**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа бакалавриата «Программная инженерия»

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Научный руководитель,  Преподаватель департамента  программной инженерии    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Н. Степанов  «21» апреля 2022 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  профессор департамента  программной инженерии,  канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г. |

|  |  |
| --- | --- |
| ***Подп. и дата*** |  |
| ***Инв. № дубл.*** |  |
| ***Взам. инв. №*** |  |
| ***Подп. и дата*** |  |
| ***Инв. № подл*** |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Клиент-серверное приложение для обмена навыками и услугами для студентов НИУ ВШЭ. Серверная часть.**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.06.02-01 12 01-1-ЛУ** | | |
|  |  | |
| Исполнитель:  студент группы БПИ191  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Царёв К.Д. /  «21» апреля 2022 г. | |
|  | | |
|  | |  |

**Москва 2022**

|  |  |  |  |
| --- | --- | --- | --- |
| **УТВЕРЖДЕНО**  **RU.17701729.06.02-01 12 01-1-ЛУ** | |  |  |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **Текст программы**  **RU.17701729.06.02-01 12 01-1-ЛУ**    **Листов 43** | | | | |
|  |  | | | |
|  | | | |
|  | | | | |

**Москва 2022**

**АННОТАЦИЯ**

В данном документе приведен текст «Серверной части приложения для обмена навыками и услугами для студентов НИУ ВШЭ». Программа разработана на языке C#.   
Среда разработки - Microsoft Visual Studio Professional 2019.

Настоящий документ разработан в соответствии с требованиями:

1) ГОСТ 19.101-77 Виды программ и программных документов;

2) ГОСТ 19.102-77 Стадии разработки;

3) ГОСТ 19.103-77 Обозначения программ и программных документов;

4) ГОСТ 19.104-78 Основные надписи;

5) ГОСТ 19.105-78 Общие требования к программным документам;

6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом;

7) ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению.

СОДЕРЖАНИЕ

[1. Текст программы 4](#_Toc100012369)

[1.1. Класс SkillsController.cs 4](#_Toc100012370)

[1.2. Класс TransactionsController.cs 6](#_Toc100012371)

[1.3. Класс UsersController.cs 10](#_Toc100012372)

[1.4. Класс FeedbacksController.cs 15](#_Toc100012373)

[1.5. Класс MessagesController.cs 19](#_Toc100012374)

[1.6. Класс ChatsController.cs 22](#_Toc100012375)

[1.7. Класс SharingSkillsContext.cs 24](#_Toc100012376)

[1.8. Класс Skill.cs 25](#_Toc100012377)

[1.9. Класс Transaction.cs 27](#_Toc100012378)

[1.10. Класс User.cs 28](#_Toc100012379)

[1.11. Класс Feedback.cs 31](#_Toc100012380)

[1.12. Класс Message.cs 31](#_Toc100012381)

[1.13. Класс Chat.cs 32](#_Toc100012382)

[1.14. Класс DateRangeAttribute.cs 34](#_Toc100012383)

[1.15. Класс Mail.cs 35](#_Toc100012384)

[1.16. Класс Censorship.cs 35](#_Toc100012385)

[1.17. Класс ChatHub.cs 38](#_Toc100012386)

[1.18. Класс Startup.cs 40](#_Toc100012387)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 42](#_Toc100012388)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 43](#_Toc100012389)

# **1. Текст программы**

Программа состоит из 18 классов (учтены только вручную написанные классы). В данном документе не представлен полностью автоматически сгенерированный исходный код.

Также код программы можно посмотреть в git-репозитории по данной ссылке (<https://github.com/kdtsaryov/SharingSkills_backend>).

## Класс SkillsController.cs

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using SharingSkills\_HSE\_backend.Models;

namespace SharingSkills\_HSE\_backend.Controllers

{

/// <summary>

/// Контроллер навыков

/// </summary>

[Route("api/[controller]")]

[ApiController]

public class SkillsController : ControllerBase

{

/// <summary>

/// Контекст базы данных

/// </summary>

private readonly SharingSkillsContext \_context;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="context">Контекст базы данных</param>

public SkillsController(SharingSkillsContext context)

{

\_context = context;

}

/// <summary>

/// Возвращает все навыки

/// </summary>

// GET: api/Skills

[HttpGet]

public async Task<ActionResult<IEnumerable<Skill>>> GetSkills()

{

return await \_context.Skills.ToListAsync();

}

/// <summary>

/// Возвращает конкретный навык

/// </summary>

/// <param name="id">ID навыка</param>

// GET: api/Skills/1

[HttpGet("{id}")]

public async Task<ActionResult<Skill>> GetSkill(long id)

{

var skill = await \_context.Skills.FirstOrDefaultAsync(t => t.Id == id);

if (skill == null)

return NotFound();

return skill;

}

/// <summary>

/// Возвращает все навыки конкретного пользователя

/// </summary>

/// <param name="mail">Почта</param>

// GET: api/Skills/kdtsaryov@edu.hse.ru/skills

[HttpGet("{mail}/skills")]

public async Task<ActionResult<IEnumerable<Skill>>> GetUserSkills(string mail)

{

return (await \_context.Skills.ToListAsync()).FindAll(s => s.UserMail == mail);

}

/// <summary>

/// Изменение навыка

/// </summary>

/// <param name="id">ID навыка</param>

/// <param name="skill">Навык</param>

// PUT: api/Skills/1

[HttpPut("{id}")]

public async Task<IActionResult> PutSkill(long id, Skill skill)

{

if (id != skill.Id)

return BadRequest();

\_context.Entry(skill).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!SkillExists(id))

return NotFound();

else

throw;

}

return NoContent();

}

/// <summary>

/// Добавление навыка

/// </summary>

/// <param name="skill">Навык</param>

// POST: api/Skills

[HttpPost]

public async Task<ActionResult<Skill>> PostSkill(Skill skill)

{

// Если такой навык уже есть

var s = await \_context.Skills.FindAsync(skill.Id);

if (s != null)

return BadRequest();

// Находим пользователя

var user = await \_context.Users.FindAsync(skill.UserMail);

if (user == null)

return BadRequest();

// Добавляем навык

user.Skills.Add(skill);

\_context.Skills.Add(skill);

// Сохраняем изменения

\_context.Entry(user).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return CreatedAtAction("GetSkill", new { id = skill.Id }, skill);

}

/// <summary>

/// Удаление навыка

/// </summary>

/// <param name="id">ID навыка</param>

// DELETE: api/Skills/5

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteSkill(long id)

{

var skill = await \_context.Skills.FindAsync(id);

if (skill == null)

return NotFound();

// Находим пользователя

var user = await \_context.Users.FindAsync(skill.UserMail);

if (user == null)

return BadRequest();

// Удаляем навык

user.Skills.Remove(skill);

\_context.Skills.Remove(skill);

// Сохраняем изменения

\_context.Entry(user).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return NoContent();

}

/// <summary>

/// Проверка наличия навыка

/// </summary>

/// <param name="id">ID навыка</param>

private bool SkillExists(long id)

{

return \_context.Skills.Any(e => e.Id == id);

}

}

}

## Класс TransactionsController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using SharingSkills\_HSE\_backend.Models;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using SharingSkills\_HSE\_backend.Other;

namespace SharingSkills\_HSE\_backend.Controllers

{

/// <summary>

/// Контроллер обменов

/// </summary>

[Route("api/[controller]")]

[ApiController]

public class TransactionsController : ControllerBase

{

/// <summary>

/// Контекст базы данных

/// </summary>

private readonly SharingSkillsContext \_context;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="context">Контекст базы данных</param>

public TransactionsController(SharingSkillsContext context)

{

\_context = context;

}

/// <summary>

/// Возвращает конкретный обмен

/// </summary>

/// <param name="id">ID обмена</param>

// GET: api/Transactions/1

[HttpGet("{id}")]

public async Task<ActionResult<Transaction>> GetTransaction(long id)

{

var t = await \_context.Transactions.FirstOrDefaultAsync(t => t.Id == id);

if (t == null)

return NotFound();

return t;

}

/// <summary>

/// Возвращает все обмены

/// </summary>

// GET: api/Transactions

[HttpGet]

public async Task<ActionResult<IEnumerable<Transaction>>> GetTransactions()

{

return await \_context.Transactions.ToListAsync();

}

/// <summary>

/// Изменение обмена

/// </summary>

/// <param name="id">ID обмена</param>

/// <param name="transaction">Обмен</param>

/// <param name="mail">Почта того, кто завершил обмен</param>

// PUT: api/Transactions/1?mail=kdtsaryov@edu.hse.ru

[HttpPut("{id}")]

public async Task<IActionResult> PutTransaction(int id, Transaction transaction, string mail = "")

{

if (id != transaction.Id)

return BadRequest();

\_context.Entry(transaction).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!TransactionExists(id))

return NotFound();

else

throw;

}

// Если обмен приняли, то оповещаем отправителя

if (transaction.Status == 1)

{

// Находим отправителя и получателя

var sender = await \_context.Users.FindAsync(transaction.SenderMail);

var receiver = await \_context.Users.FindAsync(transaction.ReceiverMail);

await Mail.SendEmailAsync(sender.Mail, "Принятый обмен",

$"{receiver.Name} {receiver.Surname} принял(а) Ваш обмен.\n" +

$"Зайдите в приложение \"Обмен навыками\", чтобы узнать детали.");

}

// Если обмен завершил один пользователь, то оповещаем другого пользователя

if (transaction.Status == 2)

{

// Находим отправителя и получателя

var sender = await \_context.Users.FindAsync(transaction.SenderMail);

var receiver = await \_context.Users.FindAsync(transaction.ReceiverMail);

// Если завершил отправитель

if (sender.Mail == mail)

{

await Mail.SendEmailAsync(receiver.Mail, "Обмен завершен",

$"{sender.Name} {sender.Surname} завершил(а) обмен.\n" +

$"Зайдите в приложение \"Обмен навыками\", чтобы узнать детали.");

}

// Если завершил получатель

if (receiver.Mail == mail)

{

await Mail.SendEmailAsync(sender.Mail, "Обмен завершен",

$"{receiver.Name} {receiver.Surname} завершил(а) обмен.\n" +

$"Зайдите в приложение \"Обмен навыками\", чтобы узнать детали.");

}

}

return NoContent();

}

/// <summary>

/// Добавление нового обмена

/// </summary>

/// <param name="transaction">Обмен</param>

// POST: api/Transactions

[HttpPost]

public async Task<ActionResult<Transaction>> PostTransaction(Transaction transaction)

{

// Если такой обмен уже есть

var t = await \_context.Transactions.FindAsync(transaction.Id);

if (t != null)

return BadRequest();

// Находим отправителя и получателя

var sender = await \_context.Users.FindAsync(transaction.SenderMail);

var receiver = await \_context.Users.FindAsync(transaction.ReceiverMail);

if (sender == null || receiver == null || sender == receiver)

return BadRequest();

transaction.Status = 0;

// Добавляем обмен

sender.Transactions.Add(transaction);

receiver.Transactions.Add(transaction);

\_context.Transactions.Add(transaction);

// Оповещаем получателя нового обмена

await Mail.SendEmailAsync(receiver.Mail, "Новый обмен",

$"{sender.Name} {sender.Surname} предложил(а) Вам новый обмен.\n" +

$"Зайдите в приложение \"Обмен навыками\", чтобы узнать детали.");

// Сохраняем изменения

\_context.Entry(sender).State = EntityState.Modified;

\_context.Entry(receiver).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return CreatedAtAction("GetTransaction", new { id = transaction.Id }, transaction);

}

/// <summary>

/// Удаление обмена

/// </summary>

/// <param name="id">ID обмена</param>

/// <param name="sendNotification">Отправлять ли уведомление</param>

// DELETE: api/Transactions/1?sendNotification=true

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteTransaction(long id, bool sendNotification = false)

{

var t = await \_context.Transactions.FindAsync(id);

if (t == null)

return NotFound();

// Находим отправителя и получателя

var sender = await \_context.Users.FindAsync(t.SenderMail);

var receiver = await \_context.Users.FindAsync(t.ReceiverMail);

// Если надо, то отправляем уведомление

if (sendNotification)

{

await Mail.SendEmailAsync(sender.Mail, "Отказ в обмене",

$"{receiver.Name} {receiver.Surname} отказал(а) Вам в обмене.\n" +

$"Зайдите в приложение \"Обмен навыками\", чтобы узнать детали.");

}

// Удаляем обмен

sender.Transactions.Remove(t);

receiver.Transactions.Remove(t);

\_context.Transactions.Remove(t);

// Сохраняем изменения

\_context.Entry(sender).State = EntityState.Modified;

\_context.Entry(receiver).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return NoContent();

}

/// <summary>

/// Проверка наличия обмена

/// </summary>

/// <param name="id">ID обмена</param>

private bool TransactionExists(long id)

{

return \_context.Transactions.Any(e => e.Id == id);

}

}

}

## Класс UsersController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using SharingSkills\_HSE\_backend.Models;

using SharingSkills\_HSE\_backend.Other;

namespace SharingSkills\_HSE\_backend.Controllers

{

/// <summary>

/// Контроллер пользователей

/// </summary>

[Route("api/[controller]")]

[ApiController]

public class UsersController : ControllerBase

{

/// <summary>

/// Генератор случайных чисел для генерации кода подтверждения

/// </summary>

private readonly Random rnd = new Random();

/// <summary>

/// Контекст базы данных

/// </summary>

private readonly SharingSkillsContext \_context;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="context">Контекст базы данных</param>

public UsersController(SharingSkillsContext context)

{

\_context = context;

}

/// <summary>

/// Напоминание забытого пароля по почте

/// </summary>

/// <param name="mail">Почта</param>

// GET: api/Users/kdtsaryov@edu.hse.ru/password

[HttpGet("{mail}/password")]

public async Task<ActionResult<User>> ForgetPassword(string mail)

{

// Находим пользователя

var user = await \_context.Users.FirstOrDefaultAsync(u => u.Mail == mail);

// Если такой пользователь есть

if (user != null)

{

// Отправляем письмо с паролем

await Mail.SendEmailAsync(user.Mail, "Напоминание пароля", "Здравствуйте!\n" +

"В Обмене Навыками была нажата кнопка \"Забыли пароль?\"\n" +

$"Ваш пароль - {user.Password}\n");

return Ok();

}

return BadRequest();

}

/// <summary>

/// Возвращает всех пользователей

/// </summary>

// GET: api/Users

[HttpGet]

public async Task<ActionResult<IEnumerable<User>>> GetUsers()

{

return await \_context.Users.ToListAsync();

}

/// <summary>

/// Возвращает пользователя с конкретной почтой

/// </summary>

/// <param name="mail">Почта</param>

// GET: api/Users/kdtsaryov@edu.hse.ru

[HttpGet("{mail}")]

public async Task<ActionResult<User>> GetUser(string mail)

{

// Находим пользователя и возвращаем его, если такой нашелся

var user = await \_context.Users.FirstOrDefaultAsync(u => u.Mail == mail);

if (user == null)

return NotFound();

return user;

}

/// <summary>

/// Возвращает навыки, удовлетворяющие каким-то критериям без навыков конкретного пользователя

/// </summary>

/// <param name="mail">Почта</param>

/// <param name="studyingYearID">Курс</param>

/// <param name="majorID">Образовательная программа</param>

/// <param name="campusLocationID">Расположение корпуса</param>

/// <param name="dormitoryID">Общежитие</param>

/// <param name="gender">Пол</param>

/// <param name="category">Категория</param>

/// <param name="subcategory">Подкатегория</param>

/// <param name="skillstatus">Могу или Хочу</param>

// GET: api/Users/kdtsaryov@edu.hse.ru/skills?studyingYearID=1&majorID=1&campusLocationID=1&dormitoryID=1&gender=1&skillstatus=1&category=1&subcategory=1

[HttpGet("{mail}/skills")]

public async Task<ActionResult<IEnumerable<Skill>>> GetUsersSkills(string mail, int studyingYearID = -1, int majorID = -1,

int campusLocationID = -1, int dormitoryID = -1, int gender = -1,int skillstatus = -1, int category = -1, int subcategory = -1)

{

// Находим пользователя и возвращаем ошибку, если такой не нашелся

var user = await \_context.Users.Include(u => u.Skills).FirstOrDefaultAsync(u => u.Mail == mail);

if (user == null)

return NotFound();

// Если параметры не переданы, то возвращаются все навыки

var users = await \_context.Users.Include(u => u.Skills).ToListAsync();

users = users.FindAll(u => u.Mail != user.Mail);

// Сначала фильтруем пользователей

if (studyingYearID != -1)

users = users.FindAll(u => u.StudyingYearId == studyingYearID);

if (majorID != -1)

users = users.FindAll(u => u.MajorId == majorID);

if (campusLocationID != -1)

users = users.FindAll(u => u.CampusLocationId == campusLocationID);

if (dormitoryID != -1)

users = users.FindAll(u => u.DormitoryId == dormitoryID);

if (gender != -1)

users = users.FindAll(u => u.Gender == gender);

// Записываем все навыки всех отфильтрованных пользователей

List<Skill> skills = new List<Skill>();

foreach (User u in users)

{

skills.AddRange(u.Skills);

}

// Фильтруем уже их навыки по заданным параметрам

if (skillstatus != -1)

skills = skills.FindAll(s => s.Status == skillstatus);

if (category != -1)

skills = skills.FindAll(s => s.Category == category);

if (subcategory != -1)

skills = skills.FindAll(s => s.Subcategory == subcategory);

return skills;

}

/// <summary>

/// Возвращает текущие обмены конкретного пользователя

/// </summary>

/// <param name="mail">Почта</param>

// GET: api/Users/kdtsaryov@edu.hse.ru/transactions/active

[HttpGet("{mail}/transactions/active")]

public async Task<ActionResult<Transaction>> GetUserActiveTransactions(string mail)

{

// Находим пользователя

var user = await \_context.Users.FirstOrDefaultAsync(u => u.Mail == mail);

if (user == null)

return NotFound();

// Если есть обмены

if (user.Transactions != null)

// Возвращаем текущие

return Ok(\_context.Transactions.Where(t => (t.ReceiverMail == mail || t.SenderMail == mail) && t.Status == 1).ToList());

else

return NotFound();

}

/// <summary>

/// Возвращает завершенные обмены конкретного пользователя

/// </summary>

/// <param name="mail">Почта</param>

// GET: api/Users/kdtsaryov@edu.hse.ru/transactions/completed

[HttpGet("{mail}/transactions/completed")]

public async Task<ActionResult<Transaction>> GetUserCompletedTransactions(string mail)

{

// Находим пользователя

var user = await \_context.Users.FirstOrDefaultAsync(u => u.Mail == mail);

if (user == null)

return NotFound();

// Если есть обмены

if (user.Transactions != null)

// Возвращаем завершенные

return Ok(\_context.Transactions.Where(t => (t.ReceiverMail == mail || t.SenderMail == mail) && t.Status == 2).ToList());

else

return NotFound();

}

/// <summary>

/// Возвращает входящие обмены конкретного пользователя

/// </summary>

/// <param name="mail">Почта</param>

// GET: api/Users/kdtsaryov@edu.hse.ru/transactions/in

[HttpGet("{mail}/transactions/in")]

public async Task<ActionResult<Transaction>> GetUserInTransactions(string mail)

{

// Находим пользователя

var user = await \_context.Users.FirstOrDefaultAsync(u => u.Mail == mail);

if (user == null)

return NotFound();

// Если есть обмены

if (user.Transactions != null)

// Возвращаем входящие

return Ok(\_context.Transactions.Where(t => t.ReceiverMail == mail && t.Status == 0).ToList());

else

return NotFound();

}

/// <summary>

/// Возвращает исходящие обмены конкретного пользователя

/// </summary>

/// <param name="mail">Почта</param>

// GET: api/Users/kdtsaryov@edu.hse.ru/transactions/out

[HttpGet("{mail}/transactions/out")]

public async Task<ActionResult<Transaction>> GetUserOutTransactions(string mail)

{

// Находим пользователя

var user = await \_context.Users.FirstOrDefaultAsync(u => u.Mail == mail);

if (user == null)

return NotFound();

// Если есть обмены

if (user.Transactions != null)

// Возвращаем исходящие

return Ok(\_context.Transactions.Where(t => t.SenderMail == mail && t.Status == 0).ToList());

else

return NotFound();

}

/// <summary>

/// Возвращает отзывы конкретного пользователя

/// </summary>

/// <param name="mail">Почта</param>

// GET: api/Users/kdtsaryov@edu.hse.ru/feedbacks

[HttpGet("{mail}/feedbacks")]

public async Task<ActionResult<Feedback>> GetUserFeedbacks(string mail)

{

// Находим пользователя

var user = await \_context.Users.Include(u => u.Feedbacks).FirstOrDefaultAsync(u => u.Mail == mail);

if (user == null)

return NotFound();

// Если есть отзывы

if (user.Feedbacks != null)

// Возвращаем

return Ok(user.Feedbacks);

else

return NotFound();

}

/// <summary>

/// Обновление данных конкретного пользователя

/// </summary>

/// <param name="mail">Почта</param>

/// <param name="user">Пользователь</param>

// PUT: api/Users/kdtsaryov@edu.hse.ru

[HttpPut("{mail}")]

public async Task<IActionResult> PutUser(string mail, User user)

{

if (mail != user.Mail)

return BadRequest();

// Цензурим графу о себе

user.About = Censorship.DoCensorship(user.About);

\_context.Entry(user).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!UserExists(mail))

return NotFound();

else

throw;

}

return NoContent();

}

/// <summary>

/// Добавление нового пользователя

/// </summary>

/// <param name="user">Пользователь</param>

// POST: api/Users

[HttpPost]

public async Task<ActionResult<User>> PostUser(User user)

{

var u = await \_context.Users.FindAsync(user.Mail);

if (u != null)

return BadRequest();

// Цензурим графу о себе

user.About = Censorship.DoCensorship(user.About);

// Генерация кода подтверждения

user.ConfirmationCodeServer = rnd.Next(1000, 10000);

// Отправка этого кода на почту

await Mail.SendEmailAsync(user.Mail, "Код подтверждения регистрации", "Здравствуйте!\n" +

"Спасибо за регистрацию в Обмене Навыками\n" +

$"Ваш код подтверждения - {user.ConfirmationCodeServer}\n");

\_context.Users.Add(user);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetUser", new { mail = user.Mail }, user);

}

/// <summary>

/// Удаление пользователя

/// </summary>

/// <param name="mail">Почта</param>

// DELETE: api/Users/kdtsaryov@edu.hse.ru

[HttpDelete("{mail}")]

public async Task<IActionResult> DeleteUser(string mail)

{

var user = await \_context.Users.FindAsync(mail);

if (user == null)

return NotFound();

\_context.Users.Remove(user);

await \_context.SaveChangesAsync();

return NoContent();

}

/// <summary>

/// Проверка наличия пользователя

/// </summary>

/// <param name="mail">Почтовый адрес</param>

/// <returns>Существует ли такой пользователь</returns>

private bool UserExists(string mail)

{

return \_context.Users.Any(e => e.Mail == mail);

}

}

}

## Класс FeedbacksController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using SharingSkills\_HSE\_backend.Models;

using System.Collections.Generic;

using System.Threading.Tasks;

using SharingSkills\_HSE\_backend.Other;

using System.Linq;

namespace SharingSkills\_HSE\_backend.Controllers

{

/// <summary>

/// Контроллер отзывов

/// </summary>

[Route("api/[controller]")]

[ApiController]

public class FeedbacksController : ControllerBase

{

/// <summary>

/// Контекст базы данных

/// </summary>

private readonly SharingSkillsContext \_context;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="context">Контекст базы данных</param>

public FeedbacksController(SharingSkillsContext context)

{

\_context = context;

}

/// <summary>

/// Возвращает все отзывы

/// </summary>

// GET: api/Feedbacks

[HttpGet]

public async Task<ActionResult<IEnumerable<Feedback>>> GetFeedbacks()

{

return await \_context.Feedbacks.ToListAsync();

}

/// <summary>

/// Возвращает конкретный отзыв

/// </summary>

/// <param name="id">ID отзыва</param>

// GET api/Feedbacks/1

[HttpGet("{id}")]

public async Task<ActionResult<Feedback>> GetFeedback(long id)

{

var f = await \_context.Feedbacks.FirstOrDefaultAsync(f => f.Id == id);

if (f == null)

return NotFound();

return f;

}

/// <summary>

/// Добавление нового отзыва

/// </summary>

/// <param name="feedback">Отзыв</param>

// POST api/Feedbacks

[HttpPost]

public async Task<ActionResult<Feedback>> PostFeedback(Feedback feedback)

{

// Если такой отзыв уже есть

var f = await \_context.Feedbacks.FindAsync(feedback.Id);

if (f != null)

return BadRequest();

// Находим отправителя и получателя

var sender = await \_context.Users.FindAsync(feedback.SenderMail);

var receiver = await \_context.Users.FindAsync(feedback.ReceiverMail);

if (sender == null || receiver == null || sender == receiver)

return BadRequest();

// Цензурим комментарий

feedback.Comment = Censorship.DoCensorship(feedback.Comment);

// Добавляем отзыв

receiver.GradesCount++;

receiver.GradesSum += feedback.Grade;

receiver.AverageGrade = (double)receiver.GradesSum / receiver.GradesCount;

receiver.Feedbacks.Add(feedback);

\_context.Feedbacks.Add(feedback);

// Даем пользователю права модератора

if (receiver.GradesCount >= 5 && receiver.AverageGrade >= 3.5 && !receiver.IsModer)

{

receiver.IsModer = true;

await Mail.SendEmailAsync(receiver.Mail, "Статус модератора",

$"Поздравляем!\n" +

$"Вам стали доступны функции модератора в приложении \"Обмен навыками\".\n" +

$"Теперь Вы можете удалять неприличные отзывы пользователей приложения.");

}

// Оповещаем получателя нового отзыва

await Mail.SendEmailAsync(receiver.Mail, "Новый отзыв",

$"{sender.Name} {sender.Surname} оставил(а) Вам новый отзыв.\n" +

$"Зайдите в приложение \"Обмен навыками\", чтобы узнать детали.");

// Сохраняем изменения

\_context.Entry(receiver).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return CreatedAtAction("GetFeedback", new { id = feedback.Id }, feedback);

}

/// <summary>

/// Изменение отзыва

/// </summary>

/// <param name="id">ID отзыва</param>

/// <param name="feedback">Отзыв</param>

// PUT api/Feedbacks/1

[HttpPut("{id}")]

public async Task<IActionResult> PutFeedback(long id, Feedback feedback)

{

if (id != feedback.Id)

return BadRequest();;

// Цензурим комментарий

feedback.Comment = Censorship.DoCensorship(feedback.Comment);

\_context.Entry(feedback).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!FeedbackExists(id))

return NotFound();

else

throw;

}

return NoContent();

}

/// <summary>

/// Удаление отзыва пользователем

/// </summary>

/// <param name="mail">Почта</param>

/// <param name="id">ID отзыва</param>

// DELETE api/Feedbacks/user/kdtsaryov@edu.hse.ru/1

[HttpDelete("user/{mail}/{id}")]

public async Task<IActionResult> DeleteFeedbackUser(string mail, long id)

{

var f = await \_context.Feedbacks.FindAsync(id);

if (f == null)

return NotFound();

// Чужой отзыв удалять нельзя

if (f.SenderMail != mail)

return BadRequest();

// Находим получателя

var receiver = await \_context.Users.FindAsync(f.ReceiverMail);

// Удаляем отзыв

receiver.GradesCount--;

receiver.GradesSum -= f.Grade;

receiver.AverageGrade = receiver.GradesCount > 0 ? (double)receiver.GradesSum / receiver.GradesCount : 0;

receiver.Feedbacks.Remove(f);

\_context.Feedbacks.Remove(f);

// Сохраняем изменения

\_context.Entry(receiver).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return NoContent();

}

/// <summary>

/// Удаление отзыва модератором

/// </summary>

/// <param name="mail">Почта</param>

/// <param name="id">ID отзыва</param>

// DELETE api/Feedbacks/moder/kdtsaryov@edu.hse.ru/1

[HttpDelete("moder/{mail}/{id}")]

public async Task<IActionResult> DeleteFeedbackModer(string mail, long id)

{

var f = await \_context.Feedbacks.FindAsync(id);

if (f == null)

return NotFound();

// Отзыв себе удалять нельзя

if (f.ReceiverMail == mail)

return BadRequest();

// Находим получателя

var receiver = await \_context.Users.FindAsync(f.ReceiverMail);

// Удаляем отзыв

receiver.GradesCount--;

receiver.GradesSum -= f.Grade;

receiver.AverageGrade = receiver.GradesCount > 0 ? (double)receiver.GradesSum / receiver.GradesCount : 0;

receiver.Feedbacks.Remove(f);

\_context.Feedbacks.Remove(f);

// Сохраняем изменения

\_context.Entry(receiver).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return NoContent();

}

/// <summary>

/// Проверка наличия отзыва

/// </summary>

/// <param name="id">ID отзыва</param>

private bool FeedbackExists(long id)

{

return \_context.Feedbacks.Any(f => f.Id == id);

}

}

}

## Класс MessagesController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using SharingSkills\_HSE\_backend.Models;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace SharingSkills\_HSE\_backend.Controllers

{

/// <summary>

/// Контроллер сообщений

/// </summary>

[Route("api/[controller]")]

[ApiController]

public class MessagesController : ControllerBase

{

/// <summary>

/// Контекст базы данных

/// </summary>

private readonly SharingSkillsContext \_context;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="context">Контекст базы данных</param>

public MessagesController(SharingSkillsContext context)

{

\_context = context;

}

/// <summary>

/// Возвращает конкретное сообщение

/// </summary>

/// <param name="id">ID сообщения</param>

// GET: api/Messages/1

[HttpGet("{id}")]

public async Task<ActionResult<Message>> GetMessage(long id)

{

var m = await \_context.Messages.FirstOrDefaultAsync(m => m.Id == id);

if (m == null)

return NotFound();

return m;

}

/// <summary>

/// Возвращает все сообщения

/// </summary>

// GET: api/Messages

[HttpGet]

public async Task<ActionResult<IEnumerable<Message>>> GetMessages()

{

return await \_context.Messages.ToListAsync();

}

/// <summary>

/// Возвращает 100 последних сообщений между двумя конкретными пользователями

/// </summary>

/// <param name="mail1">Почта 1</param>

/// <param name="mail2">Почта 2</param>

/// <param name="n">Какая по счету подгрузка</param>

// GET: api/Messages/kdtsaryov@edu.hse.ru/eoshtanko@edu.hse.ru/0

[HttpGet("{mail1}/{mail2}/{n}")]

public async Task<ActionResult<IEnumerable<Message>>> GetMessages(string mail1, string mail2, int n)

{

// Находим пользователей

var u1 = await \_context.Users.FindAsync(mail1);

var u2 = await \_context.Users.FindAsync(mail2);

if (u1 == null || u2 == null || u1 == u2)

return BadRequest();

// Находим все сообщения между этими двумя пользователями

var messages = await \_context.Messages.ToListAsync();

messages = messages.FindAll(m => (m.SenderMail == mail1 && m.ReceiverMail == mail2) || (m.SenderMail == mail2 && m.ReceiverMail == mail1));

// Сортируем по времени отправки

messages = messages.OrderByDescending(m => m.SendTime).ToList();

// Пропускаем n\*100 сообщений

messages = messages.Skip(n \* 100).ToList();

// Берем 100 сообщений

messages = messages.Take(100).ToList();

return messages;

}

/// <summary>

/// Изменение конкретного сообщения

/// </summary>

/// <param name="id">ID сообщения</param>

/// <param name="message">Сообщение</param>

// PUT: api/Messages/1

[HttpPut("{id}")]

public async Task<IActionResult> PutMessage(long id, Message message)

{

if (id != message.Id)

return BadRequest();

\_context.Entry(message).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!MessageExists(id))

return NotFound();

else

throw;

}

return NoContent();

}

/// <summary>

/// Добавление нового сообщения

/// </summary>

/// <param name="message">Сообщение</param>

// POST: api/Messages

[HttpPost]

public async Task<ActionResult<Message>> PostMessage(Message message)

{

// Если такое сообщение уже есть

var m = await \_context.Messages.FindAsync(message.Id);

if (m != null)

return BadRequest();

// Находим отправителя и получателя

var sender = await \_context.Users.FindAsync(message.SenderMail);

var receiver = await \_context.Users.FindAsync(message.ReceiverMail);

if (sender == null || receiver == null || sender == receiver)

return BadRequest();

// Находим переписку

var chat = await \_context.Chats.FirstOrDefaultAsync(c => (c.Mail1 == message.SenderMail && c.Mail2 == message.ReceiverMail) ||

(c.Mail2 == message.SenderMail && c.Mail1 == message.ReceiverMail));

if (chat == null)

return BadRequest();

// Обновляем данные переписки

chat.LastMessage = message.Text;

chat.SendTime = message.SendTime;

\_context.Entry(chat).State = EntityState.Modified;

// Добавляем сообщение

\_context.Messages.Add(message);

await \_context.SaveChangesAsync();

return CreatedAtAction("GetMessage", new { id = message.Id }, message);

}

/// <summary>

/// Удаление сообщения

/// </summary>

/// <param name="id">ID сообщения</param>

// DELETE: api/Messages/5

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteMessage(long id)

{

var message = await \_context.Messages.FindAsync(id);

if (message == null)

return NotFound();

// Удаляем сообщение

\_context.Messages.Remove(message);

await \_context.SaveChangesAsync();

return NoContent();

}

/// <summary>

/// Проверка наличия сообщения

/// </summary>

/// <param name="id">ID сообщения</param>

private bool MessageExists(long id)

{

return \_context.Messages.Any(e => e.Id == id);

}

}

}

## Класс ChatsController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using SharingSkills\_HSE\_backend.Models;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace SharingSkills\_HSE\_backend.Controllers

{

/// <summary>

/// Контроллер переписок

/// </summary>

[Route("api/[controller]")]

[ApiController]

public class ChatsController : ControllerBase

{

/// <summary>

/// Контекст базы данных

/// </summary>

private readonly SharingSkillsContext \_context;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="context">Контекст базы данных</param>

public ChatsController(SharingSkillsContext context)

{

\_context = context;

}

/// <summary>

/// Возвращает конкретную переписку

/// </summary>

/// <param name="id">ID переписки</param>

// GET: api/Chats/1

[HttpGet("{id}")]

public async Task<ActionResult<Chat>> GetChat(long id)

{

var c = await \_context.Chats.FirstOrDefaultAsync(c => c.Id == id);

if (c == null)

return NotFound();

return c;

}

/// <summary>

/// Возвращает все переписки

/// </summary>

// GET: api/Chats

[HttpGet]

public async Task<ActionResult<IEnumerable<Chat>>> GetChats()

{

return await \_context.Chats.ToListAsync();

}

/// <summary>

/// Возвращает все переписки конкретного пользователя

/// </summary>

/// <param name="mail">Почта</param>

// GET: api/Chats/kdtsaryov@edu.hse.ru/user

[HttpGet("{mail}/user")]

public async Task<ActionResult<IEnumerable<Chat>>> GetChats(string mail)

{

// Находим пользователя

var user = await \_context.Users.Include(u => u.Chats).FirstOrDefaultAsync(u => u.Mail == mail);

if (user == null)

return BadRequest();

return user.Chats;

}

/// <summary>

/// Изменение конкретной переписки

/// </summary>

/// <param name="id">ID переписки</param>

/// <param name="chat">Переписка</param>

// PUT: api/Chats/1

[HttpPut("{id}")]

public async Task<IActionResult> PutChat(long id, Chat chat)

{

if (id != chat.Id)

return BadRequest();

\_context.Entry(chat).State = EntityState.Modified;

try

{

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!ChatExists(id))

return NotFound();

else

throw;

}

return NoContent();

}

/// <summary>

/// Добавление новой переписки

/// </summary>

/// <param name="chat">Переписка</param>

// POST: api/Chats

[HttpPost]

public async Task<ActionResult<Chat>> PostChat(Chat chat)

{

// Если такая переписка уже есть

var c = await \_context.Chats.FindAsync(chat.Id);

if (c != null)

return BadRequest();

// Находим собеседников

var user1 = await \_context.Users.FindAsync(chat.Mail1);

var user2 = await \_context.Users.FindAsync(chat.Mail2);

if (user1 == null || user2 == null || user1 == user2)

return BadRequest();

// Добавляем переписку

user1.Chats.Add(chat);

user2.Chats.Add(chat);

\_context.Chats.Add(chat);

// Сохраняем изменения

\_context.Entry(user1).State = EntityState.Modified;

\_context.Entry(user2).State = EntityState.Modified;

await \_context.SaveChangesAsync();

return CreatedAtAction("GetChat", new { id = chat.Id }, chat);

}

/// <summary>

/// Удаление переписки

/// </summary>

/// <param name="id">ID переписки</param>

// DELETE: api/Chats/5

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteChat(long id)

{

var chat = await \_context.Chats.FindAsync(id);

if (chat == null)

return NotFound();

// Удаляем переписку

\_context.Chats.Remove(chat);

await \_context.SaveChangesAsync();

return NoContent();

}

/// <summary>

/// Проверка наличия переписки

/// </summary>

/// <param name="id">ID переписки</param>

private bool ChatExists(long id)

{

return \_context.Chats.Any(e => e.Id == id);

}

}

}

## Класс SharingSkillsContext.cs

using Microsoft.EntityFrameworkCore;

namespace SharingSkills\_HSE\_backend.Models

{

/// <summary>

/// Контекст базы данных

/// </summary>

public class SharingSkillsContext : DbContext

{

/// <summary>

/// Конструктор

/// </summary>

/// <param name="options">Параметры базы данных</param>

public SharingSkillsContext(DbContextOptions<SharingSkillsContext> options) : base(options)

{

Database.EnsureCreated();

}

/// <summary>

/// Создание моделей

/// </summary>

/// <param name="builder">"Строитель", связывающий модеди с контекстом БД</param>

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

}

/// <summary>

/// Навыки

/// </summary>

public DbSet<Skill> Skills { get; set; }

/// <summary>

/// Обмены

/// </summary>

public DbSet<Transaction> Transactions { get; set; }

/// <summary>

/// Пользователи

/// </summary>

public DbSet<User> Users { get; set; }

/// <summary>

/// Отзывы

/// </summary>

public DbSet<Feedback> Feedbacks { get; set; }

/// <summary>

/// Сообщения

/// </summary>

public DbSet<Message> Messages { get; set; }

/// <summary>

/// Переписки

/// </summary>

public DbSet<Chat> Chats { get; set; }

}

}

## Класс Skill.cs

using System.ComponentModel.DataAnnotations;

namespace SharingSkills\_HSE\_backend.Models

{

/// <summary>

/// Навык

/// </summary>

public class Skill

{

/// <summary>

/// ID навыка

/// </summary>

[Key]

public long Id { get; set; }

/// <summary>

/// Статус: 1 - могу, 2 - хочу

/// </summary>

[Range(0, 2, ErrorMessage = "Некорректный статус навыка")]

public int Status { get; set; }

/// <summary>

/// Название

/// </summary>

[StringLength(51, ErrorMessage = "Слишком много символов")]

public string Name { get; set; }

/// <summary>

/// Описание

/// </summary>

[StringLength(301, ErrorMessage = "Слишком много символов")]

public string Description { get; set; }

/// <summary>

/// Категория

/// </summary>

[Range(0, 1, ErrorMessage = "Некорректная категория")]

public int Category { get; set; }

/// <summary>

/// Подкатегория

/// </summary>

[Range(0, 21, ErrorMessage = "Некорректная подкатегория")]

public int Subcategory { get; set; }

/// <summary>

/// Почта пользователя

/// </summary>

[RegularExpression(@"[\_A-Za-z0-9]+@edu.hse.ru", ErrorMessage = "Некорректный почтовый адрес")]

public string UserMail { get; set; }

/// <summary>

/// Имя пользователя

/// </summary>

[StringLength(41, MinimumLength = 2, ErrorMessage = "Имя должно быть от 2 до 40 символов")]

[RegularExpression(@"[А-Яа-яЁёA-Za-z]+$", ErrorMessage = "В имени могут присутствовать только буквы")]

public string UserName { get; set; }

/// <summary>

/// Фамилия пользователя

/// </summary>

[StringLength(41, MinimumLength = 2, ErrorMessage = "Фамилия должна быть от 2 до 40 символов")]

[RegularExpression(@"[А-Яа-яЁёA-Za-z]+$", ErrorMessage = "В фамилии могут присутствовать только буквы")]

public string UserSurname { get; set; }

/// <summary>

/// Фотография пользователя

/// </summary>

public byte[] UserPhoto { get; set; }

}

}

## Класс Transaction.cs

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace SharingSkills\_HSE\_backend.Models

{

/// <summary>

/// Обмен

/// </summary>

public class Transaction

{

/// <summary>

/// ID обмена

/// </summary>

[Key]

public long Id { get; set; }

/// <summary>

/// Первый навык

/// </summary>

[StringLength(51, ErrorMessage = "Слишком много символов")]

public string Skill1 { get; set; }

/// <summary>

/// Второй навык

/// </summary>

[StringLength(51, ErrorMessage = "Слишком много символов")]

public string Skill2 { get; set; }

/// <summary>

/// Описание

/// </summary>

[StringLength(301, ErrorMessage = "Слишком много символов")]

public string Description { get; set; }

/// <summary>

/// Почта отправителя

/// </summary>

[RegularExpression(@"[\_A-Za-z0-9]+@edu.hse.ru", ErrorMessage = "Некорректный почтовый адрес отправителя")]

public string SenderMail { get; set; }

/// <summary>

/// Почта получателя

/// </summary>

[RegularExpression(@"[\_A-Za-z0-9]+@edu.hse.ru", ErrorMessage = "Некорректный почтовый адрес получателя")]

public string ReceiverMail { get; set; }

/// <summary>

/// Почта того, кто "хочет"

/// </summary>

[RegularExpression(@"[\_A-Za-z0-9]+@edu.hse.ru", ErrorMessage = "Некорректный почтовый адрес")]

public string WhoWantMail { get; set; }

/// <summary>

/// Статус: 0 - отправлен, 1 - подтвержден (активен), 2 - завершен

/// </summary>

[Range(0, 2, ErrorMessage = "Некорректный статус обмена")]

public int Status { get; set; }

/// <summary>

/// Отправитель и получатель

/// </summary>

public List<User> Users { get; set; } = new List<User>();

}

}

## Класс User.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace SharingSkills\_HSE\_backend.Models

{

/// <summary>

/// Пользователь

/// </summary>

public class User

{

/// <summary>

/// Почта

/// </summary>

[RegularExpression(@"[\_A-Za-z0-9]+@edu.hse.ru", ErrorMessage = "Некорректный почтовый адрес")]

[Key]

public string Mail { get; set; }

/// <summary>

/// Код подтверждения почты, хранящийся на сервере

/// </summary>

public int ConfirmationCodeServer { get; set; }

/// <summary>

/// Код подтверждения почты, запрашиваемый у пользователя

/// </summary>

[Compare("ConfirmationCodeServer", ErrorMessage = "Неверный код подтверждения")]

public int ConfirmationCodeUser { get; set; }

/// <summary>

/// Пароль

/// </summary>

[DataType(DataType.Password)]

[StringLength(41, MinimumLength = 6, ErrorMessage = "Пароль должен быть от 6 до 40 символов")]

public string Password { get; set; }

/// <summary>

/// Имя

/// </summary>

[StringLength(41, MinimumLength = 2, ErrorMessage = "Имя должно быть от 2 до 40 символов")]

[RegularExpression(@"[А-Яа-яЁёA-Za-z]+$", ErrorMessage = "В имени могут присутствовать только буквы")]

public string Name { get; set; }

/// <summary>

/// Фамилия

/// </summary>

[StringLength(41, MinimumLength = 2, ErrorMessage = "Фамилия должна быть от 2 до 40 символов")]

[RegularExpression(@"[А-Яа-яЁёA-Za-z]+$", ErrorMessage = "В фамилии могут присутствовать только буквы")]

public string Surname { get; set; }

/// <summary>

/// Дата рождения

/// </summary>

[DateRange(13, 100, ErrorMessage = "Некорректная дата рождения")]

public DateTime BirthDate { get; set; }

/// <summary>

/// Пол

/// </summary>

[Range(0, 2, ErrorMessage = "Некорректный пол")]

public int Gender { get; set; }

/// <summary>

/// Курс

/// <summary>

[Range(0, 7, ErrorMessage = "Некорректный ID курса")]

public int StudyingYearId { get; set; }

/// <summary>

/// Образовательная программа

/// </summary>

[Range(0, 53, ErrorMessage = "Некорректный ID образовательной программы")]

public int MajorId { get; set; }

/// <summary>

/// Расположение корпуса

/// </summary>

[Range(0, 5, ErrorMessage = "Некорректный ID расположения корпуса")]

public int CampusLocationId { get; set; }

/// <summary>

/// Общежитие

/// </summary>

[Range(0, 12, ErrorMessage = "Некорректный ID общежития")]

public int DormitoryId { get; set; }

/// <summary>

/// О себе

/// </summary>

[StringLength(201, ErrorMessage = "Слишком много символов")]

public string About { get; set; } = "";

/// <summary>

/// Ссылка на мессенджер (telegram или vk)

/// </summary>

[RegularExpression(@"(vk.com/|t.me/)+[-A-Za-z\_.0-9]+$", ErrorMessage = "Некорректный формат ссылки")]

public string Contact { get; set; }

/// <summary>

/// Фотография

/// </summary>

public byte[] Photo { get; set; }

/// <summary>

/// Обмены

/// </summary>

public List<Transaction> Transactions { get; set; } = new List<Transaction>();

/// <summary>

/// Навыки

/// </summary>

public List<Skill> Skills { get; set; } = new List<Skill>();

/// <summary>

/// Отзывы

/// </summary>

public List<Feedback> Feedbacks { get; set; } = new List<Feedback>();

/// <summary>

/// Переписки

/// </summary>

public List<Chat> Chats { get; set; } = new List<Chat>();

/// <summary>

/// Количество оценок

/// </summary>

public int GradesCount { get; set; } = 0;

/// <summary>

/// Сумма оценок

/// </summary>

public int GradesSum { get; set; } = 0;

/// <summary>

/// Средняя оценка

/// </summary>

public double AverageGrade { get; set; } = 0;

/// <summary>

/// Является ли пользователь модератором

/// </summary>

public bool IsModer { get; set; } = false;

}

}

## Класс Feedback.cs

using System.ComponentModel.DataAnnotations;

namespace SharingSkills\_HSE\_backend.Models

{

/// <summary>

/// Отзыв

/// </summary>

public class Feedback

{

/// <summary>

/// ID отзыва

/// </summary>

[Key]

public long Id { get; set; }

/// <summary>

/// Оценка

/// </summary>

[Range(0, 5, ErrorMessage = "Некорректная оценка")]

public int Grade { get; set; }

/// <summary>

/// Комментарий

/// </summary>

[StringLength(301, ErrorMessage = "Слишком много символов")]

public string Comment { get; set; } = "";

/// <summary>

/// Почта отправителя

/// </summary>

[RegularExpression(@"[\_A-Za-z0-9]+@edu.hse.ru", ErrorMessage = "Некорректный почтовый адрес отправителя")]

public string SenderMail { get; set; }

/// <summary>

/// Почта получателя

/// </summary>

[RegularExpression(@"[\_A-Za-z0-9]+@edu.hse.ru", ErrorMessage = "Некорректный почтовый адрес получателя")]

public string ReceiverMail { get; set; }

}

}

## Класс Message.cs

using System;

using System.ComponentModel.DataAnnotations;

namespace SharingSkills\_HSE\_backend.Models

{

/// <summary>

/// Сообщение

/// </summary>

public class Message

{

/// <summary>

/// ID сообщения

/// </summary>

[Key]

public long Id { get; set; }

/// <summary>

/// Время отправки

/// </summary>

public string SendTime { get; set; }

/// <summary>

/// Текст

/// </summary>

[StringLength(301, ErrorMessage = "Слишком много символов")]

public string Text { get; set; }

/// <summary>

/// Почта отправителя

/// </summary>

[RegularExpression(@"[\_A-Za-z0-9]+@edu.hse.ru", ErrorMessage = "Некорректный почтовый адрес отправителя")]

public string SenderMail { get; set; }

/// <summary>

/// Почта получателя

/// </summary>

[RegularExpression(@"[\_A-Za-z0-9]+@edu.hse.ru", ErrorMessage = "Некорректный почтовый адрес получателя")]

public string ReceiverMail { get; set; }

}

}

## Класс Chat.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace SharingSkills\_HSE\_backend.Models

{

/// <summary>

/// Переписка

/// </summary>

public class Chat

{

/// <summary>

/// ID переписки

/// </summary>

[Key]

public long Id { get; set; }

/// <summary>

/// Последнее сообщение

/// </summary>

[StringLength(301, ErrorMessage = "Слишком много символов")]

public string LastMessage { get; set; }

/// <summary>

/// Время отправки последнего сообщения

/// </summary>

public string SendTime { get; set; }

/// <summary>

/// Почта первого собеседника

/// </summary>

[RegularExpression(@"[\_A-Za-z0-9]+@edu.hse.ru", ErrorMessage = "Некорректный почтовый адрес собеседника")]

public string Mail1 { get; set; }

/// <summary>

/// Имя первого собеседника

/// </summary>

[StringLength(41, MinimumLength = 2, ErrorMessage = "Имя должно быть от 2 до 40 символов")]

[RegularExpression(@"[А-Яа-яЁёA-Za-z]+$", ErrorMessage = "В имени могут присутствовать только буквы")]

public string Name1 { get; set; }

/// <summary>

/// Фамилия первого собеседника

/// </summary>

[StringLength(41, MinimumLength = 2, ErrorMessage = "Фамилия должна быть от 2 до 40 символов")]

[RegularExpression(@"[А-Яа-яЁёA-Za-z]+$", ErrorMessage = "В фамилии могут присутствовать только буквы")]

public string Surname1 { get; set; }

/// <summary>

/// Фотография первого собеседника

/// </summary>

public byte[] Photo1 { get; set; }

/// <summary>

/// Почта второго собеседника

/// </summary>

[RegularExpression(@"[\_A-Za-z0-9]+@edu.hse.ru", ErrorMessage = "Некорректный почтовый адрес собеседника")]

public string Mail2 { get; set; }

/// <summary>

/// Имя второго собеседника

/// </summary>

[StringLength(41, MinimumLength = 2, ErrorMessage = "Имя должно быть от 2 до 40 символов")]

[RegularExpression(@"[А-Яа-яЁёA-Za-z]+$", ErrorMessage = "В имени могут присутствовать только буквы")]

public string Name2 { get; set; }

/// <summary>

/// Фамилия второго собеседника

/// </summary>

[StringLength(41, MinimumLength = 2, ErrorMessage = "Фамилия должна быть от 2 до 40 символов")]

[RegularExpression(@"[А-Яа-яЁёA-Za-z]+$", ErrorMessage = "В фамилии могут присутствовать только буквы")]

public string Surname2 { get; set; }

/// <summary>

/// Фотография второго собеседника

/// </summary>

public byte[] Photo2 { get; set; }

/// <summary>

/// Собеседники

/// </summary>

public List<User> Users { get; set; } = new List<User>();

}

}

## Класс DateRangeAttribute.cs

using System;

using System.ComponentModel.DataAnnotations;

namespace SharingSkills\_HSE\_backend

{

/// <summary>

/// Атрибут валидации даты рождения

/// </summary>

public class DateRangeAttribute : ValidationAttribute

{

/// <summary>

/// Макимальный год рождения

/// </summary>

int maxYear;

/// <summary>

/// Минимальный год рождения

/// </summary>

int minYear;

/// <summary>

/// Конструктор

/// </summary>

/// <param name="min">Минимальный возраст</param>

/// <param name="max">Максимальный возраст</param>

public DateRangeAttribute(int min, int max)

{

maxYear = DateTime.Now.Year - min;

minYear = DateTime.Now.Year - max;

}

/// <summary>

/// Непосредственно валидация

/// </summary>

/// <param name="value">Переданная дата рождения</param>

/// <returns></returns>

public override bool IsValid(object value)

{

DateTime basic = new DateTime();

if (((DateTime)value) == basic)

return true;

if (value != null && ((DateTime)value).Year >= minYear && ((DateTime)value).Year <= maxYear)

return true;

return false;

}

}

}

## Класс Mail.cs

using MimeKit;

using MailKit.Net.Smtp;

using System.Threading.Tasks;

namespace SharingSkills\_HSE\_backend.Other

{

/// <summary>

/// Класс работы с почтой

/// </summary>

public class Mail

{

/// <summary>

/// Отправка письма на почту

/// </summary>

/// <param name="email">Почта получателя</param>

/// <param name="subject">Тема письма</param>

/// <param name="message">Сообщение</param>

/// <returns></returns>

public static async Task SendEmailAsync(string email, string subject, string message)

{

// Создание самого письма

var emailMessage = new MimeMessage();

emailMessage.From.Add(new MailboxAddress("Обмен навыками", "sharingskills@sharingskills.xyz"));

emailMessage.To.Add(new MailboxAddress("", email));

emailMessage.Subject = subject;

emailMessage.Body = new TextPart("Plain") { Text = message };

// Подключение к почтовому сервису, с которого будет отправляться почта

using (var client = new SmtpClient())

{

await client.ConnectAsync("smtp.yandex.ru", 465, true);

await client.AuthenticateAsync("sharingskills@sharingskills.xyz", "admin1337admin");

// Отправка письма

await client.SendAsync(emailMessage);

await client.DisconnectAsync(true);

}

}

}

}

## Класс Censorship.cs

using System;

using System.Collections.Generic;

namespace SharingSkills\_HSE\_backend.Other

{

/// <summary>

/// Класс для автоматической цензуры нецензцрной лексики

/// </summary>

public class Censorship

{

/// <summary>

/// Нецензурные слова

/// </summary>

private static List<string> words = new List<string> {

"апездал", "апездошенная", "блядь", "блядство", "выебон", "выебать", "членосос", "членоплет", "шлюха",

"ебунок", "еблан", "ёбнуть", "ёболызнуть", "ебош", "заебал", "мокрощелка", "наебка", "наебал", "наебаловка",

"напиздеть", "отъебись", "пидарас", "пиздатый", "пиздец", "пизданутый", "поебать", "поебустика", "разъебанный",

"сука", "сучка", "трахать", "уебок", "уебать", "хуево", "хуесос", "хуеть", "хуевертить", "хуеглот", "хуистика",

"гомосек", "долбоёб", "ебло", "еблище", "ебать", "вхуюжить", "злаебучий", "заёб", "колдоебина", "манда", "мандовошка",

"заебатый", "отхуевертить", "опизденеть", "охуевший", "отебукать", "пизда", "охуеть", "подзалупный", "пизденыш",

"припиздак", "разъебать", "распиздяй", "проебать", "уебан", "хитровыебанный", "хуй", "хуйня", "хуета", "угондошить" };

/// <summary>

/// Варианты написания одних и тех же букв

/// </summary>

private static Dictionary<char, List<char>> d = new Dictionary<char, List<char>>() {

{ 'а', new List<char> { 'а', 'a', '@' } },

{ 'б', new List<char> { 'б', '6', 'b' } },

{ 'в', new List<char> { 'в', 'b', 'v' } },

{ 'г', new List<char> { 'г', 'r', 'g' } },

{ 'д', new List<char> { 'д', 'd' } },

{ 'е', new List<char> { 'е', 'e' } },

{ 'ё', new List<char> { 'ё', 'e' } },

{ 'ж', new List<char> { 'ж', '\*' } },

{ 'з', new List<char> { 'з', '3', 'z' } },

{ 'и', new List<char> { 'и', 'u', 'i' } },

{ 'й', new List<char> { 'й', 'u', 'i' } },

{ 'к', new List<char> { 'к', 'k', } },

{ 'л', new List<char> { 'л', 'l' } },

{ 'м', new List<char> { 'м', 'm' } },

{ 'н', new List<char> { 'н', 'h', 'n' } },

{ 'о', new List<char> { 'о', 'o', '0' } },

{ 'п', new List<char> { 'п', 'n', 'p' } },

{ 'р', new List<char> { 'р', 'r', 'p' } },

{ 'с', new List<char> { 'с', 'c', 's' } },

{ 'т', new List<char> { 'т', 'm', 't' } },

{ 'у', new List<char> { 'у', 'y', 'u' } },

{ 'ф', new List<char> { 'ф', 'f' } },

{ 'х', new List<char> { 'х', 'x', 'h' } },

{ 'ц', new List<char> { 'ц', 'c' } },

{ 'ч', new List<char> { 'ч' } },

{ 'ш', new List<char> { 'ш' } },

{ 'щ', new List<char> { 'щ' } },

{ 'ь', new List<char> { 'ь', 'b' } },

{ 'ы', new List<char> { 'ы' } },

{ 'ъ', new List<char> { 'ъ' } },

{ 'э', new List<char> { 'э', 'e' } },

{ 'ю', new List<char> { 'ю' } },

{ 'я', new List<char> { 'я' } }

};

/// <summary>

/// Метод, определяющий расстояние Левенштейна для двух слов

/// </summary>

/// <param name="firstWord">Первое слово</param>

/// <param name="secondWord">Второе слово</param>

/// <returns></returns>

private static int LevenshteinDistance(string firstWord, string secondWord)

{

int n = firstWord.Length + 1;

int m = secondWord.Length + 1;

int[,] matrixD = new int[n, m];

for (int i = 0; i < n; i++)

{

matrixD[i, 0] = i;

}

for (int j = 0; j < m; j++)

{

matrixD[0, j] = j;

}

for (int i = 1; i < n; i++)

{

for (int j = 1; j < m; j++)

{

int cost = firstWord[i - 1] == secondWord[j - 1] ? 0 : 1;

matrixD[i, j] = Math.Min(Math.Min(matrixD[i - 1, j] + 1, matrixD[i, j - 1] + 1), matrixD[i - 1, j - 1] + cost);

}

}

return matrixD[n - 1, m - 1];

}

/// <summary>

/// Непосредственно зацензуривание

/// </summary>

/// <param name="phrase">Фраза</param>

public static string DoCensorship(string phrase)

{

// Делаем из фразы новую в нижнем регистре, и с буквами только из русского алфавита

string newphrase = phrase.ToLower();

foreach (var letter in d)

{

foreach (char l in letter.Value)

{

foreach (char p in newphrase)

{

if (l == p)

newphrase = newphrase.Replace(p, letter.Key);

}

}

}

// Для каждого нецензурного слова проходимся по отрезкам фразы, равным длине этого слова

foreach (string w in words)

{

for (int i = 0; i < newphrase.Length - w.Length + 1; i++)

{

string fragment = newphrase.Substring(i, w.Length);

// Если расстояние Левенштейна для отрезка <= 25% длины слова

if (LevenshteinDistance(fragment, w) <= w.Length \* 0.25)

{

// Заменяем отрезок на символы цензуры

string fr = phrase.Substring(i, w.Length);

phrase = phrase.Replace(fr, "\*\*\*");

newphrase = newphrase.Replace(fragment, "\*\*\*");

}

}

}

return phrase;

}

}

}

## Класс ChatHub.cs

using Microsoft.AspNetCore.SignalR;

using SharingSkills\_HSE\_backend.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace SharingSkills\_HSE\_backend.Other

{

/// <summary>

/// Класс с информацией о подключениях к чату

/// </summary>

public class UserChatInfo

{

/// <summary>

/// ID подключения

/// </summary>

public string ConnectionID { get; set; }

/// <summary>

/// Почта пользователя

/// </summary>

public string Mail { get; set; }

/// <summary>

/// Конструктор

/// </summary>

/// <param name="mail">Почта</param>

/// <param name="connectionId">ID подключения</param>

public UserChatInfo(string mail, string connectionId)

{

this.Mail = mail;

this.ConnectionID = connectionId;

}

}

/// <summary>

/// Хаб всех чатов

/// </summary>

public class ChatHub : Hub

{

public static List<UserChatInfo> users = new List<UserChatInfo>();

/// <summary>

/// Отправка сообщения

/// </summary>

/// <param name="mail">Почта получателя</param>

/// <param name="message">Сообщение</param>

public async Task Send(string mail, Message message)

{

// Находим пользователя по почте в подключенных к хабу

var user = users.Where(u => u.Mail == mail).FirstOrDefault();

if (user != null)

{

await Clients.Client(user.ConnectionID).SendAsync("Receive", message);

await Clients.Caller.SendAsync("Receive", message);

}

else

{

await Mail.SendEmailAsync(mail, "Новое сообщение",

$"Вам отправили новое сообщение.\n" +

$"Зайдите в приложение \"Обмен навыками\", чтобы узнать детали.");

await Clients.Caller.SendAsync("Receive", message);

}

}

/// <summary>

/// Сопоставление почты с ID подключения

/// </summary>

/// <param name="mail">Почта</param>

public void SetMail(string mail)

{

// Находим пользователя с текущим ID подключения в подключенных к хабу

var user = users.SingleOrDefault(u => u.ConnectionID == Context.ConnectionId);

// Если такой есть, то устанавливаем почту

if (user != null)

user.Mail = mail;

}

/// <summary>

/// Подключение клиента к хабу

/// </summary>

public override Task OnConnectedAsync()

{

// Находим пользователя с текущим ID подключения в подключенных к хабу

var user = users.Where(u => u.ConnectionID == Context.ConnectionId).SingleOrDefault();

// Если такого нет, то добавляем с пустым значением почты

if (user == null)

{

user = new UserChatInfo("", Context.ConnectionId);

users.Add(user);

}

return base.OnConnectedAsync();

}

/// <summary>

/// Отключение клиента от хаба

/// </summary>

public override Task OnDisconnectedAsync(Exception e)

{

// Находим пользователя с текущим ID подключения в подключенных к хабу

var user = users.Where(p => p.ConnectionID == Context.ConnectionId).FirstOrDefault();

// Если такой есть, то удаляем его из списка подключенных

if (user != null)

users.Remove(user);

return base.OnDisconnectedAsync(e);

}

}

}

## Класс Startup.cs

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.EntityFrameworkCore;

using SharingSkills\_HSE\_backend.Models;

using System;

using SharingSkills\_HSE\_backend.Other;

namespace SharingSkills\_HSE\_backend

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

/// <summary>

/// Подключение сервисов

/// </summary>

/// <param name="services">Сервисы</param>

public void ConfigureServices(IServiceCollection services)

{

// Подключение SignalR для чата

services.AddSignalR();

// Подключение JSON сериализатора

services.AddControllers().AddNewtonsoftJson(options =>

options.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore);

// Подключение контроллеров

services.AddControllers();

// Подключение контекста базы данных

if (Environment.GetEnvironmentVariable("ASPNETCORE\_ENVIRONMENT") == "Production")

services.AddDbContext<SharingSkillsContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("SharingSkillsContextProd")));

else

services.AddDbContext<SharingSkillsContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("SharingSkillsContext")));

services.BuildServiceProvider().GetService<SharingSkillsContext>().Database.Migrate();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapHub<ChatHub>("/chat");

endpoints.MapControllers();

});

}

}

}

**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2. ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3. ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4. ГОСТ 19.104-78 Основные надписи. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5. ГОСТ 19.105-78 Общие требования к программным документам. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7. ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8. ГОСТ 19.603-78 Общие правила внесения изменений. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способ
10. Клиент-сервер [Электронный ресурс] / Википедия. Режим доступа: <https://ru.wikipedia.org/wiki/Клиент_—_сервер>, свободный. (дата обращения: 20.11.2021).
11. Введение в REST API — RESTful веб-сервисы [Электронный ресурс] / Хабр. Режим доступа: <https://habr.com/ru/post/483202/>, свободный. (дата обращения: 09.10.2021).
12. Deploy an ASP.NET Core App with EF Core and SQL Server to Azure [Электронный̆ ресурс] / medium. Режим доступа: <https://medium.com/net-core/deploy-an-asp-net-core-app-with-ef-core-and-sql-server-to-azure-e11df41a4804>, свободный. (дата обращения: 15.12.2021).
13. Руководство по ASP.NET Core 5 [Электронный ресурс] / metanit. Режим доступа: <https://metanit.com/sharp/aspnet5>, свободный. (дата обращения: 29.12.2021).
14. PushSharp — простая работа с Push Notifications [Электронный ресурс] / Хабр. Режим доступа: <https://habr.com/ru/post/178775/>, свободный. (дата обращения: 09.01.2022).
15. iOS and Android Push Notifications with PushSharp [Электронный ресурс] / Matt Burke. Режим доступа: <https://www.mattburkedev.com/ios-and-android-push-notifications-with-pushsharp/>, свободный. (дата обращения: 12.01.2022).
16. Introduction to SignalR [Электронный ресурс] / Microsoft Docs. Режим доступа: <https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr>, свободный. (дата обращения: 01.02.2022).

# **ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированх |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |