**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  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  профессор департамента  программной инженерии,  канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 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 БПИ193  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Царёв К.Д. /  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г. | |
|  | | |
|  | |  |

**Москва 2021**

|  |  |  |  |
| --- | --- | --- | --- |
| **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 25** | | | | |
|  |  | | | |
|  | | | |
|  | | | | |

**Москва 2021**

# **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](#_Toc71121551)

[1.1. Класс SkillsController.cs 4](#_Toc71121552)

[1.2. Класс TransactionsController.cs 6](#_Toc71121553)

[1.3. Класс UsersController.cs 10](#_Toc71121554)

[1.4. Класс SharingSkillsContext.cs 15](#_Toc71121555)

[1.5. Класс Skill.cs 16](#_Toc71121556)

[1.6. Класс Transaction.cs 17](#_Toc71121557)

[1.7. Класс User.cs 18](#_Toc71121558)

[1.8. Класс DateRangeAttribute.cs 20](#_Toc71121559)

[1.9. Класс Mail.cs 21](#_Toc71121560)

[1.10. Класс Startup.cs 22](#_Toc71121561)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 24](#_Toc71121562)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 25](#_Toc71121563)

# **1 Текст программы**

Программа состоит из 10 классов (учтены только вручную написанные классы). В данном документе не представлен полностью автоматически сгенерированный исходный код.

## 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.Include(s => s.User).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.Include(s => s.User).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.Include(s => s.User).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();

\_context.Skills.Remove(skill);

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(int id)

{

var t = await \_context.Transactions.Include(t => t.Users).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.Include(t => t.Users).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.");

}

\_context.Transactions.Remove(t);

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.Include(u => u.Transactions).Include(u => u.Skills).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.Include(u => u.Transactions).Include(u => u.Skills).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.Include(u => u.Transactions).Include(u => u.Skills).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.Transactions).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.Transactions).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.Include(u => u.Transactions).Include(u => u.Skills).FirstOrDefaultAsync(u => u.Mail == mail);

if (user == null)

return NotFound();

// Если есть обмены

if (user.Transactions != null)

// Возвращаем текущие

return Ok(user.Transactions.Where(t => 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.Include(u => u.Transactions).Include(u => u.Skills).FirstOrDefaultAsync(u => u.Mail == mail);

if (user == null)

return NotFound();

// Если есть обмены

if (user.Transactions != null)

// Возвращаем завершенные

return Ok(user.Transactions.Where(t => 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.Include(u => u.Transactions).Include(u => u.Skills).FirstOrDefaultAsync(u => u.Mail == mail);

if (user == null)

return NotFound();

// Если есть обмены

if (user.Transactions != null)

// Возвращаем входящие

return Ok(user.Transactions.Where(t => t.Status == 0).Where(t => t.ReceiverMail == user.Mail).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.Include(u => u.Transactions).Include(u => u.Skills).FirstOrDefaultAsync(u => u.Mail == mail);

if (user == null)

return NotFound();

// Если есть обмены

if (user.Transactions != null)

// Возвращаем исходящие

return Ok(user.Transactions.Where(t => t.Status == 0).Where(t => t.SenderMail == user.Mail).ToList());

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();

\_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();

// 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 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; }

}

}

## Класс Skill.cs

using System;

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>

/// Пользователь

/// </summary>

public User User { get; set; }

}

}

## Класс Transaction.cs

using System;

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 "может"

/// </summary>

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

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 (Кириллица)")]

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 (Кириллица)")]

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>();

}

}

## 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", "sharing-skills@sharing-skills.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("sharing-skills@sharing-skills.xyz", "admin1337");

// Отправка письма

await client.SendAsync(emailMessage);

await client.DisconnectAsync(true);

}

}

}

}

## 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;

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 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.MapControllers();

});

}

}

}

**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. ГОСТ 19.102-77 Стадии разработки. Единая система программной документации. – М.: Стандартинформ, 2010.
2. ГОСТ 19.101-77 Виды программ и программных документов. Единая система программной документации. – М.: Стандартинформ, 2010.
3. ГОСТ 19.103-77 Обозначения программ и программных документов. Единая система программной документации. – М.: Стандартинформ, 2010.
4. ГОСТ 19.104-78 Основные надписи. Единая система программной документации. – М.: Стандартинформ, 2010.
5. ГОСТ 19.105-78 Общие требования к программным документам. Единая система программной документации. – М.: Стандартинформ, 2010.
6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. Единая система программной документации. – М.: Стандартинформ, 2010.
7. ГОСТ 19.401-78 Единая система программной документации. Текст программы. Требования к содержанию и оформлению. – М.: Стандартинформ, 2010.
8. Клиент-сервер [Электронный ресурс] // Википедия. [2020—2021]. Дата обновления: 21.10.2017. URL: https://ru.wikipedia.org/wiki/Клиент\_—\_сервер (дата обращения: 20.11.2020).
9. Deploy an ASP.NET Core App with EF Core and SQL Server to Azure [Электронный ресурс] // medium. [2020-2021]. Дата обновления: 01.01.2020. URL: https://medium.com/net-core/deploy-an-asp-net-core-app-with-ef-core-and-sql-server-to-azure-e11df41a4804 (дата обращения: 15.02.2021).
10. Руководство по ASP.NET Core 5 [Электронный ресурс] // metanit. [2020—2021]. Дата обновления: 15.12.2020. URL: https://metanit.com/sharp/aspnet5 (дата обращения: 29.12.2020).

# **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 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |