*Федеральное государственное бюджетное образовательное учреждение высшего образования*

*«Владимирский государственный университет*

*имени Александра Григорьевича и Николая Григорьевича Столетовых»*

*Кафедра информационных систем и программной инженерии*

***ПОЯСНИТЕЛЬНАЯ ЗАПИСКА***

***к курсовому проекту по дисциплине   
"Технологии программирования"***

***на тему***

*Проектирование и разработка программной системы   
информационной системы «Мессенджер»*

*Выполнил: студент гр. ПРИ-120*

*Бочков М. А.*

*Принял: доц. кафедры ИСПИ*

*Вершинин В.В.*

*Владимир, 2022*

**Аннотация**

В процессе выполнения курсового проекта производились проектирование и разработка программной системы “Мессенджер”. Процесс выполнения состоял из трёх этапов, включающих в себя описание предметной области, моделирование структуры объектов предметной области и их взаимодействия на концептуальном уровне, реализацию программной системы, содержащую серверную часть системы и интерфейс пользователя.

Реализованная система содержит 5 сущностей, представляющих данные о пользователях системы с возможностью присваивания различных ролей, друзьях пользователей, сообщениях. В рамках разработки был реализован функционал для отправки сообщений и добавления пользователей в друзья, функционал для авторизации, функционал для удаления нежелательных пользователей администратором.

During the course project developing, the design and development of the “Messenger” software system was carried out. The execution process consisted of three stages, including a description of the subject area, modeling the structure of objects in the subject area and their interaction at the conceptual level, implementing the software system containing the server part of the system and the interface user.

The implemented system contains 5 entities representing data about system users with the possibility of assigning different roles, users' friends, messages. As part of the development, functionality was implemented for sending messages and adding users to friends, functionality for authorization, functionality for removing unwanted users by the administrator.

**Содержание**

ВВЕДЕНИЕ 3

1 Описание предметной области 3

1.1. Описание предметно области 5

1.2. Требования к системе 5

1.3. Пользователи разрабатываемой системы 5

1.4. Начальная оценка и выделение сущностей 5

1.5. Словарь предметной области 5

2. проектировние системы 6

1.5. Диаграмма прецедентов 6

1.5. Спецификация прецдентов 6

1.5. Диаграмма классов 9

1.5. Диаграмма состояния 9

1.5. Диаграмма последовательности 9

2. Разработка системы 10

1.5. Используемый стек технологий 10

1.5. Схема базы данных 10

1.5. Создание моделей и применение миграций 10

1.5. Интерфейс и реализация системы 11

заключение 18

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19

ПРИЛОЖЕНИЕ А 20

ПРИЛОЖЕНИЕ Б 28

# ВВЕДЕНИЕ

В процессе выполнения курсового проекта требуется разработать систему хранения данных предметной области «Мессенджер» для автоматизации работы с данными о пользователях, их сообщениях и друзьях.

Цель работы: изучение процесса разработки сложных систем на примере конкретной программной системы, позволяющей вести общение между пользователями в реальном времени, добавлять в друзья других пользователей в реальном времени, удалять нежелательных пользователей посредством администрации.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать хранилище данных, в котором будет находиться информация о различных сущностях системы.

2. Разработать интерфейс, с которым будет взаимодействовать пользователь.

3. Разработать серверную часть приложения, которая будет являться посредником между интерфейсом пользователя и хранилищем данных.

# Описание предметной области

* 1. Описание предметной области

Мессенджер – программа для обмена информацией в реальном времени.

Каждый мессенджер должен иметь как минимум 2-ух пользователей. Также должен быть чат для обмена информацией между пользователями и должен включать в себя оповещение о получении сообщения. Некоторые мессенджеры поддерживают функцию добавления в друзья.

Также некоторые мессенджеры включают в себя создание групповых чатов.

* 1. Требования к системе

Функциональные требования:

* Регистрация аккаунта.
* Вход в аккаунт.
* Отправка сообщений другим пользователям
* Получение сообщений от других пользователей в реальном времени
* Добавление в друзья
* Удаление пользователей администратором

Нефункциональные требования:

* Серверная часть должна быть реализована на платформе .NET 5 с использованием технологии ASP.NET MVC.
* Реализация интерфейса с использованием технологии Razor View.
* Взаимодействие с базой данных осуществляется посредством Entity Framework Core.
* В качестве СУБД будет использоваться SQL Server.
  1. Пользователи разрабатываемой подсистемы

*Пользователь –* может общаться с другими пользователями, добавлять в друзья.

*Администратор –* может удалять других пользователей.

* 1. Начальная оценка и выделение сущностей

Для предметной области «Мессенджер» были выделены следующие сущности: аккаунт, профиль, друзья, личные сообщения, групповой чат, групповые сообщения.

* 1. Словарь предметной области

*Аккаунт (Account) —* создание аккаунтов.

Атрибуты: уникальный номер, почта пользователя, пароль пользователя

*Пользователь (User) —* Профиль пользователя, который отображает информацию о нём.

Атрибуты: уникальный номер, номер аккаунта, имя и фамилия пользователя, дополнительная информация.

*Друг (friend)* — состояние пользователя по отношению к другому пользователю.

Атрибуты: уникальный номер, номер первого пользователя, номер второго пользователя, статус.

*Групповой чат* – чат, поддерживающий общение между несколькими пользователями.

Атрибуты: уникальный номер, номер группового чата, номер пользователя

*Групповое сообщение (GroupMessage)* — сообщение в групповом чате.

Атрибуты: уникальный номер, номер пользователя, номер группового чата, текст сообщения, дата.

*Личное сообщение (PrivateMessage) –* сообщение в личной переписке.

Атрибуты: уникальный номер, номер первого пользователя, номер второго пользователя, текст сообщения, дата.

# проектирование системы

* 1. Диаграмма прецедентов

См. приложение Б

* 1. Спецификация прецедентов

Описание прецедентов

**Название: “**Регистрация”

**Предусловие:** -

**Действующее лицо:** Пользователь

**Основной поток:** Регистрация пройдена

Данные о пользователя сохранятся в базе данных и будут использованы в дальнейшем.

**Альтернативный поток:** Ошибка при регистрации

На шаге 2 пользователь не смог пройти регистрацию, по причине уже существующего пользователя с таким никнеймом.

**Постусловие:** Пользователь сможет пройти авторизацию

**Название: “**Авторизация”

**Предусловие:** Пользователь должен пройти регистрацию

**Действующее лицо:** Пользователь

**Основной поток:** Пользователь авторизируется в системе.

Данные о пользователя сверяются с данными из базы данных, и в случае нахождения соответствия, пользователь войдет в систему.

**Альтернативный поток:** Ошибка при авторизации

На шаге 2 пользователь не смог пройти авторизацию, по ряду возможных причин:

1. Не прошел регистрацию
2. Ошибка при вводе данных.

**Постусловие:** Пользователь войдет в систему.

**Название: “**Поиск людей”

**Предусловие:** Пользователь должен быть авторизирован

**Действующее лицо:** Пользователь

**Основной поток:** Пользователь получает список всех людей

Пользователь получает список всех людей, котором он может отправить сообщение, либо добавить в друзья.

**Альтернативный поток:** Отсутствие людей в системе

На шаге 2 пользователь смог воспользоваться поиском, но при этом список людей был пуст.

**Постусловие:** Пользователь сможет отправить сообщение, либо добавить в друзья

**Название: “**Отправлять сообщения”

**Предусловие:** Пользователь должен быть авторизирован,

**Действующее лицо:** Пользователь

**Основной поток:** Сообщение будет отправлено

Пользователь набирает свое сообщение, которое будет отправлено.

**Альтернативный поток:** Ошибка при отправке

На шаге 2 пользователь не смог отправить сообщение, в связи с техническими неполадками, либо при попытке отправить пустое сообщение. /-

**Постусловие:** Другой пользователь получает сообщение

**Название: “**Получать сообщения”

**Предусловие:** Пользователь должен быть авторизирован, другой пользователь должен отправить текущему сообщение.

**Действующее лицо:** Пользователь

**Основной поток:** будет получено сообщение

Пользователь получает сообщение от другого пользователя

**Альтернативный поток:** Ошибка при получении

На шаге 2 пользователь не смог получить сообщение, в связи с техническими неполадками.

**Постусловие:** Пользователь получает оповещение.

**Название: “**Получать оповещение”

**Предусловие:** Пользователь должен быть авторизирован, другой пользователь должен отправить текущему сообщение/заявку в друзья/создать с текущим пользователем групповой чат.

**Действующее лицо:** Пользователь

**Основной поток:** будет получено оповещение

Пользователь получает оповещение о полученном сообщении/о том, что пришла заявка в друзья / о том, что другой пользователь создал с ним групповой чат.

**Альтернативный поток:** Ошибка при получении

На шаге 2 пользователь не смог получить сообщение, в связи с техническими неполадками.

**Постусловие:** -

**Название: “**Создать групповой чат”

**Предусловие:** пользователь должен быть авторизирован,

**Действующее лицо:** Пользователь

**Основной поток:** будет создан групповой чат

Пользователь приглашает других пользователей в групповой чат

**Альтернативный поток:** Другой пользователь не будет добавлен

На шаге 2 пользователь не смог авторизоваться, в связи с этим ему стоит проверить данные, либо пользователь пытается добавить другого пользователя, который не является для него другом. /-

**Постусловие:** Другие пользователи будут добавлены в групповой чат, и они получат оповещение

**Название: “**Добавить в друзья”

**Предусловие:** пользователь должен быть зарегистрирован и авторизирован в системе в системе.

**Действующее лицо:** Пользователь

**Основной поток:** принять заявку в друзья

В БД с таблицей “Пользователь” будет связана таблица “Друзья”, в которую будут заноситься данные пользователя, которого добавляют в друзья.

**Альтернативный поток:** отклонить заявку в друзья

Пользователь, которого добавляют в друзья, может отклонить приглашение, в этом случаи данные не будут добавляться в БД

**Постусловие:** Пользователь, который добавлял в друзья другого пользователя, получит соответствующее оповещение.

* 1. Диаграмма классов

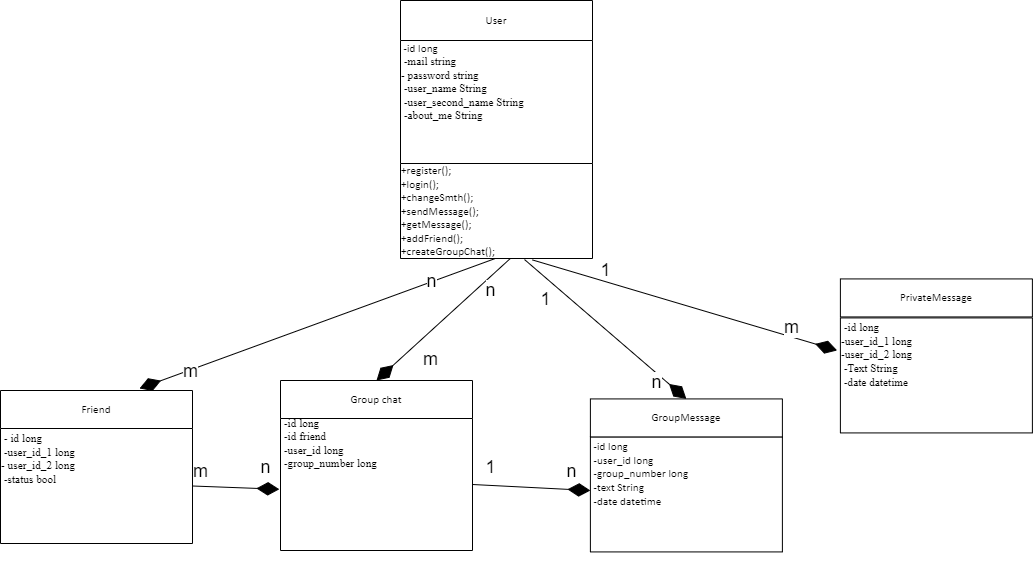


Рис.1. Диаграмма классов.

* 1. Диаграмма состояния

См. приложение Б

* 1. Диаграмма последовательностей

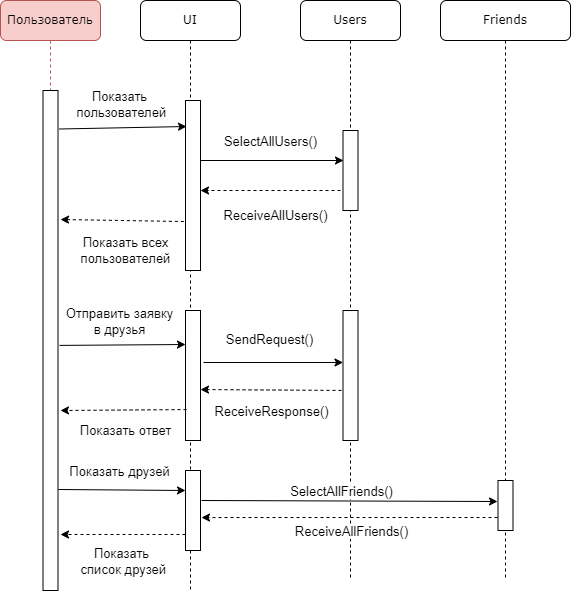


Рис.2. Диаграмма последовательности

# РАЗРАБОТКА системы

* 1. Используемый стек технологий

Разработка ведётся на платформе .NET 5 с использованием технологии ASP.NET Core (Шаблон ASP.NET Core Web App MVC).

Для организации работы функций в реальном времени была использована библиотека Signalr Core

Веб-страницы обрабатываются с использованием технологии Razor.

В качестве СУБД используется реляционная СУБД SQL Server.

Взаимодействие серверной части и СУБД происходит с использованием Entity Framework Core.

* 1. Схема базы данных

За основу структуры базы данных была взята UML диаграмма классов сущностей.

* 1. Создание моделей и применение миграций

Entity Framework Core позволяет на основе готовых моделей сущностей (C#-классов) создавать структуру базы данных напрямую в СУБД с использованием миграций. Таким образом, в соответствии с UML-диаграммой классов были созданы модели сущностей в виде C#-классов, указаны отношения между ними, далее был создан контекст (наследник класса DbContext), с помощью которого будут применяться миграции (Рис. 6-8). Контекст используется для взаимодействия системы с внешней СУБД.

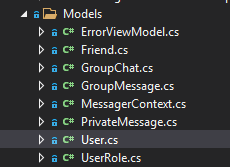


Рис.3. Список всех моделей данных

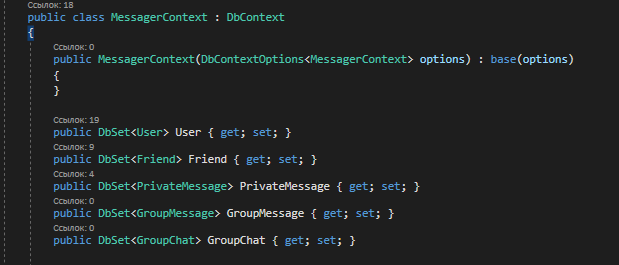


Рис.4. Контекст данных

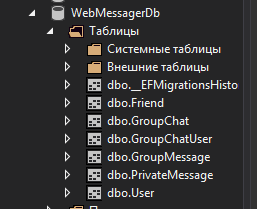


Рис.5. Список таблиц

* 1. Интерфейс и реализация функционала

Основной функционал системы обрабатывает на сервере, в частности в классе ChatHub, обращение к серверу происходит по средством Java Script, определенном на конкретной странице.

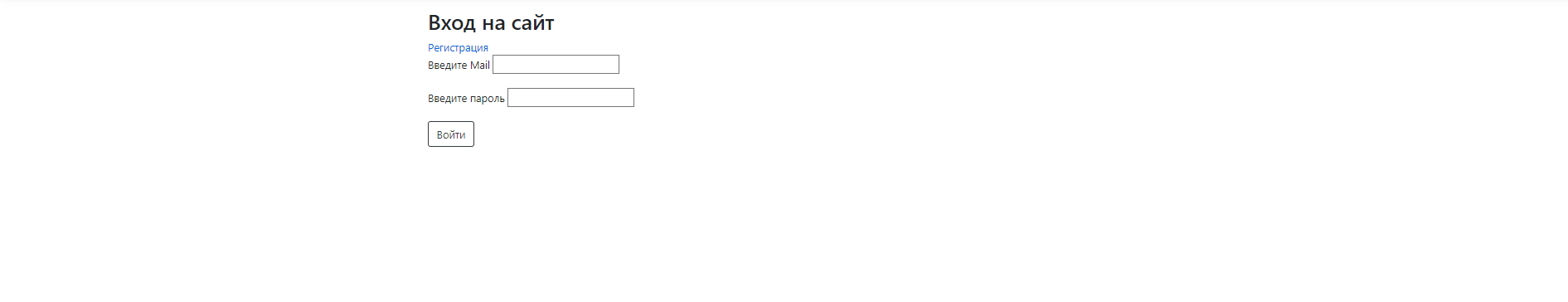


Рис.6. Начальная страница

На рисунке №6 видна начальная страница, которая находится по маршруту “/”, на которой находится форма для входа, если аккаунт еще не создан, то можно перейти по ссылке на регистрации. Перейдем на регистрацию (Рисунок №7).

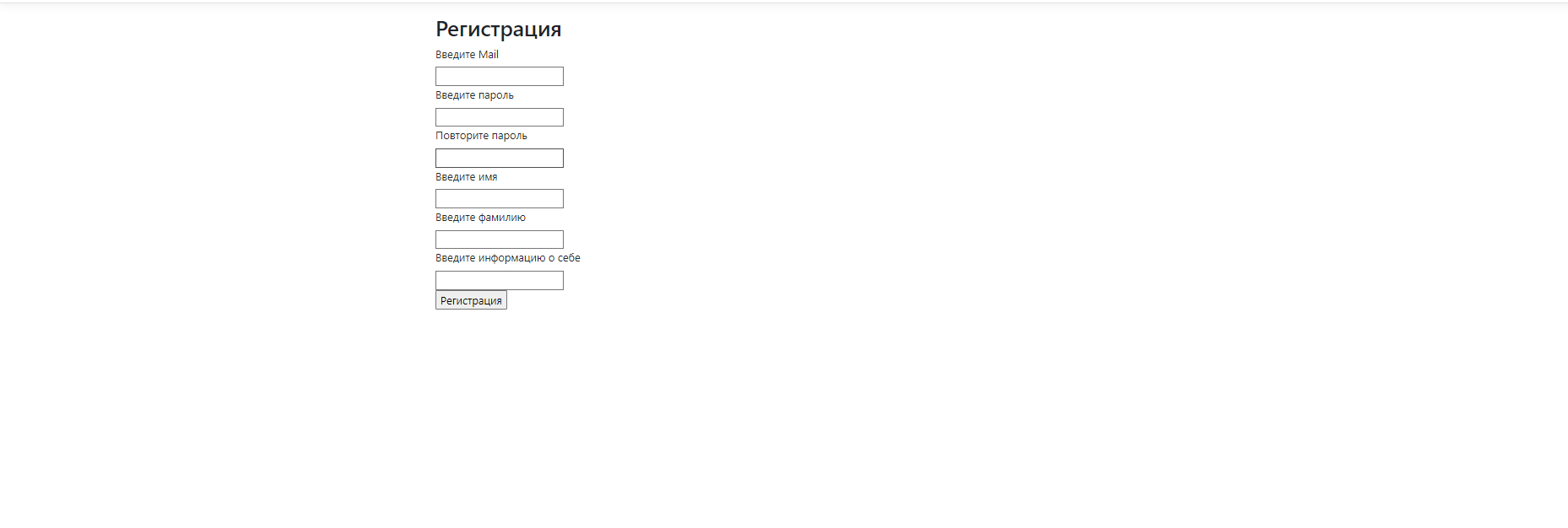


Рис.7. Страница регистрации

На странице регистрации присутствуем форма содержащая обязательны поля для заполнения (Mail, пароль, подтверждение пароля, имя, фамилию) и необязательное поле (Информация о себе).

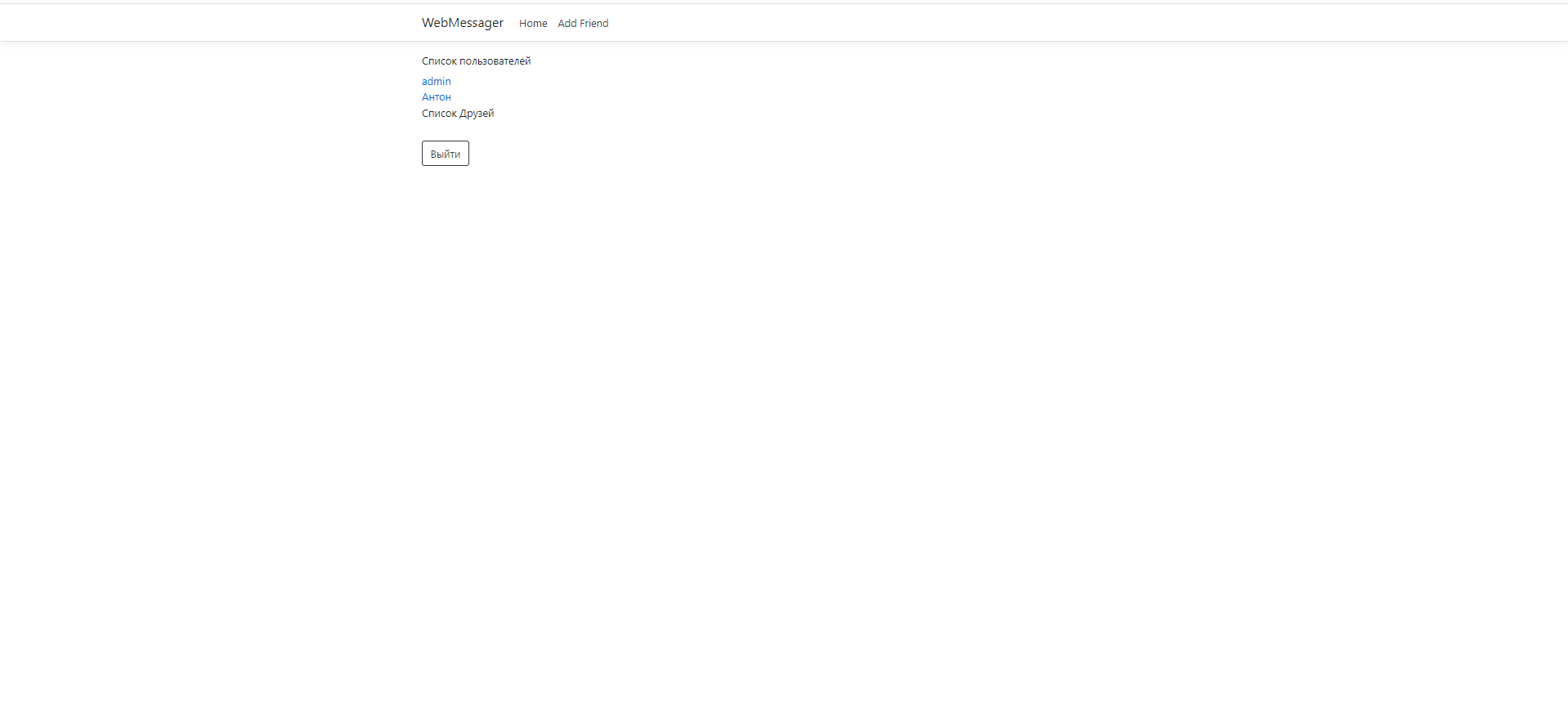


Рис.8. Главная страница.

На рисунке №8 видна главная страница, на которой рендерится секция со списком всех пользователей и секция со списком друзей конкретного пользователя, присутствует ссылка для добавления в друзья, кнопка выхода из аккаунта.

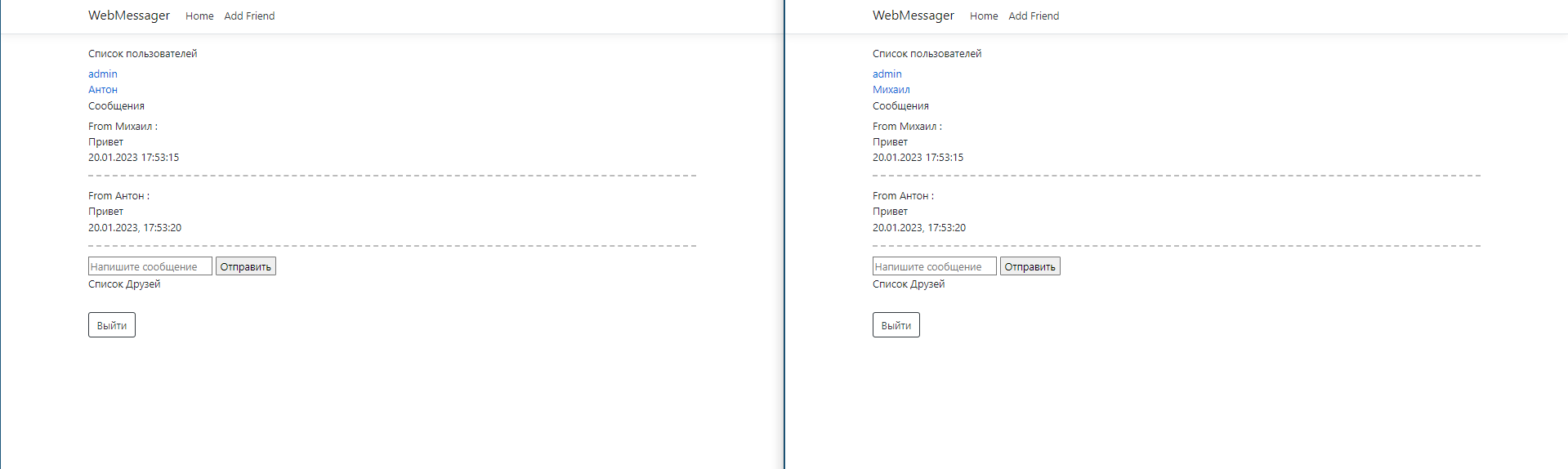


Рис.9. Страница с приватными сообщениями.

При переходе на чат с конкретным пользователем (Рисунок №9) рендерится секция с сообщениями, которые достаются из БД в методе контроллера, при отправке сообщений, посылается запрос на сервер, который добавляет пришедшее сообщение в БД, и рассылает его другому пользователю.

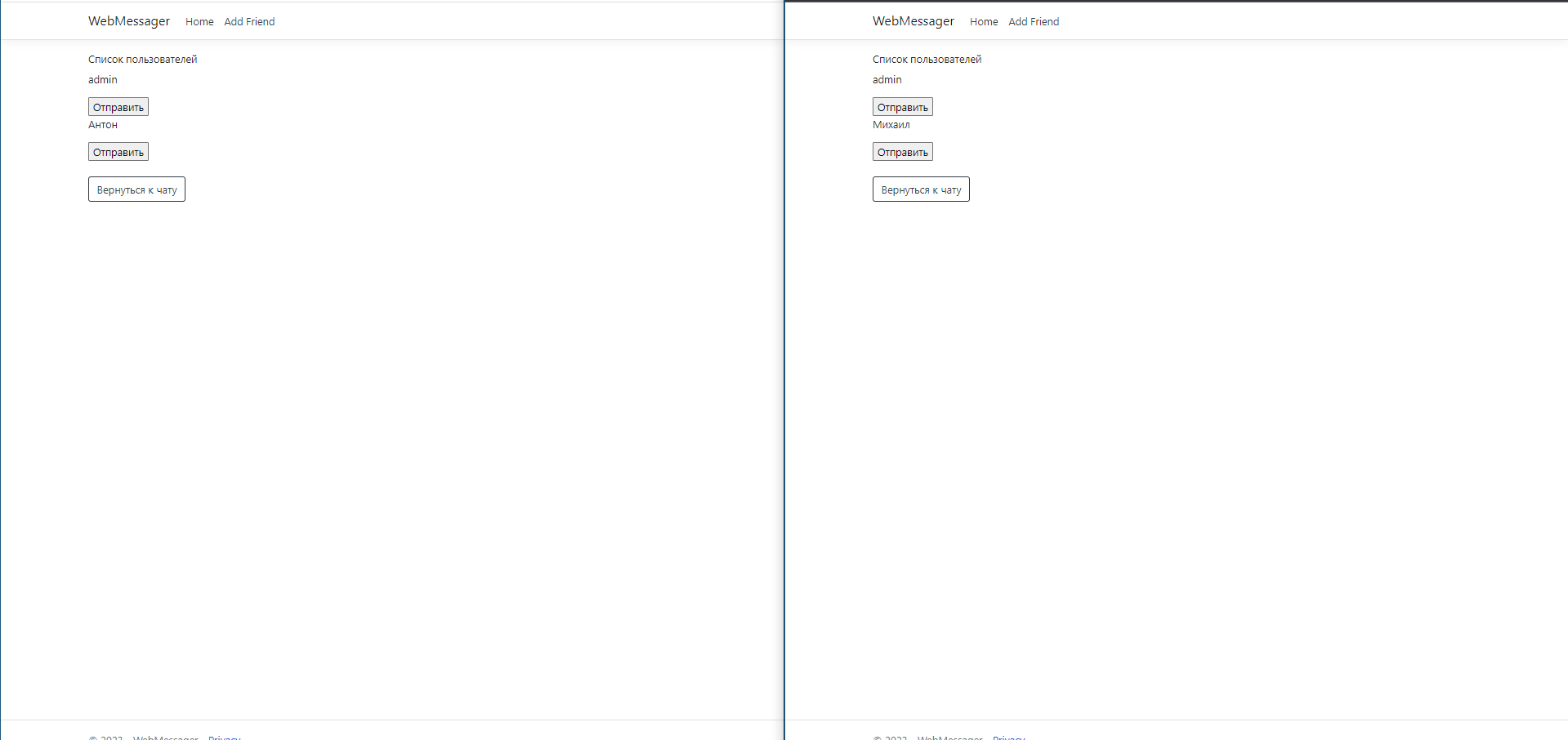


Рис.10. Страница добавления в друзья.

На рисунке №10 видная страница добавления в друзья, на которой можно отправить заявку в друзья другим пользователям или вернуться к чатам с другим пользователям.

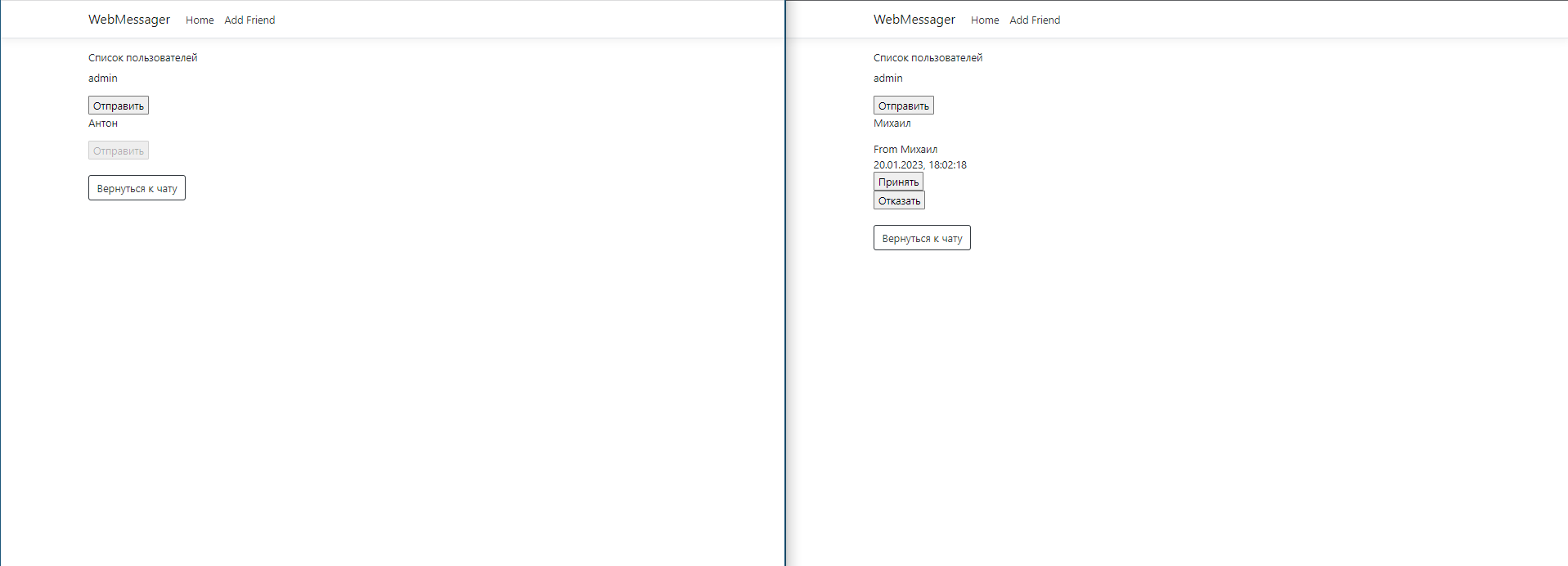


Рис.11. Отправленная заявка в друзья.

При отправке заявки в друзья у текущего пользователя (Рисунок №11) блокируется кнопка отправки, на сервер посылается запрос, который сервер обрабатывает, добавляет в БД состояние “Pending” между этими пользователями, а также возвращает другому пользователю две кнопки “Принять и “Отклонить”.

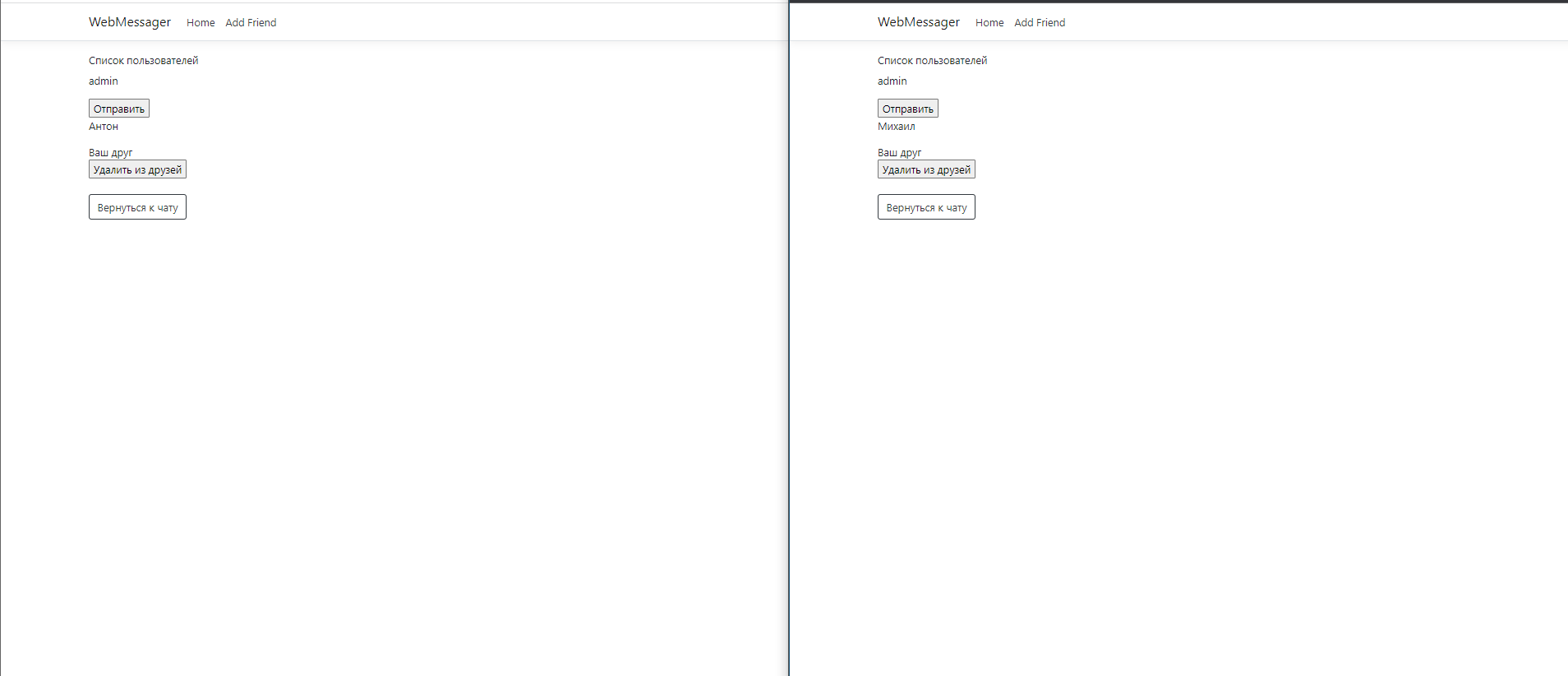


Рис.12. Принятая заявка в друзья.

Если пользователь, которому пришла заявка принимает её, то еще раз посылается запрос на сервер, сервер обрабатывает его, изменяет состояние в БД на “Approve”, и возвращает клиентам ответ (Рисунок №12). Также на главной странице появятся друзья у пользователей (Рисунок №13).

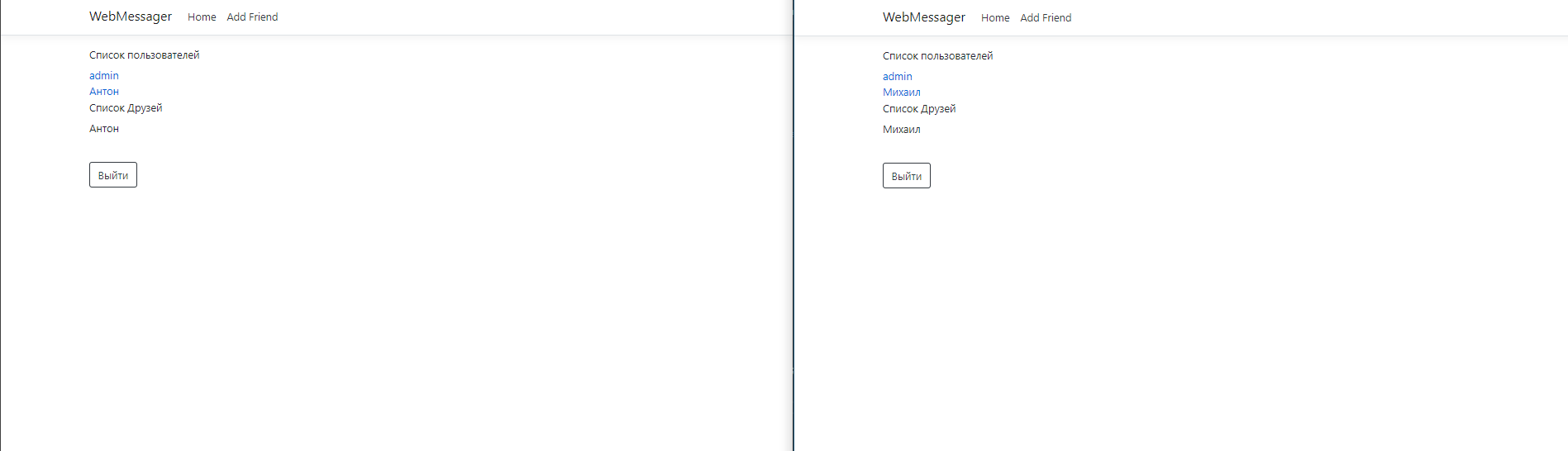


Рис.13. Главная страница после добавления в друзья.

При нажатии на кнопку удаления или кнопку отклонения запроса в друзья, сервер удалит данные из бд, и вернет начальную страницу с отправкой.

У администратора есть возможность удаления пользователей (Рисунок №14).

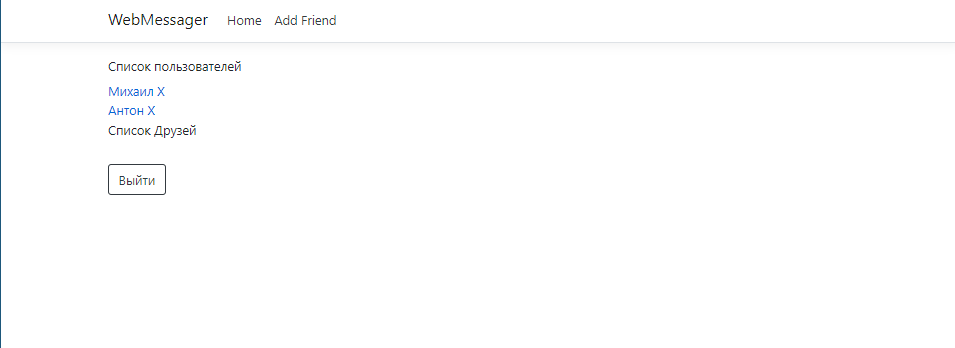


Рис.14. Главная страница администратора.

При удалении пользователя, удаляются все данные из БД, связанные с ним.

# заключение

В процессе выполнения курсового проекта была спроектирована и разработана часть программной системы для предметной области «Мессенджер».

В процессе разработки были изучены методы проектирования систем с использованием нотации UML.

Были получены навыки разработки сложных систем с использованием ASP.NET Core MVC на ЯП высокого уровня C#, взаимодействие с базой данных с помощью технологии Entity Framework Core, хранение и работа с СУБД SQL Server.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Липаев В.В. Проектирование программных систем. М.: Высш. шк, 1990.

2. Буч Г. Объектно-ориентированное проектирование / Пер. с анг. Конкорд, 1996.

3. Майерс Г. Надежность программного обеспечения. М.: Мир, 1980

4. ASP.NET Core Documentation [Электронный ресурс]: Microsoft – Режим доступа: <https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0>

5. Документация по Entity Framework [Электронный ресурс]: Microsoft – Режм доступа: <https://learn.microsoft.com/ru-ru/ef/>

ПРИЛОЖЕНИЕ А

**Листинг контроллеров и сервера.**

HomeController.cs:

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using WebMessager.Models;

using WebMessager.ViewModels;

namespace WebMessager.Controllers

{

public class HomeController : Controller

{

private readonly ILogger<HomeController> \_logger;

private MessagerContext db;

public HomeController(ILogger<HomeController> logger, MessagerContext context)

{

\_logger = logger;

db = context;

}

[Authorize]

public IActionResult Index()

{

return View();

}

[Authorize]

public IActionResult Friend()

{

return View();

}

[Authorize]

public IActionResult CreateGroupChat()

{

return View();

}

public IActionResult DeleteUser(int id)

{

var user = db.User.First(u => u.Id == id);

var friends = db.Friend.Where(f => f.User1Id == user.Id || f.User2Id == user.Id);

var mess = db.PrivateMessage.Where(f => f.UserFromId == user.Id || f.UserToId == user.Id);

db.User.Remove(user);

db.Friend.RemoveRange(friends);

db.PrivateMessage.RemoveRange(mess);

db.SaveChanges();

return Redirect("/");

}

[Authorize]

[Route("Home/Messages/{userId}")]

public IActionResult Messages(long userId)

{

var user = db.User.First(x => x.Id == userId);

var currentUserId = int.Parse(HttpContext.User.Claims.FirstOrDefault(x => x.Type == ClaimTypes.NameIdentifier).Value);

var Messages = db.PrivateMessage

.Where(x =>(x.UserFromId == userId && x.UserToId == currentUserId) || (x.UserFromId == currentUserId && x.UserToId == userId))

.Select(x => new PrivateMessageViewModel { Date = x.Date, From = x.UserFrom.Name, FromId = x.UserFromId, Text = x.Text, To = x.UserTo.Name, ToId = x.UserToId })

.ToList();

return View(new ChatViewModel

{

UserId = user.Id,

Messages = Messages

});

}

[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

public IActionResult Error()

{

return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });

}

}

}

AccountController.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Mvc;

using System.Security.Claims;

using WebMessager.ViewModels;

using WebMessager.Models;

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Authentication.Cookies;

namespace WebMessager.Controllers

{

public class AccountController : Controller

{

private MessagerContext db;

public AccountController(MessagerContext context)

{

db = context;

}

[HttpGet]

public IActionResult Login()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(LoginModel model)

{

if (ModelState.IsValid)

{

User user = await db.User.FirstOrDefaultAsync(u => u.Mail == model.Mail && u.Password == model.Password);

if (user != null)

{

await Authenticate(user.Id, model.Mail); // аутентификация

return RedirectToAction("Index", "Home");

}

ModelState.AddModelError("", "Некорректные логин и(или) пароль");

}

return View(model);

}

[HttpGet]

public IActionResult Register()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(RegisterModel model)

{

if (ModelState.IsValid)

{

User user = await db.User.FirstOrDefaultAsync(u => u.Mail == model.Mail);

if (user == null)

{

// добавляем пользователя в бд

user = new User { Mail = model.Mail, Password = model.Password, Name = model.Name, SecondName = model.SecondName, AboutMe = model.AboutMe, Role = UserRole.Default };

db.User.Add(user);

await db.SaveChangesAsync();

await Authenticate(user.Id, model.Mail); // аутентификация

return RedirectToAction("Login", "Account");

}

else

ModelState.AddModelError("", "Некорректные логин и(или) пароль");

}

return View(model);

}

private async Task Authenticate(long userId, string userName)

{

// создаем один claim

var claims = new List<Claim>

{

new Claim(ClaimTypes.NameIdentifier, userId.ToString()),

new Claim(ClaimsIdentity.DefaultNameClaimType, userName)

};

// создаем объект ClaimsIdentity

ClaimsIdentity id = new ClaimsIdentity(claims, "ApplicationCookie", ClaimsIdentity.DefaultNameClaimType, ClaimsIdentity.DefaultRoleClaimType);

// установка аутентификационных куки

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(id));

var user = HttpContext.User;

}

public async Task<IActionResult> Logout()

{

await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

return RedirectToAction("Login", "Account");

}

}

}

ChatHub.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Claims;

using Microsoft.AspNetCore.SignalR;

using System.Threading.Tasks;

using WebMessager.Controllers;

using WebMessager.SignalrModels;

using WebMessager.ViewModels;

using WebMessager.Models;

namespace WebMessager.Server

{

public class ChatHub : Hub

{

private MessagerContext db;

public ChatHub(MessagerContext context)

{

db = context;

}

/// <summary>

/// The dictionary

/// </summary>

private static Dictionary<string, long> Connections = new();

public async Task Send(MessageInfo mesInf)

{

var currentUserId = Connections[Context.ConnectionId];

var privateMessageViewModel = new PrivateMessageViewModel()

{

FromId = currentUserId,

ToId = mesInf.IdTo,

From = db.User.First(x => x.Id == currentUserId).Name,

To = db.User.First(x => x.Id == mesInf.IdTo).Name,

Date = DateTime.Now,

Text = mesInf.Text

};

try

{

db.PrivateMessage.Add(new PrivateMessage { Date = privateMessageViewModel.Date, UserFromId = privateMessageViewModel.FromId, UserToId = privateMessageViewModel.ToId, Text = privateMessageViewModel.Text });

await db.SaveChangesAsync();

}

catch (Exception e)

{

return;

}

var connectionIds = Connections.Where(x => x.Value == mesInf.IdTo || x.Value == currentUserId).Select(x => x.Key).ToList();

await Clients.Clients(connectionIds).SendAsync("Receive", privateMessageViewModel);

}

public async Task SendFriendRequest(FriendRequestInfo request)

{

var currentUserId = Connections[Context.ConnectionId];

var friendViewModel = new FriendViewModel()

{

FromId = currentUserId,

ToId = request.IdTo,

From = db.User.First(x => x.Id == currentUserId).Name,

To = db.User.First(x => x.Id == request.IdTo).Name,

Date = DateTime.Now,

Status = FriendStatus.Pending

};

db.Friend.Add(new Friend

{

User1Id = friendViewModel.FromId,

User2Id = friendViewModel.ToId,

Date = friendViewModel.Date,

Status = (int)FriendStatus.Pending

});

await db.SaveChangesAsync();

var friend = db.Friend.First(x => x.User1Id == friendViewModel.FromId && x.User2Id == friendViewModel.ToId);

friendViewModel.RelationshipId = friend.Id;

var connectionIds = Connections.Where(x => x.Value == request.IdTo).Select(x => x.Key).ToList();

await Clients.Clients(connectionIds).SendAsync("ReceiveReq", friendViewModel);

}

public async Task ApproveRequest(long relationshipId, bool status)

{

var fr = db.Friend.First(fr => fr.Id == relationshipId);

if(fr != null)

{

var connectionIds = Connections.Where(x => x.Value == fr.User1Id || x.Value == fr.User2Id).Select(x => x.Key).ToList();

if (status)

{

fr.Status = (int)FriendStatus.Approve;

db.Friend.Update(fr);

await db.SaveChangesAsync();

}

else

{

fr.Status = (int)FriendStatus.Decline;

db.Friend.Remove(fr);

await db.SaveChangesAsync();

}

await Clients.Clients(connectionIds).SendAsync("changeRelation", new FriendViewModel()

{

RelationshipId = fr.Id,

FromId = fr.User1Id.Value,

ToId = fr.User2Id.Value,

From = db.User.First(x => x.Id == fr.User1Id).Name,

To = db.User.First(x => x.Id == fr.User2Id).Name,

Date = fr.Date,

Status = (FriendStatus)fr.Status

});

}

}

public override Task OnConnectedAsync()

{

var httpContext = Context.GetHttpContext();

var currentUserId = int.Parse(httpContext.User.Claims.FirstOrDefault(x => x.Type == ClaimTypes.NameIdentifier).Value);

Connections.Add(Context.ConnectionId, currentUserId);

return base.OnConnectedAsync();

}

/// <summary>

/// Ons the disconnected using the specified exception

/// </summary>

/// <param name="exception">The exception</param>

public override Task OnDisconnectedAsync(Exception exception)

{

Connections.Remove(Context.ConnectionId);

return base.OnDisconnectedAsync(exception);

}

}

}

ПРИЛОЖЕНИЕ Б

