)

=> await \_context.Subscriptions.CountAsync(x => x.UserSubscriber.Id == userId && x.UserSubscriber.IsActive == true);

public async Task<bool> IsSubscribedOn(Guid userId, Guid currentUserId)

{

if (!(await CheckUserExistById(userId)) || !(await CheckUserExistById(currentUserId)))

{

throw new UserNotFoundException();

}

var res = await \_context.Subscriptions

.AsNoTracking()

.FirstOrDefaultAsync(x => x.SubscriptionId == userId

&& x.SubscriberId == currentUserId);

if (res == null)

{

return false;

}

else

{

return true;

}

}

private async Task<bool> CheckUserExistById(Guid userId)

{

var user = await \_context.Users.AsNoTracking().FirstOrDefaultAsync(x => x.Id == userId);

if (user == null || user.IsActive == false || user == default)

{

return false;

}

return true;

}

}

}

[SubscriptionService]

[UserService]

using AutoMapper;

using DDApp.API.Models;

using DDApp.DAL;

using Microsoft.EntityFrameworkCore;

using DDApp.DAL.Entites;

using DDApp.Common.Exceptions;

using DDApp.Common.Exceptions.NotFound;

using DDApp.Common.Exceptions.Forbidden;

namespace DDApp.API.Services

{

public class UserService

{

private readonly IMapper \_mapper;

private readonly DataContext \_context;

private readonly AttachService \_attachService;

public UserService(IMapper mapper, DataContext context,

AttachService attachService)

{

\_mapper = mapper;

\_context = context;

\_attachService = attachService;

}

public async Task CreateUser(CreateUserModel model)

{

if (await CheckUserExist(model.Email))

{

throw new UserCreationForbiddenException();

}

var dbUser = \_mapper.Map<User>(model);

await \_context.Users.AddAsync(dbUser);

await \_context.SaveChangesAsync();

}

public async Task Delete(Guid userId)

{

var user = await GetUserById(userId);

if(user != null)

{

user.IsActive = false;

await \_context.SaveChangesAsync();

}

}

public async Task<List<UserWithLinkModel>?> GetUsers(int skip, int take, Guid currentUserId)

=> await \_context.Users

.AsNoTracking()

.Where(x => x.IsActive && x.Id != currentUserId)

.OrderByDescending(x => x.Name)

.Skip(skip)

.Take(take)

.Include(x => x.Avatar)

.Select(x => \_mapper.Map<User, UserWithLinkModel>(x))

.ToListAsync();

public async Task<UserWithLinkModel> GetUser(Guid id)

=>\_mapper.Map<UserWithLinkModel>(await GetUserById(id));

public async Task AddAvatarToUser(Guid userId, MetadataModel meta)

{

var user = await GetUserById(userId);

var filePath = \_attachService.CopyImageFile(meta);

var avatar = new Avatar

{

User = user,

Author = user,

MimeType = meta.MimeType,

FilePath = filePath,

Size = meta.Size,

Name = meta.Name,

};

user.Avatar = avatar;

await \_context.SaveChangesAsync();

}

public async Task SetPushToken(Guid userId, string? token = null)

{

var user = await GetUserById(userId);

user.PushToken = token;

await \_context.SaveChangesAsync();

}

public async Task<string?> GetPushToken(Guid userId)

{

var user = await GetUserById(userId);

return user.PushToken;

}

private async Task<bool> CheckUserExist(string email)

{

var user = await \_context.Users.AsNoTracking().FirstOrDefaultAsync(x => x.Email.ToLower() == email.ToLower());

if (user == null || user.IsActive == false)

{

return false;

}

return true;

}

private async Task<User> GetUserById(Guid id)

{

var user = await \_context.Users.Include(x => x.Avatar).FirstOrDefaultAsync(x => x.Id == id);

if (user == null || user.IsActive == false || user == default)

{

throw new UserNotFoundException();

}

return user;

}

}

}

[UserService]

#### **7 Разработка базы данных информационной системы**

Информационное обеспечение состоит из базы данных DDAppdb (о измерения толщины материнских рулонов). Внутри базы данных есть следующие сущности Attaches, Avatars, DirectMembers, DirectMessages, Directs, PostComments, PostLikes, Posts, Subscriptions, Users.

Структура документа представлена в таблицах 4.

Таблица 4 – Структура сущностей

|  |  |  |
| --- | --- | --- |
| Поле(я) | Тип данных | Значение |
| Attaches( Id, Name, MimeType, FilePath, Size, AuthorId) | uuid, text, text, text, bigint, uuid | Id файла, Название файла, тип файла, путь, размер, id автора |
| Avatars( Id, UserId) | uuid, uuid | Id файла аватарки, id пользователя |
| DirectMembers(Id, UserId) | uuid, uuid | Id директа, id пользователя |
| DirectMessages( DirectMessageId, DirectId, DirectMessage, Sended, SenderId) | uuid, uuid, text, timestamp with time zone, uuid | Id сообщения в директе, id директа, сообщение, дата отправки сообщения, id отправителя |
| Directs( DirectId, DirectTitle, IsDirectGroup, Created) | uuid, text, boolean, timestamp with time zone | Id директа, название директа, является ли директ группой, дата создания директа |
| PostComments( Id, Text, Created, IsActive, PostId, AuthorId) | uuid, text, timestamp with time zone, boolean, uuid, uuid | Id комментария под постом, дата создания комментария, активен ли комментарий, id поста, id отправителя |
| PostLikes (PostId, UserId) | uuid, uuid | Id поста, Id пользователя |
| Posts (Id, Created, Text, IsActive, AuthorId) | uuid, timestamp with time zone, text, boolean, uuid | Id поста, когда пост был создан, текст поста, активен ли пост, id автора поста |
| Subscriptions (SubscriptionId, SubscriberId) | uuid, uuid | Id подписки, id подписчика |
| Users (id, name, email, passwordHash, isActive) | uuid, text, text, text, boolean | Id пользователя, имя пользователя, email пользователя, пароль, активен ли пользователь |

#### **8 Реализация с использованием JS проекта**

Далее представлен листинг программы, где реализуется обращение к серверу за данными с помощью JS.

[auth.html]

const baseUri = 'http://localhost';

async function Register() {

let date = new Date();

console.log(date.toJSON());

await fetch(baseUri + "/api/User/CreateUser", {

method: 'POST',

headers: {

'Accept': 'application/json, text/plain',

'Content-Type': 'application/json;charset=UTF-8',

},

body: JSON.stringify({

name: $("#email-field").val(),

password: $("#password-field").val(),

retryPassword: $("#password-field").val(),

email: $("#email-field").val(),

birthDate: date.toJSON()

}

),

}).then(data => {

if (!data.ok) {

if (data.status == 404) {

alert("wrong cridentials");

} else {

alert("something went wrong");

}

} else {

fetch(baseUri + "/api/Auth/Token", {

method: 'POST',

headers: {

'Accept': 'application/json, text/plain',

'Content-Type': 'application/json;charset=UTF-8',

},

body: JSON.stringify({

login: $("#email-field").val(),

password: $("#password-field").val(),

}),

})

.then(data => {

if (!data.ok) {

if (data.status == 404) {

alert("wrong cridentials");

} else {

alert("something went wrong");

}

} else {

return data.json();

}

}).then(async data => {

localStorage.setItem("accessToken", data.accessToken);

localStorage.setItem("refreshToken", data.refreshToken);

localStorage.setItem("uri", baseUri);

await getCurrentUser();

window.location.replace('my\_page.html');

}).catch(err => { throw "err" });

}

});

}

async function changeSubscribeState() {

const item = {

login: $("#email-field").val(),

password: $("#password-field").val(),

};

fetch(baseUri + "/api/Auth/Token", {

method: 'POST',

headers: {

'Accept': 'application/json, text/plain',

'Content-Type': 'application/json;charset=UTF-8',

},

body: JSON.stringify(item),

})

.then(data => {

if (!data.ok) {

if (data.status == 404) {

alert("wrong cridentials");

} else {

alert("something went wrong");

}

} else {

return data.json();

}

}).then(async data => {

localStorage.setItem("accessToken", data.accessToken);

localStorage.setItem("refreshToken", data.refreshToken);

localStorage.setItem("uri", baseUri);

await getCurrentUser();

window.location.replace('my\_page.html');

}).catch(err => { throw "err" });

}

async function getCurrentUser() {

fetch(baseUri + "/api/User/GetCurrentUser", {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(user => {

console.log(user.id);

localStorage.setItem('userId', user.id);

}).catch(err => { throw "err" });

}

[auth.html]

[change\_user\_avatar.html]

async function sendFiles() {

if (images.length == 0) {

alert("Add some files");

return;

}

var data = new FormData()

for (const file of images) {

data.append('files', file, file.type)

}

await fetch(localStorage.getItem("uri") + "/api/Attach/UploadFiles", {

method: 'POST',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'text/plain',

// 'Content-Type': 'multipart/form-data',

},

body: data,

}).then(response => response.json()).then(async metadata => {

console.log(metadata[0]);

await fetch(localStorage.getItem("uri") + "/api/User/AddAvatarToUser", {

method: 'POST',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'application/json;charset=UTF-8',

},

body: JSON.stringify(metadata[0]),

}).catch(response => alert("Data is not a picture")).then(response => window.location.replace('my\_page.html'));

});

}

[change\_user\_avatar.html]

[create\_post.html]

async function sendFiles() {

if (images.length == 0) {

alert("Add some files");

return;

}

var data = new FormData()

for (const file of images) {

data.append('files', file, file.type)

}

// console.log(data);

await fetch(localStorage.getItem("uri") + "/api/Attach/UploadFiles", {

method: 'POST',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'text/plain',

// 'Content-Type': 'multipart/form-data',

},

body: data,

}).then(response => response.json()).then(async metadata => {

await fetch(localStorage.getItem("uri") + "/api/Post/CreatePost", {

method: 'POST',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'application/json;charset=UTF-8',

},

body: JSON.stringify({

"text": $("#create-post-textarea").val(),

"files": metadata

}),

}).catch(response => alert("Data is not a picture")).then(response => window.location.replace('my\_page.html'));

});

}

[creare\_post.html]

[direct.html]

async function getDirects() {

await fetch(localStorage.getItem("uri") + "/api/Direct/GetUserDirects?" + new URLSearchParams({

skip: 0,

take: 1000,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(async directs => {

for (const direct of directs) {

let image;

if (direct.directImage == null) {

image = await getAuthorAvatar(direct.directMembers[0].directMember);

if (image == null) {

image = 'http://i.stack.imgur.com/Dj7eP.jpg';

}

else {

image = localStorage.getItem("uri") + image;

}

}

else {

image = localStorage.getItem("uri") + direct.directImage.link;

}

$('.user-directs-ul').append(

'<li class="direct-row">' +

'<div class="particular-direct">' +

` <div id="avatar">` +

' <img class="avatar-picture" src="' + image + '">' +

' </div>' +

' <div class="direct-button">' +

` <button class="to-profile" onclick='goToDirect("${direct.directId}")'>Direct</button>` +

' </div>' +

' <div class="direct-last-message">' +

await getDirectMessage(direct.directId) +

' </div>' +

' </div>' +

' </li>'

);

}

});

}

function goToDirect(directId) {

localStorage.setItem("directId", directId);

window.location.replace('particular\_direct.html');

}

async function getDirectMessage(directId) {

let res;

await fetch(localStorage.getItem("uri") + "/api/Direct/GetDirectMessage?" + new URLSearchParams({

directId: directId,

skip: 0,

take: 1,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(async directMessage => {

if (directMessage.length == 0) {

res = "";

}

else {

res = directMessage[0].directMessage;

}

});

return res;

}

async function getAuthorAvatar(userId) {

let res;

await fetch(localStorage.getItem("uri") + "/api/User/GetUser?" + new URLSearchParams({

userId: userId,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(data => { res = data.avatar });

return res;

}

[direct.html]

[my\_page.html]

async function getCurrentUser() {

fetch(localStorage.getItem("uri") + "/api/User/GetCurrentUser", {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(user => {

localStorage.setItem('userId', user.id);

});

}

async function getUser(userId) {

await fetch(localStorage.getItem("uri") + "/api/User/GetUser?" + new URLSearchParams({

userId: userId,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(user => {

let avatar;

if (user.avatar == null) {

avatar = 'http://i.stack.imgur.com/Dj7eP.jpg';

}

else {

avatar = localStorage.getItem("uri") + user.avatar;

}

if (user.id == localStorage.getItem("userId")) {

$('.' + "post-creator").append(

'<ul>' +

'<li>' +

`<div id="avatar" onclick='gotoChangePicture()'>` +

'<img class="avatar-picture" src="' + avatar + '">' +

'</div>' +

'</li>' +

'<li class="nickname-field">' +

'<div class="nickname">' + user.name + ' </div>' +

'</li>' +

'</ul>'

);

} else {

$('.' + "post-creator").append(

'<ul>' +

'<li>' +

`<div id="avatar" onclick='goToProfile("${user.id}")'>` +

'<img class="avatar-picture" src="' + avatar + '">' +

'</div>' +

'</li>' +

'<li class="nickname-field">' +

'<div class="nickname">' + user.name + ' </div>' +

'</li>' +

'</ul>'

);

}

});

}

async function getUserPosts() {

await fetch(localStorage.getItem("uri") + "/api/Post/GetPosts?" + new URLSearchParams({

skip: 0,

take: 10,

userId: localStorage.getItem("userId"),

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(data => {

for (const userPost of data) {

$('.user-posts').append(

`<div class="user-post-picture" onclick='goToPost("${userPost.id}")'>` +

'<img class="user-post-picture-img"' +

'src="' + localStorage.getItem("uri") + userPost.postFiles[0].link + '">' +

'</div>'

);

}

localStorage.getItem("userId", null);

});

}

function gotoChangePicture() {

window.location.replace('change\_user\_avatar.html');

}

[my\_page.html]

[particular\_direct.html]

async function sendMessage() {

await fetch(localStorage.getItem("uri") + "/api/Direct/CreateDirectMessage", {

method: 'POST',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'application/json;charset=UTF-8',

},

body: JSON.stringify({

directId: localStorage.getItem('directId'),

message: $(".direct-textarea").val()

}),

});

$(".direct-messages").prepend(

`<li class='direct-message-my'>${$(".direct-textarea").val()}</li>`

);

}

async function getCurrentUser() {

fetch(localStorage.getItem("uri") + "/api/User/GetCurrentUser", {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(user => {

localStorage.setItem('userId', user.id);

});

}

async function getMessages() {

let curUserId;

await getCurrentUser();

await fetch(localStorage.getItem("uri") + "/api/Direct/GetDirectMessage?" + new URLSearchParams({

directId: localStorage.getItem('directId'),

take: 1000,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

})

.then(response => response.json()).then(messages => {

for (const message of messages) {

if (localStorage.getItem("userId") == message.senderId) {

$(".direct-messages").append(

`<li class='direct-message-my'>${message.directMessage}</li>`

);

}

else {

$(".direct-messages").append(

`<li class='direct-message-other'>${message.directMessage}</li>`

);

}

}

});

}

[particular\_direct.html]

[post.html]

async function getSubsriptionPost() {

fetch(localStorage.getItem("uri") + "/api/Post/GetPost?" + new URLSearchParams({

postId: localStorage.getItem('postId'),

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(post => {

drawPost(post);

});

}

async function getPostComments(index, postId) {

let res = '';

await fetch(localStorage.getItem("uri") + "/api/PostComment/GetPostComments?" + new URLSearchParams({

// lastPostCreated: null,

postId: postId,

skip: 0,

take: 10,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(async postComments => {

//console.log(postComments);

res += '<div class="users-comments-div" id="user-comments-div' + index + '">';

for (const postComment of postComments) {

res += '<div id="user-created-comment" class="user-created-comment">' +

`<div id="avatar" onclick='goToProfile("${postComment.authorId}")'>` +

'<img class="avatar-picture" src="' + localStorage.getItem("uri") + await getAuthorAvatar(postComment.authorId) + '">' +

'</div>' +

'<div class="comment">' +

postComment.text +

'</div>' +

'</div>'

;

}

res += '</div>';

});

return res;

}

async function getAuthorAvatar(userId) {

let res;

await fetch(localStorage.getItem("uri") + "/api/User/GetUser?" + new URLSearchParams({

userId: userId,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(data => { res = data.avatar });

return res;

}

async function addPostCreator(authorId) {

res = '';

await fetch(localStorage.getItem("uri") + "/api/User/GetUser?" + new URLSearchParams({

userId: authorId,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

})

.then(response => response.json()).then(data => {

res =

`<div class="post-creator" onclick='goToProfile("${data.id}")'><ul>` +

'<li>' +

'<div id="avatar">' +

'<img class="avatar-picture" src="' + localStorage.getItem("uri") + data.avatar + '">' +

'</div>' +

'</li>' +

'<li class="nickname-field">' +

'<div class="nickname">' + data.name + ' </div>' +

'</li>' +

'</ul></div>';

});

return res;

}

function goToProfile(userId) {

console.log(userId);

localStorage.setItem("userId", userId);

window.location.replace('my\_page.html');

}

async function addLike(element, postId) {

await fetch(localStorage.getItem("uri") + "/api/Post/ChangePostLikeState?" + new URLSearchParams({

postId: postId,

}), {

method: 'POST',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json',

'Content-Type': 'application/json',

},

});

await fetch(localStorage.getItem("uri") + "/api/Post/GetPostLikeState?" + new URLSearchParams({

postId: postId,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json',

'Content-Type': 'application/json',

},

}).then(response => response.json()).then(data => {

if (data == true) {

$('#' + element).html(parseInt(document.getElementById(element).innerText) + 1);

}

else {

$('#' + element).html(parseInt(document.getElementById(element).innerText) - 1);

}

});

// $('#' + element).html(parseInt(document.getElementById(element).innerText) + 1);

}

async function sendComment(postId, commentId, commentsId, realPostId) {

console.log(postId);

await fetch(localStorage.getItem("uri") + "/api/PostComment/CreatePostComment", {

method: 'POST',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json',

'Content-Type': 'application/json',

},

body: JSON.stringify({

"text": $("#" + commentId).val(),

"postId": realPostId

})

});

$('#' + postId).append('<div id="user-created-comment" class="user-created-comment"><div id="avatar"><img class="avatar-picture"'

+ 'src="http://i.stack.imgur.com/Dj7eP.jpg">'

+ '</div> <div class="comment"> '

+ $("#" + commentId).val() + ' </div></div>');

$('#' + commentsId).html(parseInt(document.getElementById(commentsId).innerText) + 1);

}

[post.html]

[posts.html]

async function getSubsriptionPosts() {

fetch(localStorage.getItem("uri") + "/api/Post/GetSubscriptionPosts?" + new URLSearchParams({

// lastPostCreated: null,

skip: 0,

take: 1000,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(posts => {

drawPost(posts);

});

}

async function getPostComments(index, postId) {

let res = '';

await fetch(localStorage.getItem("uri") + "/api/PostComment/GetPostComments?" + new URLSearchParams({

// lastPostCreated: null,

postId: postId,

skip: 0,

take: 10,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(async postComments => {

//console.log(postComments);

res += '<div class="users-comments-div" id="user-comments-div' + index + '">';

for (const postComment of postComments) {

res += `<div id="user-created-comment" class="user-created-comment" onclick='goToProfile("${postComment.authorId}")'>` +

`<div id="avatar" >` +

'<img class="avatar-picture" src="' + localStorage.getItem("uri") + await getAuthorAvatar(postComment.authorId) + '">' +

'</div>' +

'<div class="comment">' +

postComment.text +

'</div>' +

'</div>'

;

}

res += '</div>';

});

return res;

}

async function getAuthorAvatar(userId) {

let res;

await fetch(localStorage.getItem("uri") + "/api/User/GetUser?" + new URLSearchParams({

userId: userId,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(data => { res = data.avatar });

return res;

}

async function addPostCreator(authorId) {

res = '';

await fetch(localStorage.getItem("uri") + "/api/User/GetUser?" + new URLSearchParams({

userId: authorId,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

})

.then(response => response.json()).then(data => {

res =

`<div class="post-creator" onclick='goToProfile("${data.id}")'><ul>` +

'<li>' +

'<div id="avatar">' +

'<img class="avatar-picture" src="' + localStorage.getItem("uri") + data.avatar + '">' +

'</div>' +

'</li>' +

'<li class="nickname-field">' +

'<div class="nickname">' + data.name + ' </div>' +

'</li>' +

'</ul></div>';

});

return res;

}

async function addLike(element, postId) {

await fetch(localStorage.getItem("uri") + "/api/Post/ChangePostLikeState?" + new URLSearchParams({

postId: postId,

}), {

method: 'POST',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json',

'Content-Type': 'application/json',

},

});

await fetch(localStorage.getItem("uri") + "/api/Post/GetPostLikeState?" + new URLSearchParams({

postId: postId,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json',

'Content-Type': 'application/json',

},

}).then(response => response.json()).then(data => {

if (data == true) {

$('#' + element).html(parseInt(document.getElementById(element).innerText) + 1);

}

else {

$('#' + element).html(parseInt(document.getElementById(element).innerText) - 1);

}

});

}

async function sendComment(postId, commentId, commentsId, realPostId) {

console.log(postId);

await fetch(localStorage.getItem("uri") + "/api/PostComment/CreatePostComment", {

method: 'POST',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json',

'Content-Type': 'application/json',

},

body: JSON.stringify({

"text": $("#" + commentId).val(),

"postId": realPostId

})

});

$('#' + postId).append('<div id="user-created-comment" class="user-created-comment"><div id="avatar"><img class="avatar-picture"'

+ 'src="http://i.stack.imgur.com/Dj7eP.jpg">'

+ '</div> <div class="comment"> '

+ $("#" + commentId).val() + ' </div></div>');

$('#' + commentsId).html(parseInt(document.getElementById(commentsId).innerText) + 1);

}

[posts.html]

[users.html]

async function getUsers() {

fetch(localStorage.getItem("uri") + "/api/User/GetUsers?" + new URLSearchParams({

skip: 0,

take: 1000,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(async users => {

let i = 0;

for (const user of users) {

let avatar = user.avatar;

if (avatar == null) {

avatar = 'http://i.stack.imgur.com/Dj7eP.jpg';

}

else {

avatar = localStorage.getItem("uri") + avatar;

}

$('.user-directs-ul').append(

'<li>' +

' <div class="particular-direct">' +

` <div id="avatar" onclick='goToProfile("${user.id}")'>` +

' <img class="avatar-picture" src="' + avatar + '">' +

' </div>' +

' <div class="user-button">' +

' <button id="user-subscribe-button' + i + '" ' +

` onclick='changeSubscribeState("user-subscribe-button${i}", "${user.id}")'>${await getSubscriptionState(user.id)}</button>` +

' </div>' +

`<div class="direct-last-message">${user.name} </div>` +

' </div>' +

' </li>'

);

i++;

}

});

}

async function changeSubscribeState(element, userId) {

if (document.getElementById(element).innerText == 'Subscribe') {

$('#' + element).html("Unsubsribe");

}

else {

$('#' + element).html("Subscribe");

}

await fetch(localStorage.getItem("uri") + "/api/Subscription/ChangeSubscriptionStateOnUser?" + new URLSearchParams({

userId: userId,

}), {

method: 'POST',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

});

}

async function getSubscriptionState(userId) {

let res = '';

await fetch(localStorage.getItem("uri") + "/api/Subscription/IsSubscribedOn?" + new URLSearchParams({

userId: userId,

}), {

method: 'GET',

headers: {

'Authorization': 'Bearer ' + localStorage.getItem('accessToken'),

'Accept': 'application/json, text/plain',

'Content-Type': 'text/plain;charset=UTF-8',

},

}).then(response => response.json()).then(users => {

if (users == true) {

res = 'Unsubsribe';

}

else {

res = 'Subscribe';

}

});

return res;

}

[users.html]

На рисунках 6-10 представлены результаты работы сайт

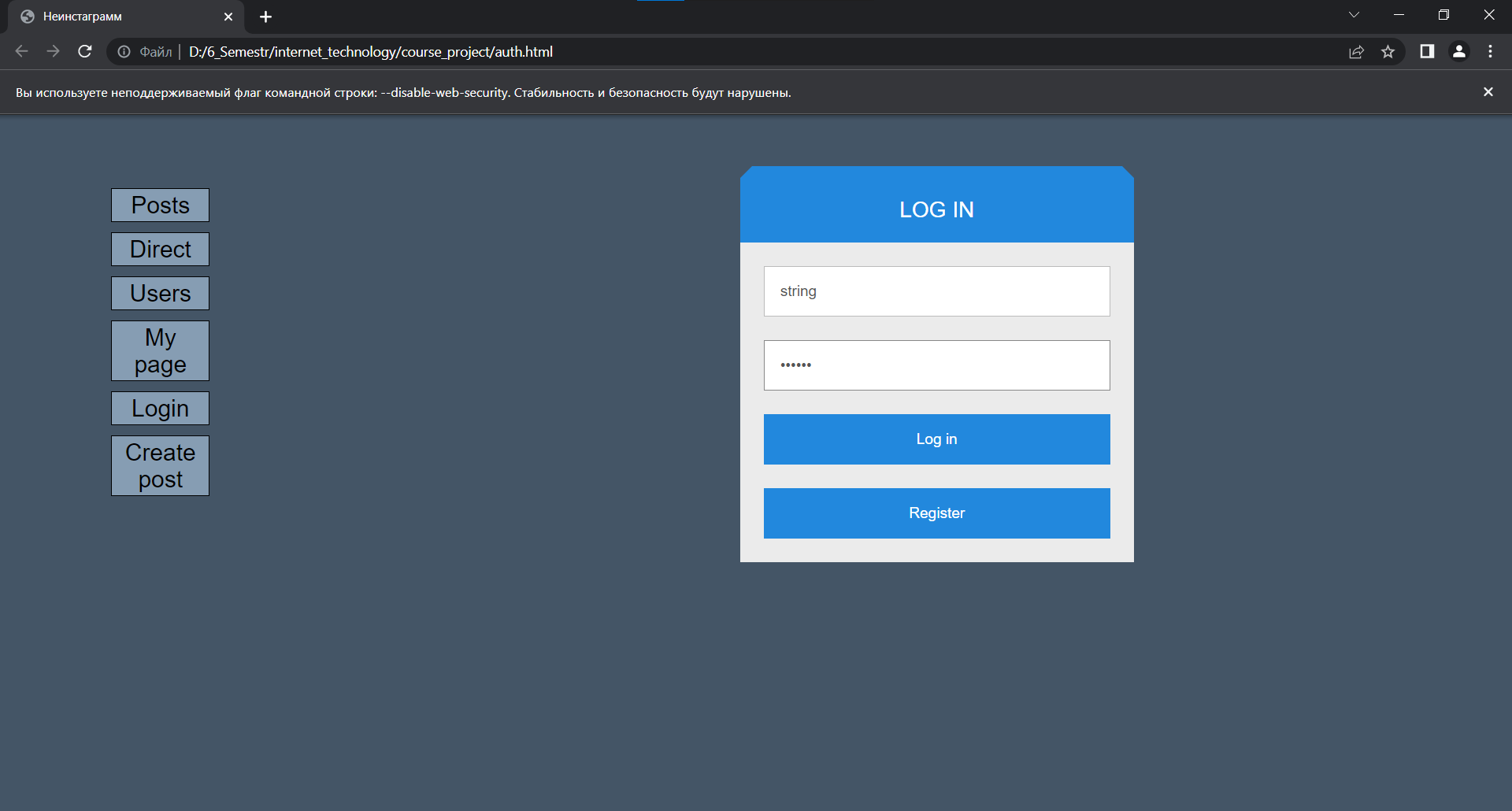


Рисунок 6 – Результат работы сайта

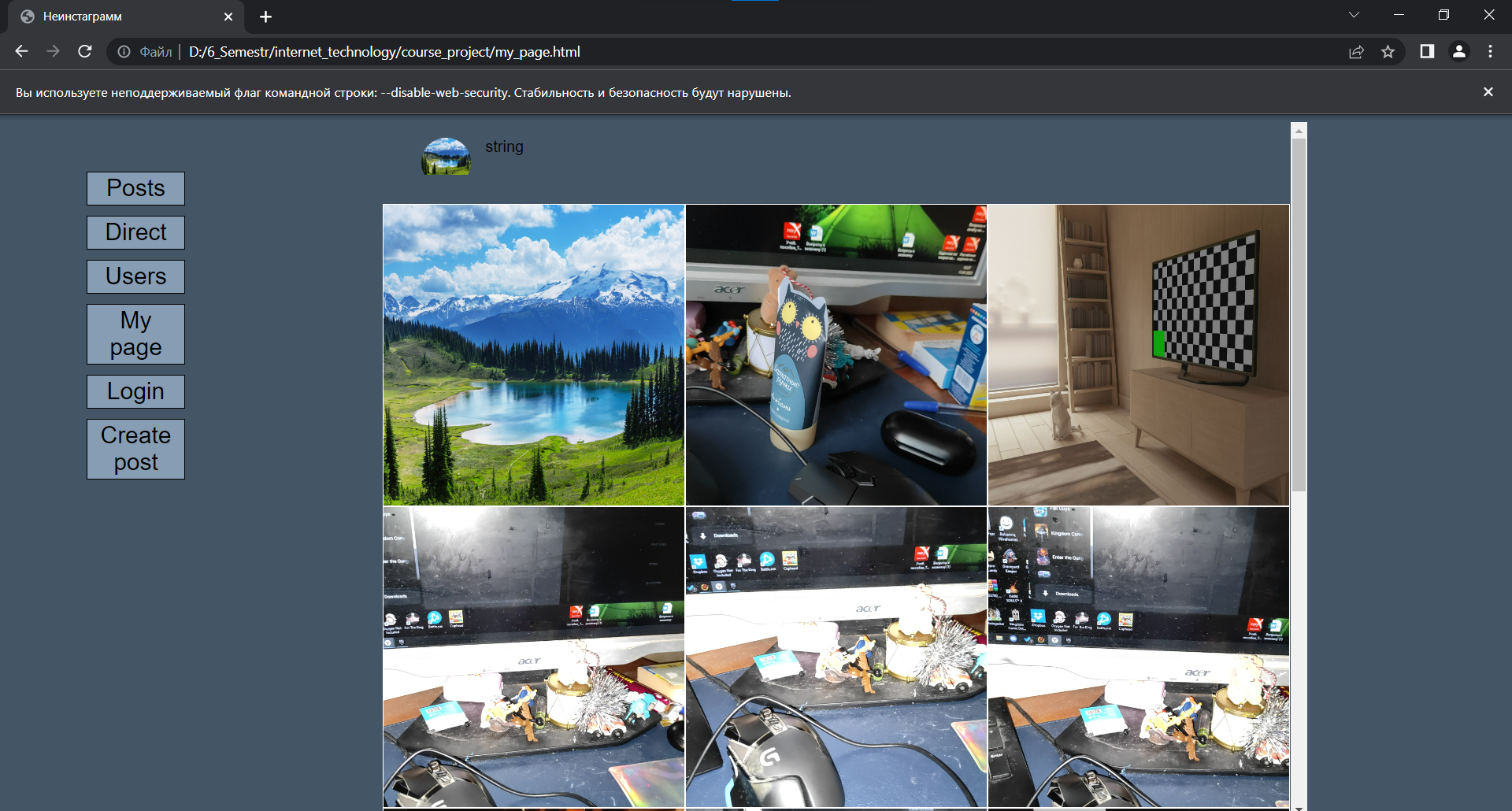


Рисунок 7 – Результат работы сайта

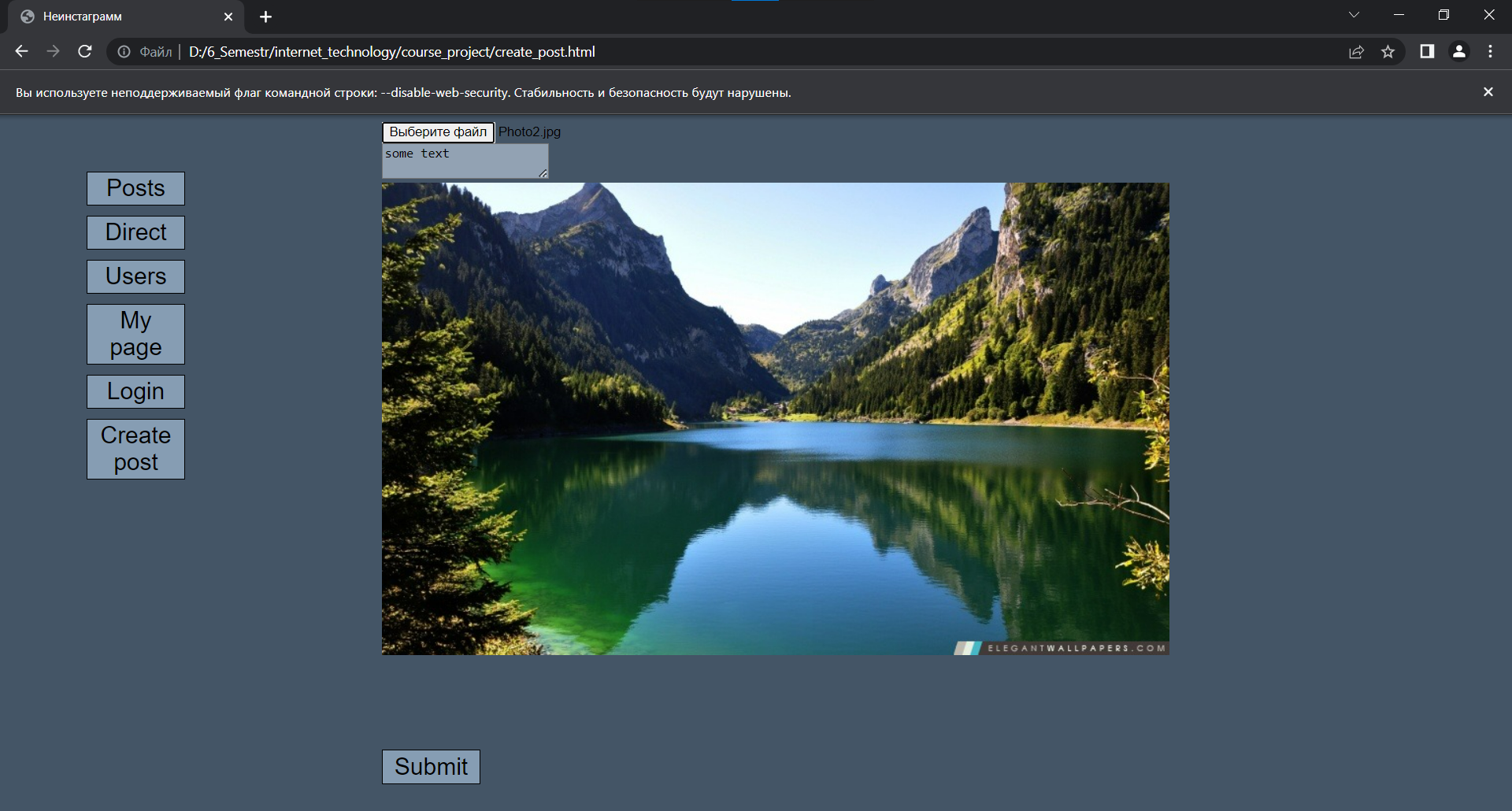


Рисунок 8 – Результат работы сайта

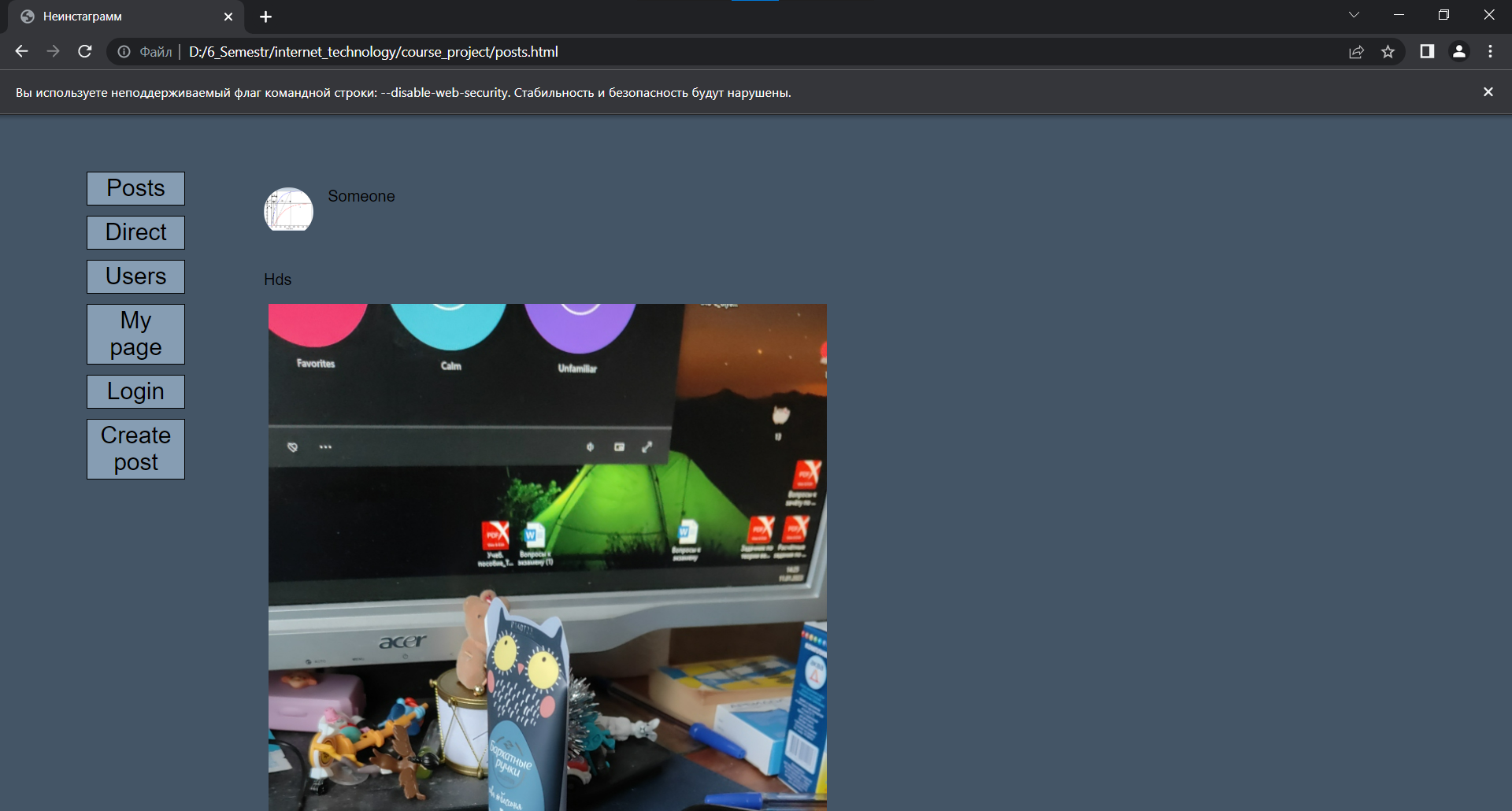


Рисунок 9 – Результат работы сайта

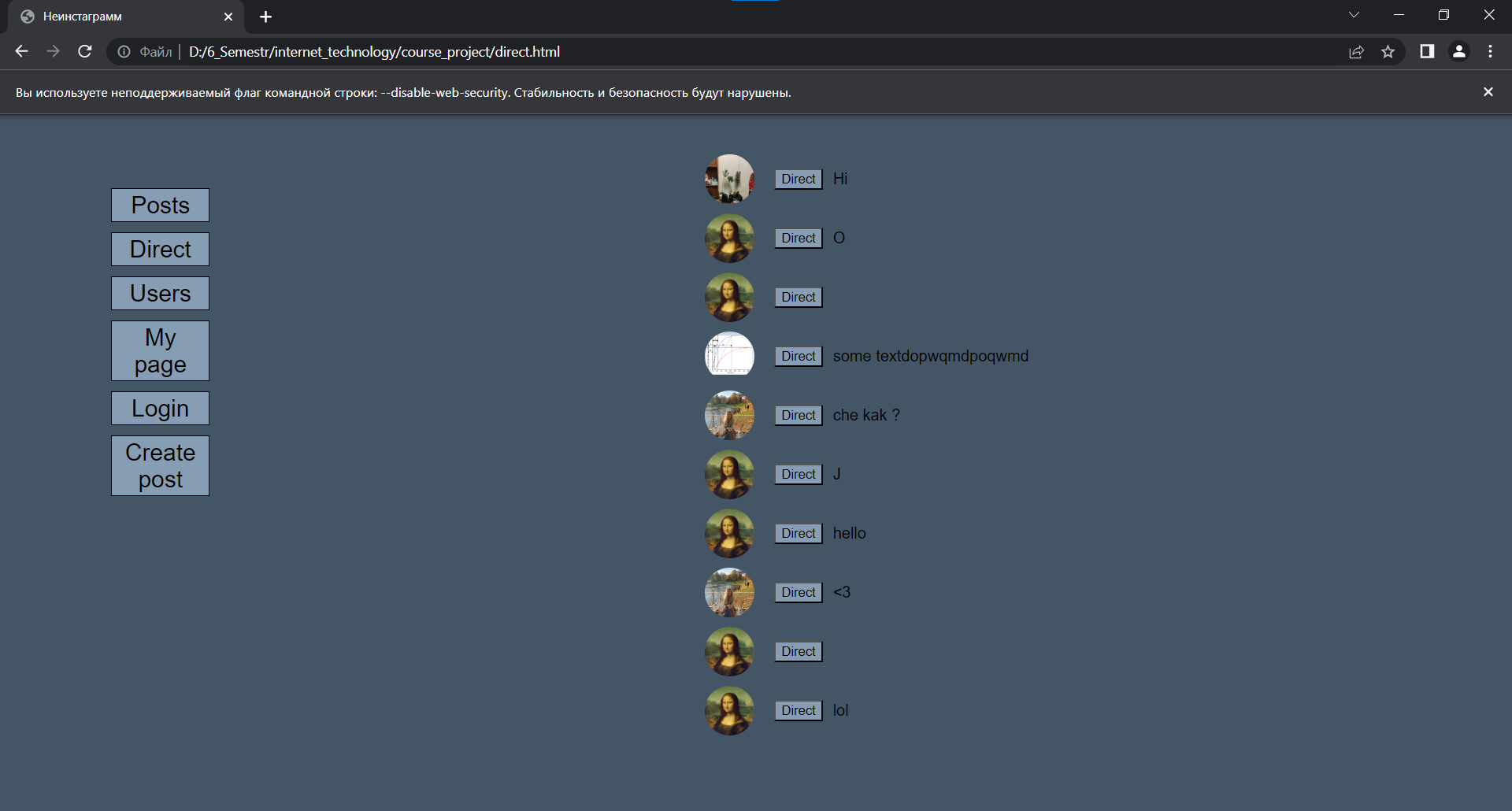


Рисунок 10 – Результаты тестирования

# **СПИСОК ЛИТЕРАТУРЫ**

1 4 типа архитектуры программного обеспечения | by Daria Sidorova | NOP::Nuances of Programming | Medium : сайт. – Хельсинки, 2021 – . – URL: https://medium.com/nuances-of-programming/4-типа-архитектуры-программного-обеспечения-917133174724 (дата обращения: 27.02.22).

2 Что такое SQL? Чем отличаются MySql и PostgreSQL?. – Москва, 2019 – . – URL: https://webhost1.ru/information/blog/2021-05-24-sql-osnovnye-sql-subd.html (дата обращения: 29.03.2023).