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 «Волонтерская служба»

Выполнил: Батаев.А.К.

Принял: доц. кафедры ИСПИ

Вершинин В.В.

Владимир, 2022

**<утвержденный Лист задания нА курсовой проект>**

**Аннотация**

В данном курсовом проекте разрабатывается программная система «Волонтерская служба». В процессе работы разрабатывается необходимый функционал, который позволяет упростить задачу поиска пропавших и координации волонтеров во время поисково-спасательных мероприятий. Были разработаны модели данных, база данных SQL, контроллеры и представления. В базе данных имеется ряд таблиц, заполненных данными. Помимо этого разработаны диаграммы прецедентов и состояний.

Курсовой проект представлен на 33 страницах, рисунков – 16, использованных источников – 4, приложений – 1. Иллюстрационный материал представлен на 5 листах формата А4.

**ANNOTANTION**

In this course project, a software system "Volunteer Service" is being developed. In the process of work, the necessary functionality is being developed, which makes it possible to simplify the task of searching for missing people and coordinating volunteers. Data models, SQL database, controllers and views were developed. The database has a table filled with data. In addition, use case and state diagrams have been developed.

The course project is presented on 33 pages, figures - 16, used sources - 4, appendices - 1. The illustrative material is presented on 5 sheets of A4 format.

**Содержание**

ВВЕДЕНИЕ 4

1 Постановка цели и задач 5

1.1. Цель 5

1.2. Задачи 5

2 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 7

2.1 Общие сведения о предмете автоматизации 7

2.2 Словарь предметной области 7

2.3 Сценарий взаимодействия пользователя с системой 8

3 ПРОЕКТИРОВАНИЕ СИСТЕМЫ 9

3.1 Диаграмма прецедентов 9

3.2 Описание прецедентов 9

3.3 Диаграмма классов 12

3.4 Диаграмма состояний 13

3.5 Диаграмма последовательностей 15

3.6 Диаграмма деятельности 15

4 Реализация СИСТЕМЫ 17

4.1. Принципы организации проекта 17

4.2. Реализация модели 17

4.3. Реализация аутентификации 17

4.4. Пользовательский интерфейс 18

заключение 19

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20

ПРИЛОЖЕНИЕ А Исходный код программной системы 21

ПРИЛОЖЕНИЕ Б Структура таблиц БД 22

# ВВЕДЕНИЕ

Заданием данной курсовой работы является проектирование и разработка программной системы «Волонтерская служба». Данная система предоставляет удобный инструментарий, значительно упрощающий работу и коммуникацию волонтеров во время поисково-спасательных операций.

Существенное преимущество построения веб-приложений заключается в том, что стандартные функции браузера должны выполняться независимо от операционной системы данного клиента. Веб-приложение состоит из клиентской и серверной части, тем самым реализуя технологию «клиент-сервер». Клиентская часть отвечает за реализацию пользовательского интерфейса.

База данных, реализованная при помощи СУБД MySQL, формирует запросы к серверу и обрабатывает ответы от него. Серверная часть получается запрос от клиента, выполняет необходимые операции, после чего формирует в ответ необходимую веб-страницу и отправляет её клиенту по сети с использованием протокола HTTP.

В ходе выполнения курсового проекта выполнена разработка проекта информационной системы с использованием языка UML. Реализация программного обеспечения с использованием технологий разработки ASP.NET Core 5. База данных проекта будет реализована при помощи языка MySQL.

# Постановка цели и задач

* 1. Цель

Целью данного курсового проекта является проектирование и разработка информационной системы «Волонтерская служба», которая предоставляет удобные инструменты для планирования, управления и координации во время поисково-розыскных мероприятий, проводимых с участием волонтерской службы.

* 1. Задачи

В процессе разработки информационной системы необходимо:

1. выполнить анализ и исследование предметной области;

2. разработать прототип ИС;

3. выполнить моделирование работы ИС;

4. разработать схему базы данных;

5. реализовать ИС с использованием выбранных средств и технологий.

Исходные данные:

1. язык моделирования UML;

2. платформа разработки ASP.NET 5 Core.

Функциональные требования, предъявляемые к разрабатываемой информационной системе:

1. регистрация пользователей с возможностью дальнейшей авторизации;

2. создание поисково-спасательной операции;

3. выдача определенных ролей участникам поисково-спасательной операции;

4. взаимодействие пользователей при помощи многопользовательского чата операции;

5. возможность оставить заявку на поиск;

6. возможность оставить заявку на становление волонтером.

Нефункциональные требования, предъявляемые к разрабатываемой информационной системе:

1. Система должна быть реализована на языке C# в виде веб-приложения.

# ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

* 1. **Общие сведения о предмете автоматизации**

Разрабатываемая система «Волонтерская служба» служит для упрощения процесса взаимодействия и координации между участниками поисково-спасательных мероприятий (волонтеров). Пользователями данной системы являются как волонтеры, так обычные люди, сочувствующие их делу. Для волонтеров эта система служит удобным инструментом взаимодействия, а для пользователей возможностью подать заявку на поиск или ознакомится с волонтерской деятельностью.

* 1. **Словарь предметной области** 
     1. **Волонтер**

*Волонтер* – человек добровольно участвующий в поисковой операции (розыске пропавших) в составе поисковой группы, не получающий за это вознаграждения.

* + 1. **Поисковая группа**

*Поисковая группа (поисковый отряд)* – группа волонтеров, во главе которой стоит командир группы, которая производит поисково-розыскные мероприятия на местности. Как правило, осматривая определенный район/опрашивая жителей.

* + 1. **Администратор**

*Администратор* – руководящая должность в волонтерской службе. Отвечает за принятие или отклонение заявок, вопросы организации и назначения и т.д.

* + 1. **Поисковая операция**

*Поисковая операция –* поисково-спасительные мероприятия, проводимые поисковым отрядом на местности, с целью нахождения пропавшего.

* + 1. **Область поиска**

*Область поиска* – район, где предположительно может быть обнаружен пропавший, или сведения, способствующие его нахождению.

* + 1. **Заявка на поиск**

*Заявка на поиск* – заявка, которую оставляет человек на сайте. Заявка рассматривается и консультируется диспетчером. Она содержит основную информацию о пропавшем: ФИО, возраст, особые приметы. Также содержит сведения о месте, где пропавшего видели в последний раз, и обстоятельствах в которых он исчез.

* + 1. **Пропавший**

*Пропавший* – человек, на которого оставлена заявка на поиск.

* + 1. **Незарегистрированный пользователь**

*Незарегистрированный пользователь* – человек, который хочет ознакомиться с волонтерским движением / списком пропавших / оставить заявку на поиск или вступление в волонтеры / передать информацию.

* 1. **Сценарий взаимодействия пользователя с системой**

Взаимодействие пользователя с системой осуществляется посредством интернет-браузера.

Пользователь изучает информацию на сайте Волонтерской службы о людях, которые разыскиваются в данный момент. Затем проходит процедуру регистрации. После этого он ознакомляется с информацией о волонтерской деятельности и решает стать волонтером. Пользователь оставляет заявку, чтобы стать волонтером и ожидает её рассмотрения администратором. После того как заявка будет одобрена, пользователь получит возможность присоединится к поисково-спасательной операции.

# ПРОЕКТИРОВАНИЕ СИСТЕМЫ

1. **Диаграмма прецедентов**

В процессе проектирования системы была разработана диаграмма прецедентов, описывающая систему на концептуальном уровне. Диаграмма прецедентов представлена на Рисунке 1.

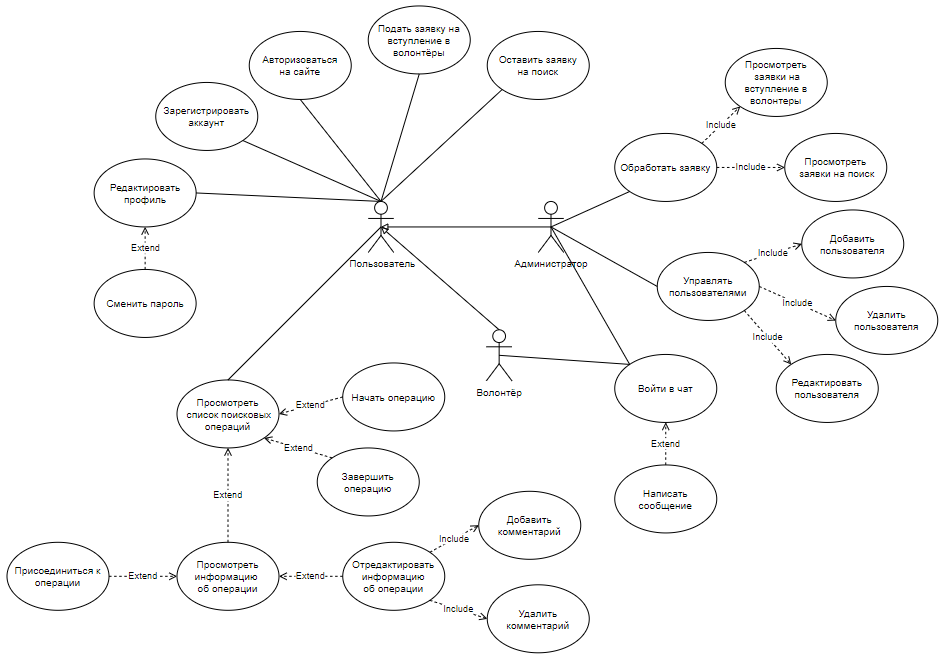
****

Рисунок 1 – Диаграмма прецедентов

1. **Описание прецедентов**

**Название прецедента:** «Оставить заявку на поиск»

**Предусловие:** пользователь ранее зарегистрирован и авторизован в системе и перешёл на главную страницу сайта.

**Действующее лицо:** пользователь.

**Основной поток:**

1. на главной странице пользователь нажимает на кнопку «Заявка на поиск» и перед ним открывается форма со следующими полями: ФИО, возраст, пол, область пропажи, время пропажи, дополнительная информация, фото;

2. пользователь заполняет форму требуемыми данными и нажимает на кнопку «Отправить», его автоматически возвращает на главную страницу.

**Альтернативный поток:** на шаге 2, если пользователь ввел невалидные данные, то рядом с неверно заполненными полями отображается сообщение об ошибке. В этом случае пользователю необходимо перепроверить данные и ввести в соответствующие поля корректные данные.

**Постусловие:** заявка на поиск отправляется в базу данных и получает статус «ожидание».

**Название прецедента**: «Зарегистрироваться»

**Предусловие:** зайти на сайт приложения

**Действующее лицо:** пользователь

**Основной поток:**

1. пользователь выбирает кнопку «Регистрация» на главной странице сайта;

2. перед ним открывается форма для ввода данных, содержащая следующие поля: электронный адрес, имя, фамилия, пароль, повторный пароль;

3. пользователь вводит учетные данные и нажимает кнопку «Регистрация», после чего попадает на главную страницу сайта.

**Альтернативный поток:** на шаге 3, если пользователь в качестве электронного адреса ввел имя, которое используется другим пользователем, либо ввёл невалидные данные, то ему высвечивается сообщение о соответствующей ошибке.

**Постусловие:** пользователь авторизирован в системе, и в базу данных добавляется новый пользователь.

**Название прецедента:** «Авторизоваться»

**Предусловие:** пользователь имеет аккаунт в системе, и находиться на главной странице

**Действующее лицо:** пользователь.

**Основной поток:**

1. пользователь нажимает на кнопку «Войти», перед ним открывается форма с полями email и пароль;

2. пользователь заполняет эти поля, нажимает на кнопку «Войти», и его перенаправляет на главную страницу. Если пользователь отметит флажок «Запомнить», то cookie для входа сохранятся после закрытия браузера.

**Альтернативный поток:** на шаге 2, если пользователь ввел невалидные данные в одно из ключевых полей, на экран выведутся соответствующие сообщения об ошибке. В этом случае пользователю необходимо перепроверить введённые данные и откорректировать их.

**Постусловие:** пользователь вошёл в систему.

**Название прецедента:** «Присоединиться к операции»

**Предусловие:** пользователь ранее зарегистрирован в системе, имеет статус волонтёра и авторизован в системе.

**Действующее лицо:** волонтёр.

**Основной поток:**

1. волонтёр на главной странице нажимает на кнопку «список операций», после чего попадает на страницу со списком текущих операций;

2. далее он выбирает одну из операций, и нажимает кнопку «Просмотреть», после чего он попадает на страницу с деталями операции;

3. затем волонтёр нажимает на кнопку «Присоединиться», страница обновляется, и кнопка больше не отображается.

**Альтернативный поток:** если пользователь уже присоединился к операции, то на шаге 3 кнопка «присоединиться» будет отсутствовать.

**Постусловие:** в базе данных пользователь добавляется в операцию и соответствующий чат, который доступен ему через меню профиля.

**Название прецедента:** «Добавить комментарий»

**Предусловие:** администратор ранее начал какую-либо операцию или присоединился к ней.

**Действующее лицо:** администратор.

**Основной поток:**

1. администратор через главную страницу открывает список операций;

2. далее он выбирает определённую операцию, к которой ранее он присоединился или начал её;

3. затем администратор вводит в текстовое поле с подписью «Добавить информацию» и нажимает на кнопку «Добавить».

**Альтернативный поток:** если на шаге 3 администратор оставляет поле ввода пустым и пытается оставить комментарий, то ему выводится соответствующее сообщение об ошибке.

**Постусловие:** в базе данных добавляется комментарий в соответствующую таблицу, а также комментарий отображается на странице деталей соответствующей операции.

1. **Диаграмма классов**

Для проектирования структур и моделей данных программной системы была разработана диаграмма классов. На ней представлены основные модели данных, входящие в контекст данных приложения, а также контроллеры, обрабатывающие запросы и манипулирующие с данными приложения. Диаграмма классов представлена на рисунке 2.

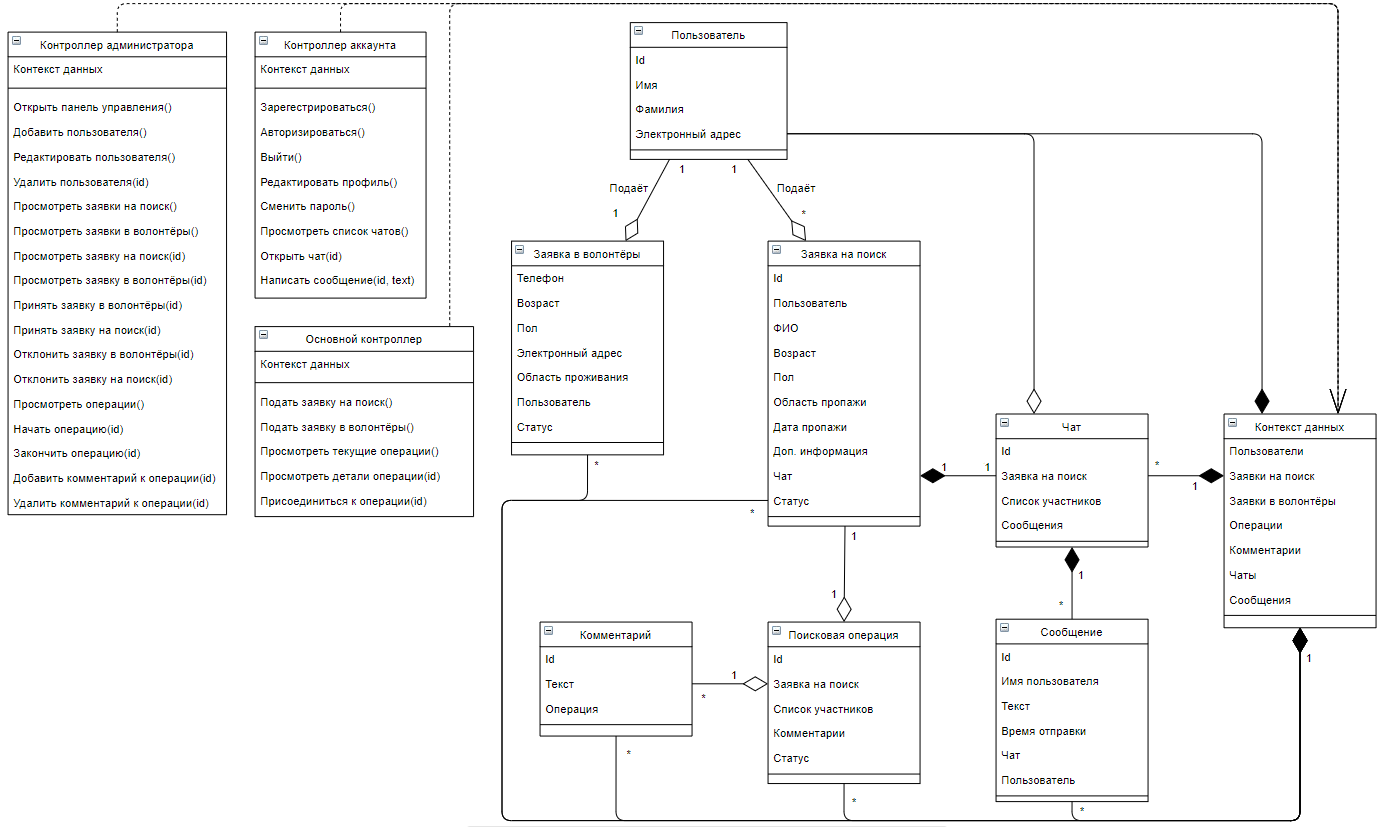


Рисунок 2 – Диаграмма классов

1. **Диаграмма состояний**

В процессе проектирования системы были разработаны диаграммы состояний, которые отображают основные состояния заявок и поисковой операции. Диаграммы состояний представлены на Рисунках 3 и 4.

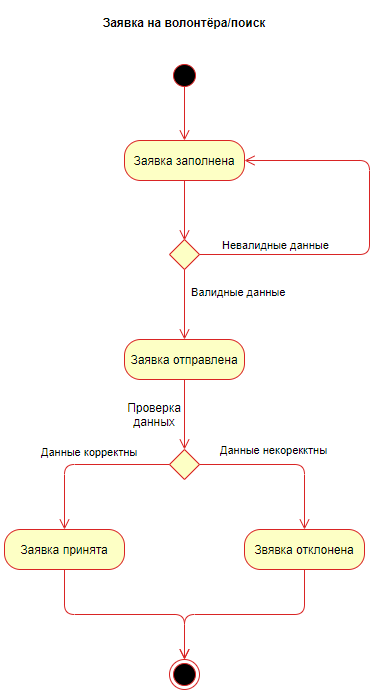
****

Рисунок 3 – Диаграмма состояний

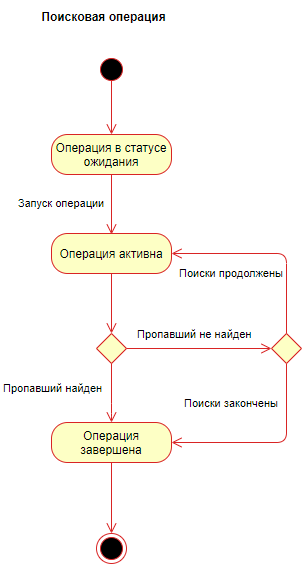
****

Рисунок 4 – Диаграмма состояний

1. **Диаграмма последовательностей**

В процессе проектирования системы была разработана диаграмма последовательностей. Она отображает процесс взаимодействия волонтёра с системой в процессе присоединения к поисковой операции. Данная диаграмма последовательностей представлена на рисунке 5.

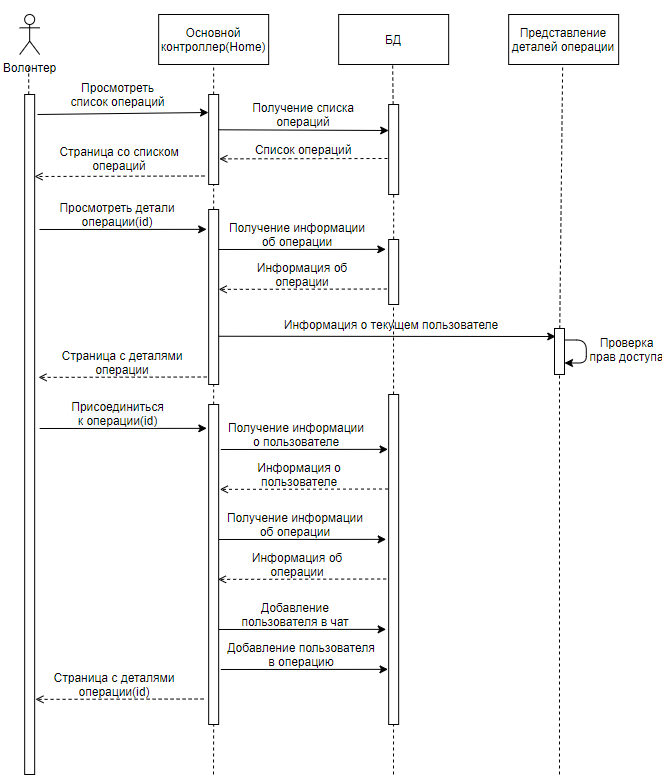
****

Рисунок 5 – Диаграмма последовательностей

1. **Диаграмма деятельности**

В процессе проектирования системы была разработана диаграмма деятельности. Она отображает процесс присоединения обычного пользователя к операции.

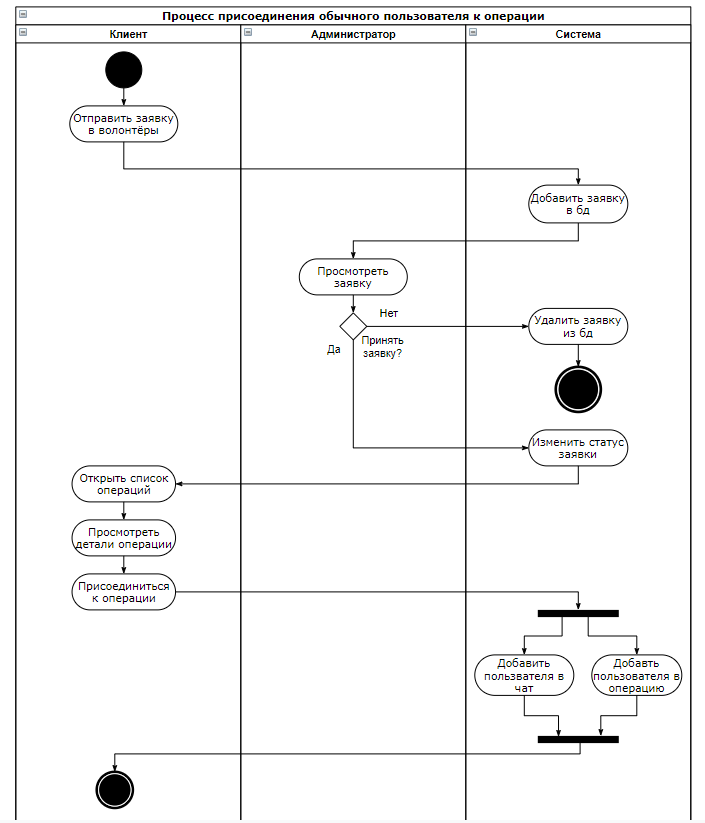


Рисунок 6 – Диаграмма последовательностей

# Реализация СИСТЕМЫ

## Принципы организации проекта

Приложение построено как Multi-page web Application. Для создания разметки представлений использовался фреймворк Bootstrap 5, для написания CSS-стилей представлений использовался препроцессор SASS. Для добавления слайдера на главную страницу использовалась библиотека swiper.js. Back-end приложения разработан на ASP.NET Core 5, в качестве СУБД применялся MS SQL. В качестве архитектуры приложения был выбран шаблон MVC, который представлен в ASP.NET Core платформой ASP.NET Core MVC. Для работы с БД был выбран ORM EntityFramework Core. Как система контроля версий использовался Git, в качестве хранилища репозитория проекта использовался GitHub.

## Реализация модели

Модель построена по принципу Code First – такой подход основан на том, что основной упор при реализации хранения данных делается на код, а не на БД. Это означает то, что сначала создавались модели, и уже по ним с помощью добавления миграции строилась БД.

## Реализация аутентификации

Для аутентификации и авторизации пользователей была использована система ASP.NET Core Identity, которая представляет удобное средство управления пользователями, их паролями и ролями. Для хранения соответствующих данных используется SQL Server. Основные методы, связанные с профилем пользователя, находятся в контроллере AccountController. Для первоначальной инициализации системы ролями использовался класс RoleInitializer с одним статическим методом.

## Пользовательский интерфейс

Разработанный пользовательский интерфейс представлен на рисунках 6-15.

Главная страница с основной информацией и слайдером с пропавшими людьми представлена на рисунках 6-7. Информация в слайдере автоматически изменяется при изменении информации в базе данных.

Кнопки «Профиль» и «Выход» заменяются на «Регистрация» и «Войти», если пользователь не авторизован в системе.

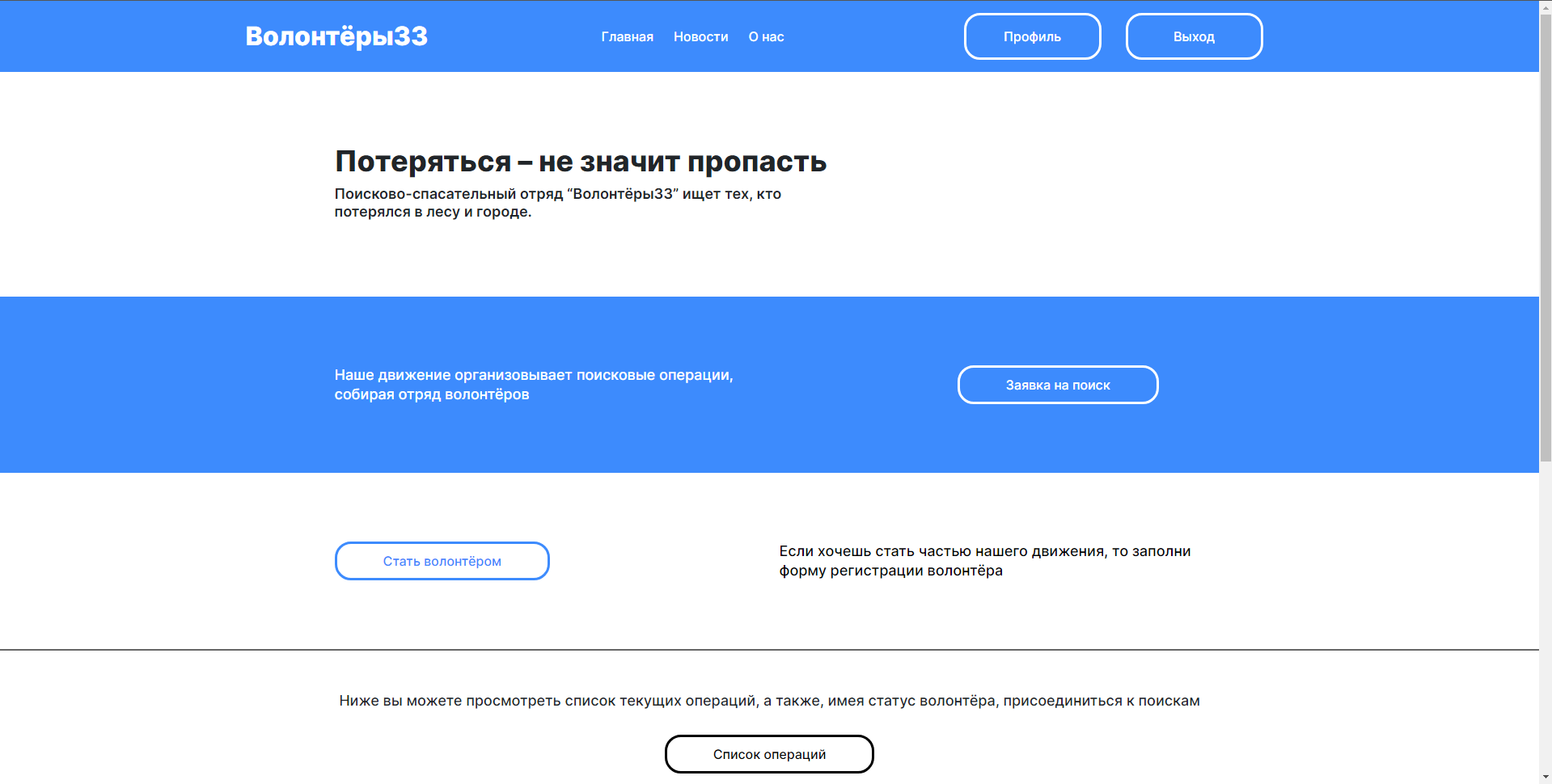


Рисунок 6 – Главная страница

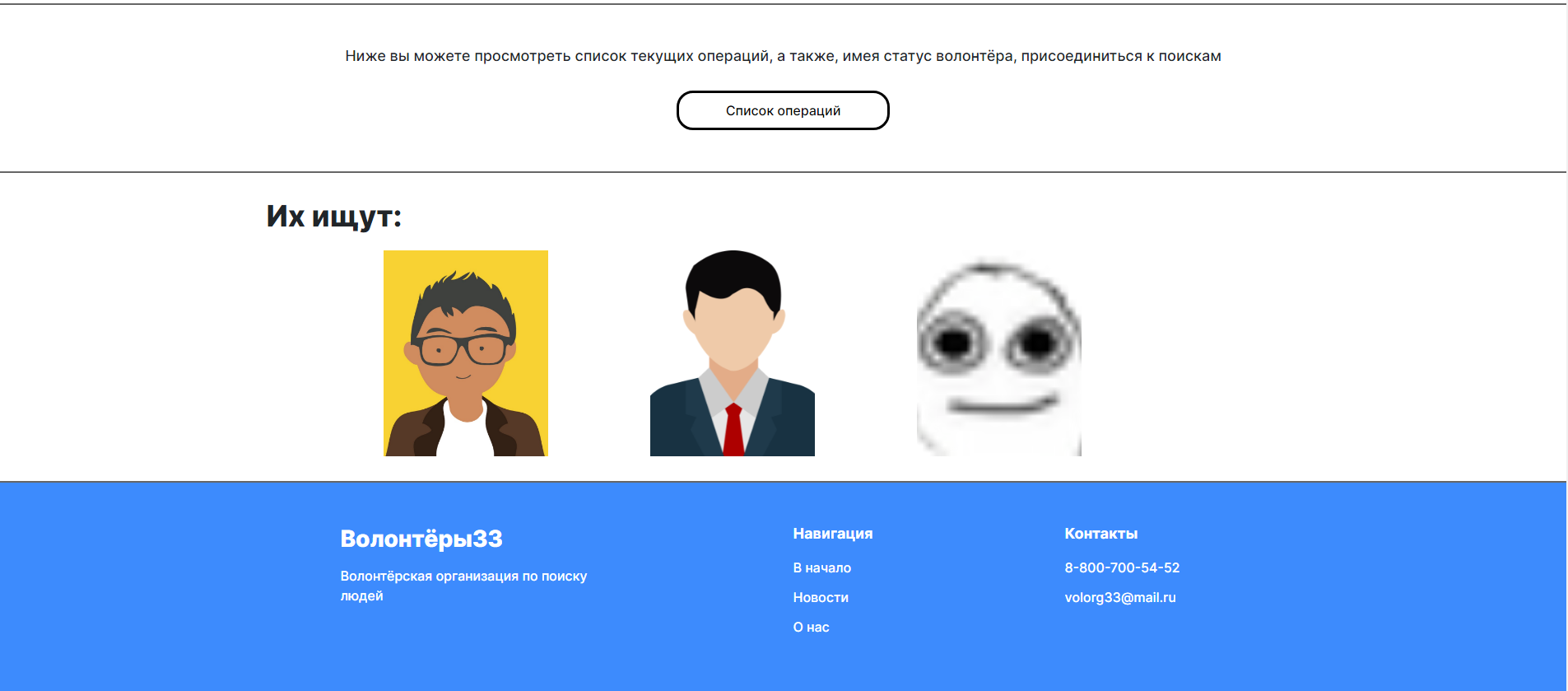


Рисунок 7 – Главная страница

Формы регистрации и авторизации пользователя представлены на рисунке 8.

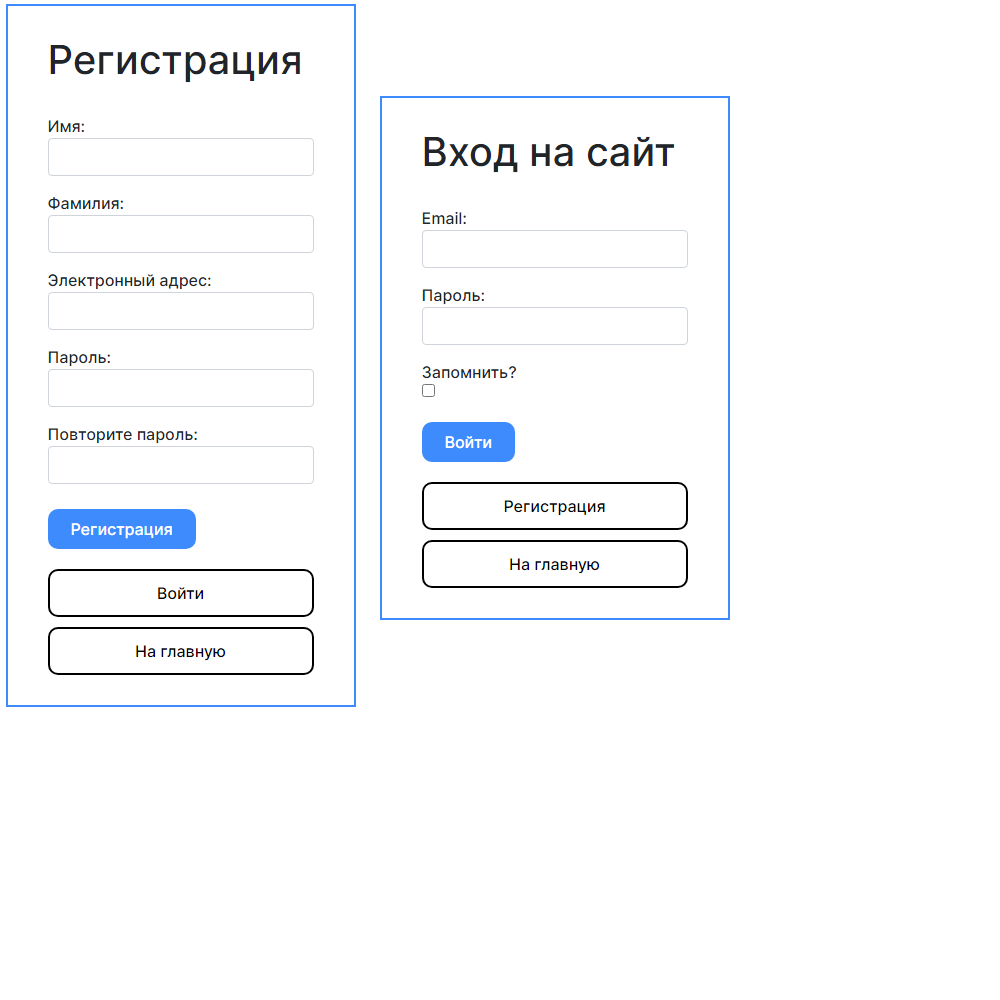


Рисунок 8 – Формы регистрации и авторизации

Список операций представлен на рисунке 9.

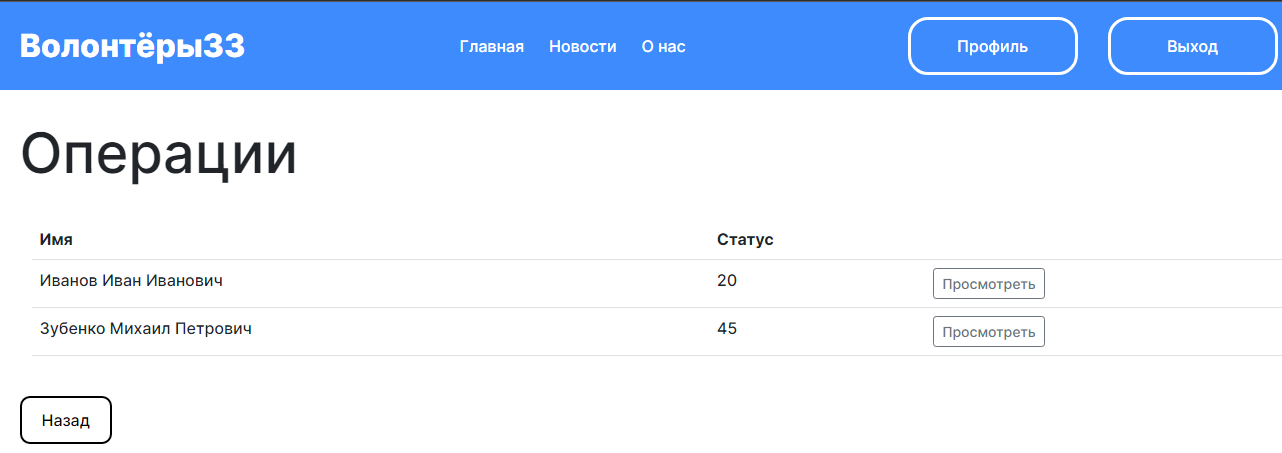


Рисунок 9 – Формы подачи заявки на поиск и на становление волонтёром

Детали операции, а также поле добавление комментария, доступное для администраторов, представлены на рисунке 10. Также для волонтёра, который не состоит в этой операции, доступна функция присоединения к данной операции.

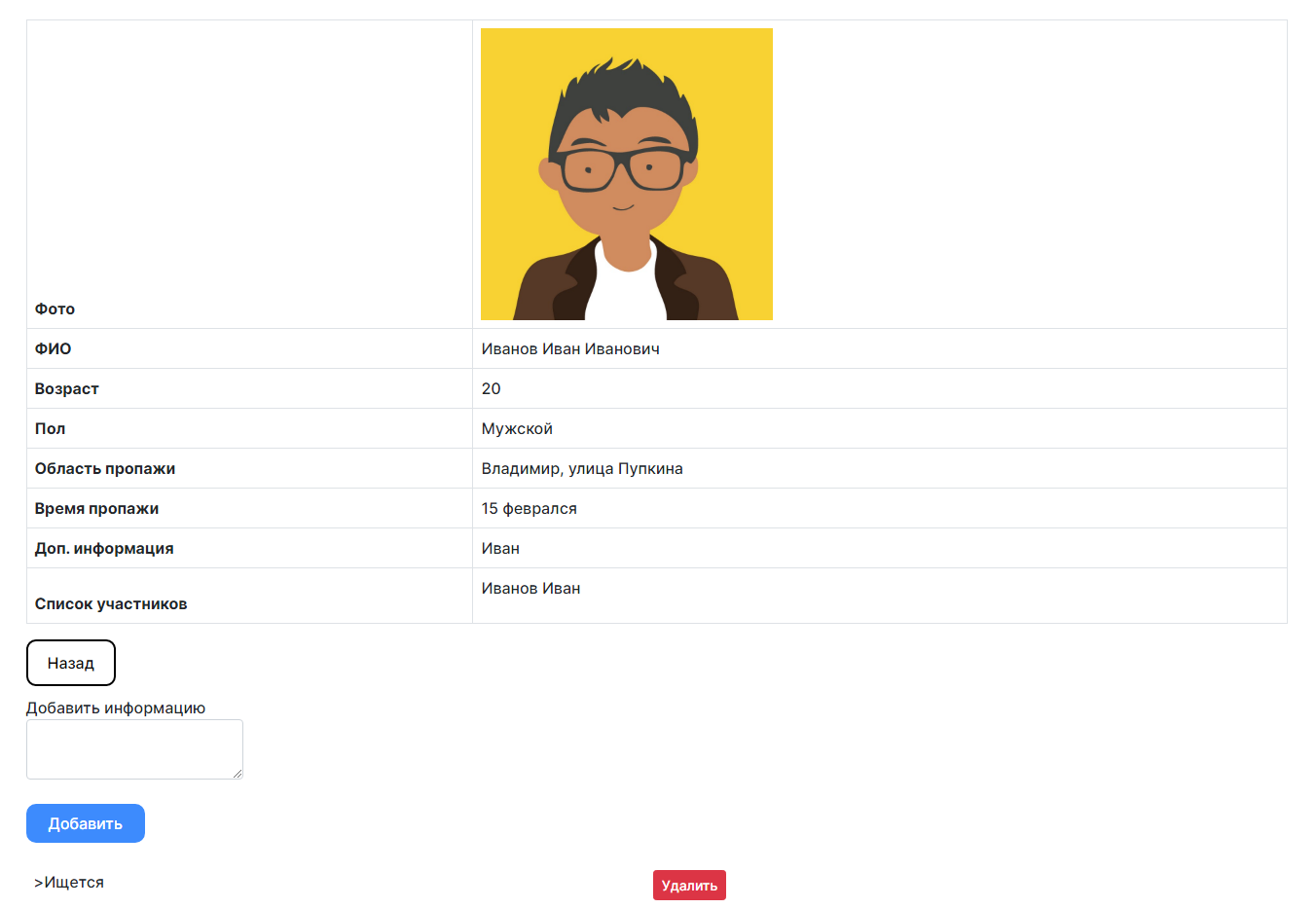


Рисунок 10 – Детали операции

Интерфейс профилей администратора и волонтёра представлены на рисунке 11. Из меню профиля волонтёр может перейти к списку текущих чатов, а администратор ещё дополнительно в панель управления. Также через это окно пользователю доступны редактирование профиля и смена пароля.

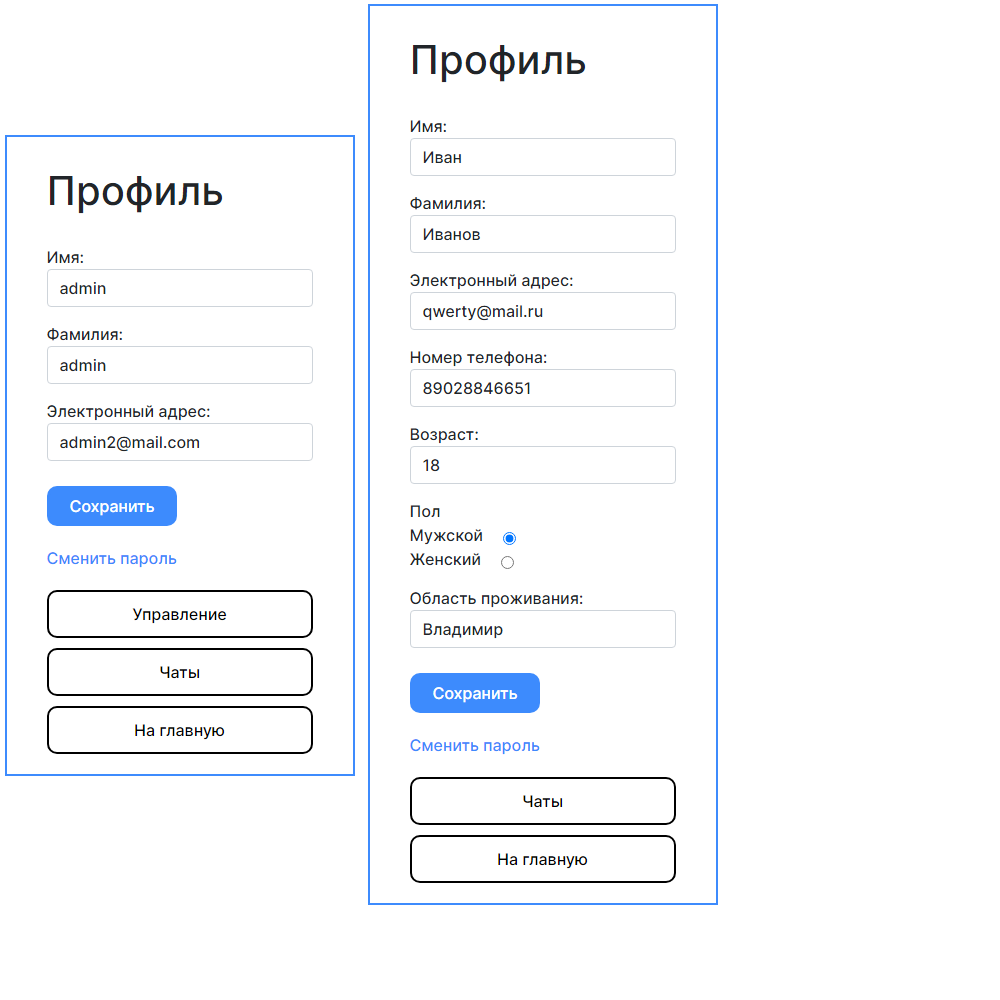


Рисунок 11 – Интерфейс профилей

Форма для изменения пароля представлена на рисунке 12.

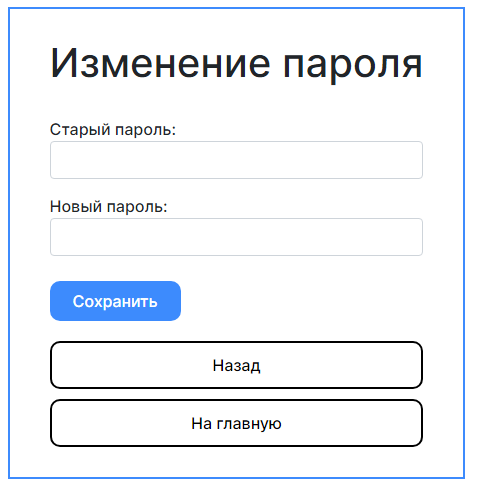


Рисунок 12 – Форма изменения пароля

Список текущих чатов и сам чат представлены на рисунке 13. Этот чат представляет средство информирования участников операции.

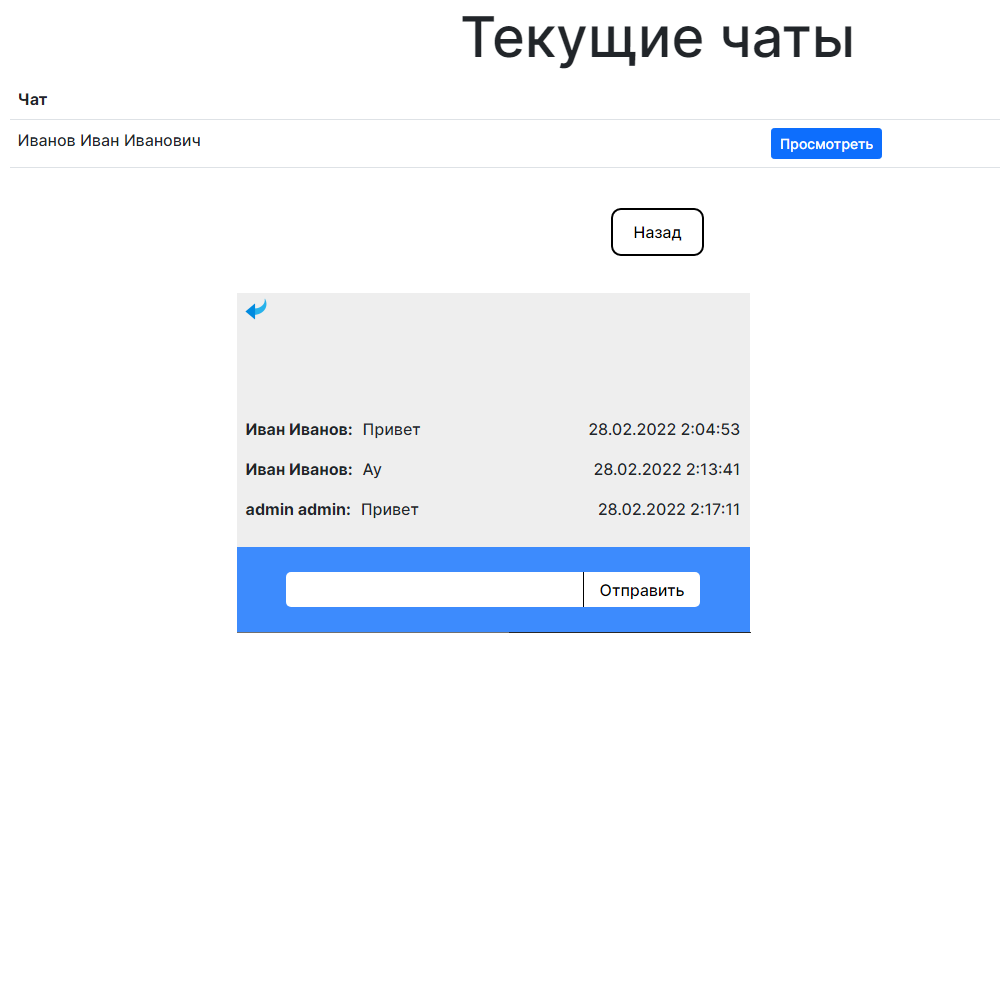


Рисунок 13 – Список чатов и чат

Интерфейс панели администратора представлен на рисунке 14. На этой панели доступны основные функции: добавление, редактирование и удаление пользователя; управление заявками и операциями.

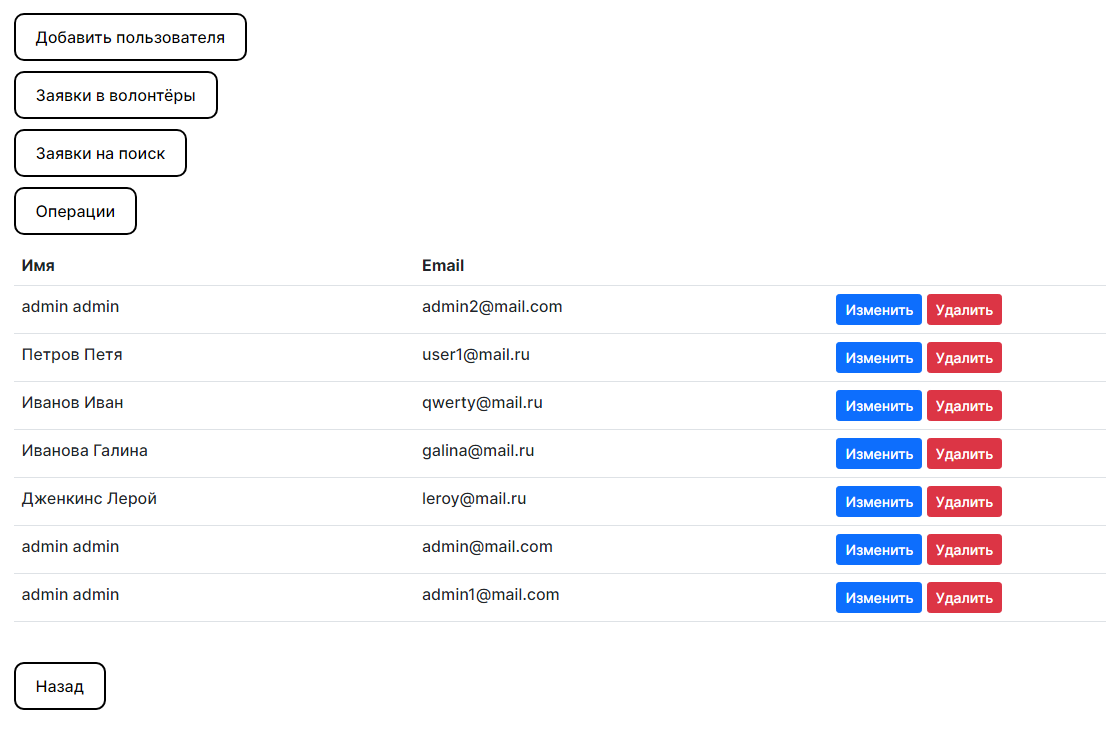


Рисунок 14 – Интерфейс администраторской панели

Администраторская панель управления операциями представлена на рисунке 15. Здесь администратор может начать операцию, связанную с принятой заявкой на поиск и завершить операцию, удалив связанный с операцией чат.

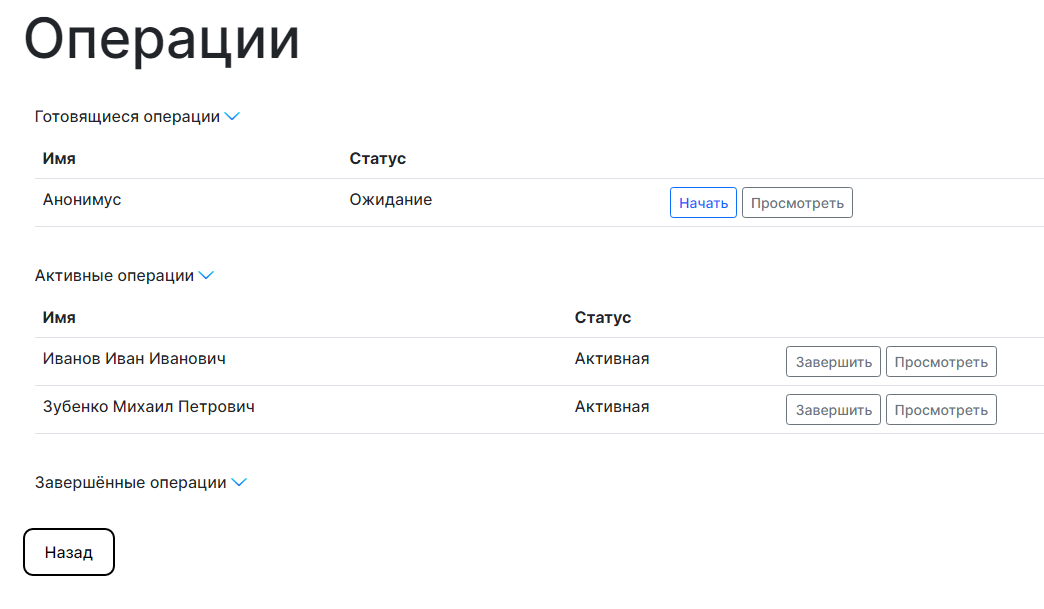


Рисунок 15 – Управление операциями

Интерфейс обработки заявки представлен на рисунке 16. Администратор может принять или отклонить заявку. При принятии заявки на поиск, заявка получает статус «Принята», при принятии заявки в волонтёры, соответствующему пользователю присваивается роль волонтёра. При отклонении заявки она удаляется из базы данных.

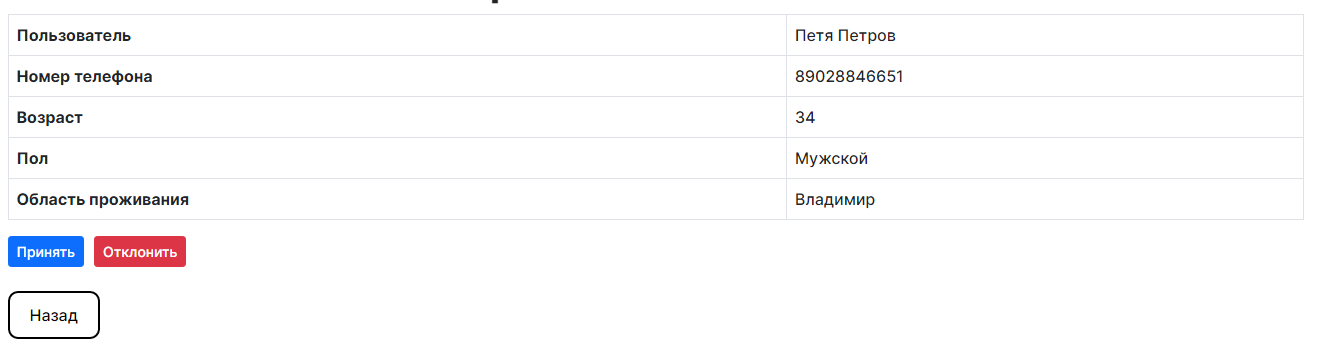


Рисунок 16 – Интерфейс обработки заявки

# заключение

В результате выполнения курсового проекта была спроектирована и реализована программная система «Волонтерская служба». Система, ориентированная на упрощение работы волонтерских организаций, занимающихся проведением поисково-спасательных мероприятий, удобна на практике. Она позволяет производить различные операции над данными, обеспечивая их целостность и сохранность. Инструментарий, представленный в системе, предоставляет возможность просматривать, редактировать данные, что значительно автоматизирует и упрощает процесс проведения поисково-спасательных операций.

В процессе выполнения курсового проекта были выполнены задачи:

1. анализа предметной области и формирования требований к системе;
2. разработки проекта программной системы;
3. реализации системы, согласно подготовленному проекту.

Разработка проекта системы была выполнена на языке UML. Программное обеспечение было реализовано на объектно-ориентированном языке программирования - C# в виде веб-приложения, построенного на базе платформы ASP.NET 5 Core.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Фримен А. ASP.NET Core MVC 2 с примерами на C# для профессионалов. 7-е издание, 2019.
2. Руководство по ASP.NET Core 5 [Электронный ресурс] – Режим доступа: https://metanit.com/sharp/aspnet5/ (дата обращения: 10.02.2021)
3. Полное руководство по языку программирования С# 10 и платформе .NET 6 [Электронный ресурс] – Режим доступа: https://metanit.com/sharp/tutorial/ (дата обращения: 10.02.2021)
4. Сетевое программирование в С# и .NET [Электронный ресурс] – Режим доступа: https://metanit.com/sharp/net/ (дата обращения: 10.02.2021)

ПРИЛОЖЕНИЕ А Исходный код программной системы

**Модели:**

**ApplicationContext.cs**

namespace MVC.Models

{

public class ApplicationContext : IdentityDbContext<User>

{

public DbSet<SearchRequest> SearchRequests { get; set; }

public DbSet<VolRequest> VolRequests { get; set; }

public DbSet<Operation> Operations { get; set; }

public DbSet<Comment> Comments { get; set; }

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

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

public ApplicationContext(DbContextOptions<ApplicationContext> options)

: base(options)

{

Database.EnsureCreated(); // создаем базу данных при первом обращении

}

}

}

**User.cs**

namespace MVC.Models

{

public class User : IdentityUser

{

[Required(ErrorMessage = "Не указано имя")]

public string Name { get; set; }

[Required(ErrorMessage = "Не указана фамилия")]

public string SurName { get; set; }

public SearchRequest SearchRequest { get; set; }

public VolRequest VolRequest { get; set; }

public List<Operation> Operations { get; set; } = new List<Operation>();

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

public User() { }

}

}

**Chat.cs**

namespace MVC.Models

{

public class Chat

{

public int Id { get; set; }

public SearchRequest SearchRequest { get; set; }

public int SearchRequestId { get; set; }

public List<Message> Messages { get; set; } = new List<Message>();

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

}

}

**Comment.cs**

namespace MVC.Models

{

public class Comment

{

public int Id { get; set; }

[Required]

public string Text { get; set; }

public Operation Operation { get; set; }

public int OperationId { get; set; }

}

}

**Message.cs**

using System;

namespace MVC.Models

{

public class Message

{

public int Id { get; set; }

public string UserName { get; set; }

public string Text { get; set; }

public DateTime Timestamp { get; set; }

public string UserId { get; set; }

public User User { get; set; }

public int ChatId { get; set; }

public Chat Chat { get; set; }

}

}

public class Operation

{

public int Id { get; set; }

[Required]

public string Status { get; set; }

[Required]

public int? SearchRequestId { get; set; }

[Required]

public SearchRequest SearchRequest { get; set; }

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

public List<Comment> Comments { get; set; } = new List<Comment>();

}

**Контроллеры:**

**AccountController.cs**

namespace MVC.Controllers

{

public class AccountController : Controller

{

private readonly UserManager<User> \_userManager;

private readonly SignInManager<User> \_signInManager;

private readonly ApplicationContext \_db;

public AccountController(UserManager<User> userManager, SignInManager<User> signInManager, ApplicationContext db)

{

\_userManager = userManager;

\_signInManager = signInManager;

\_db = db;

}

[HttpGet]

public IActionResult Register()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Register(RegisterModel model)

{

if (ModelState.IsValid)

{

User user = new User { Name = model.Name, SurName = model.SurName, Email = model.Email, UserName = model.Email };

// добавляем пользователя в бд

var result = await \_userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

// установка роли

await \_userManager.AddToRoleAsync(user, "user");

// установка куки

await \_signInManager.SignInAsync(user, false);

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

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

return View(model);

}

[HttpGet]

public IActionResult Login(string returnUrl = null)

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(LoginModel model)

{

if (ModelState.IsValid)

{

var result =

await \_signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, false);

if (result.Succeeded)

{

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

}

else

{

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

}

}

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Logout()

{

// удаляем аутентификационные куки

await \_signInManager.SignOutAsync();

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

}

[Authorize]

public async Task<IActionResult> EditProfile()

{

User user = await \_userManager.GetUserAsync(User);

if (user == null)

{

return NotFound();

}

if (User.IsInRole("volunteer"))

{

VolRequest volReq = await \_db.VolRequests.FirstOrDefaultAsync(v => v.UserId == user.Id);

VolunteerProfileViewModel model = new VolunteerProfileViewModel { Email = user.Email, Name = user.Name, SurName = user.SurName, Phone = volReq.Phone, Age = volReq.Age,

Sex = volReq.Sex, LivArea = volReq.LivArea };

return View("EditVolunteerProfile", model);

}

else

{

UserProfileViewModel model = new UserProfileViewModel { Email = user.Email, Name = user.Name, SurName = user.SurName };

return View(model);

}

}

[Authorize]

[HttpPost]

public async Task<IActionResult> EditProfile(EditUserViewModel model)

{

if (ModelState.IsValid)

{

User user = await \_userManager.GetUserAsync(User);

if (user != null)

{

user.Email = model.Email;

user.UserName = model.Email;

user.Name = model.Name;

user.SurName = model.SurName;

var result = await \_userManager.UpdateAsync(user);

if (result.Succeeded)

{

return View(model);

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

}

return View(model);

}

public IActionResult ChangePassword()

{

return View(new ChangePasswordViewModel());

}

[HttpPost]

public async Task<IActionResult> ChangePassword(ChangePasswordViewModel model)

{

if (ModelState.IsValid)

{

User user = await \_userManager.FindByIdAsync(\_userManager.GetUserId(User));

if (user != null)

{

IdentityResult result =

await \_userManager.ChangePasswordAsync(user, model.OldPassword, model.NewPassword);

if (result.Succeeded)

{

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

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError(string.Empty, error.Description);

}

}

}

else

{

ModelState.AddModelError(string.Empty, "Пользователь не найден");

}

}

return View(model);

}

[Authorize(Roles = "admin,volunteer")]

public async Task<IActionResult> GetChats()

{

User user = await \_userManager.GetUserAsync(User);

return View(\_db.Chats.Include(c => c.SearchRequest).Where(c => c.Users.Contains(user)));

}

[Authorize(Roles = "admin,volunteer")]

public async Task<IActionResult> OpenChat(int id)

{

Chat chat = await \_db.Chats.Include(c => c.Messages).FirstOrDefaultAsync(c => c.Id == id);

return View(chat);

}

[Authorize(Roles = "admin,volunteer")]

public async Task<IActionResult> CreateMessage(int id, string text)

{

Chat chat = await \_db.Chats.FirstOrDefaultAsync(c => c.Id == id);

User user = await \_userManager.GetUserAsync(User);

Message message = new Message() { ChatId = id, Text = text, Timestamp = DateTime.Now, UserName = $"{user.Name} {user.SurName}", UserId = user.Id };

chat.Messages.Add(message);

\_db.Messages.Add(message);

await \_db.SaveChangesAsync();

return RedirectToAction("OpenChat", "Account", message.Chat);

}

}

}

**HomeController.cs**

namespace MVC.Controllers

{

public class HomeController : Controller

{

private readonly UserManager<User> \_userManager;

private readonly ApplicationContext \_db;

private readonly ILogger<HomeController> \_logger;

IWebHostEnvironment \_appEnvironment;

public HomeController(ILogger<HomeController> logger, UserManager<User> userManager, ApplicationContext db, IWebHostEnvironment appEnvironment)

{

\_logger = logger;

\_userManager = userManager;

\_db = db;

\_appEnvironment = appEnvironment;

}

public IActionResult Index()

{

return View(\_db.Operations.Include(o => o.SearchRequest).Where(o => o.Status == "Активная" || o.Status == "Ожидание"));

}

[HttpGet]

public IActionResult VolReq()

{

if (!User.IsInRole("user"))

{

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

}

return View();

}

[HttpPost]

[Authorize(Roles = "user")]

public async Task<IActionResult> VolReq(VolRequestViewModel model)

{

if (ModelState.IsValid)

{

VolRequest volReq = new VolRequest { Age = model.Age, Sex = model.Sex, LivArea = model.LivArea, Phone = model.Phone, Status = "Ожидание", UserId = \_userManager.GetUserId(User)};

// добавляем заявку в бд

\_db.VolRequests.Add(volReq);

await \_db.SaveChangesAsync();

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

}

return View(model);

}

[HttpGet]

public IActionResult SearchReq()

{

if (!User.IsInRole("user"))

{

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

}

return View();

}

[HttpPost]

[Authorize(Roles = "user")]

public async Task<IActionResult> SearchReq(SearchRequestViewModel model)

{

if (ModelState.IsValid)

{

string path = "/imgsearch/" + model.Photo.FileName;

// сохраняем файл в папку Files в каталоге wwwroot

using (var fileStream = new FileStream(\_appEnvironment.WebRootPath + path, FileMode.Create))

{

await model.Photo.CopyToAsync(fileStream);

}

SearchRequest searchReq = new SearchRequest {FullName = model.FullName, Age = model.Age, Sex = model.Sex, MissArea = model.MissArea, MissTime = model.MissTime, AddInf = model.AddInf, Photo = path, Status = "Ожидание", UserId = \_userManager.GetUserId(User) };

// добавляем заявку в бд

\_db.SearchRequests.Add(searchReq);

await \_db.SaveChangesAsync();

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

}

return View(model);

}

public IActionResult CheckOperations()

{

return View(\_db.Operations.Include(o => o.SearchRequest).ToList());

}

public async Task<IActionResult> OperationDetails(int id)

{

Operation operation = await \_db.Operations.Include(o => o.SearchRequest).Include(o => o.Users).Include(o => o.Comments).FirstOrDefaultAsync(o => o.Id == id);

if (operation == null)

{

return NotFound();

}

ViewBag.User = await \_userManager.GetUserAsync(User);

return View(operation);

}

[Authorize(Roles = "volunteer")]

public async Task<IActionResult> JoinOperation(int id)

{

User user = await \_userManager.GetUserAsync(User);

Operation operation = await \_db.Operations

.Include(o => o.SearchRequest)

.ThenInclude(s => s.Chat)

.ThenInclude(c => c.Users)

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

if (user == null || operation == null)

{

return NotFound();

}

operation.SearchRequest.Chat.Users.Add(user);

operation.Users.Add(user);

await \_db.SaveChangesAsync();

return RedirectToAction("OperationDetails", "Home", new { id = id });

}

}

}

**AdminController.cs**

[Authorize(Roles = "admin")]

public class AdminController : Controller

{

private readonly UserManager<User> \_userManager;

private readonly ApplicationContext \_db;

IWebHostEnvironment \_appEnvironment;

public AdminController(UserManager<User> userManager, ApplicationContext db, IWebHostEnvironment appEnvironment)

{

\_userManager = userManager;

\_db = db;

\_appEnvironment = appEnvironment;

}

[HttpGet]

public ActionResult GetOperations()

{

return View(\_db.Operations.Include(o => o.SearchRequest).Include(o => o.Users).ToList());

}

[HttpGet]

public async Task<ActionResult> StartOperation(int id)

{

Operation operation = await \_db.Operations.FirstOrDefaultAsync(o => o.Id == id);

operation.Status = "Активная";

await \_db.SaveChangesAsync();

return RedirectToAction("GetOperations");

}

[HttpGet]

public async Task<ActionResult> StopOperation(int id)

{

Operation operation = await \_db.Operations

.Include(o => o.SearchRequest)

.ThenInclude(s => s.Chat)

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

operation.Status = "Завершённая";

Chat chat = await \_db.Chats.FirstOrDefaultAsync(c => c.Id == operation.SearchRequest.Chat.Id);

\_db.Chats.Remove(chat);

await \_db.SaveChangesAsync();

return RedirectToAction("GetOperations");

}

[HttpPost]

public async Task<ActionResult> CreateComment(Comment comment, int operationId)

{

comment.OperationId = operationId;

\_db.Comments.Add(comment);

Operation operation = await \_db.Operations.FirstOrDefaultAsync(o => o.Id == operationId);

operation.Comments.Add(comment);

await \_db.SaveChangesAsync();

return RedirectToAction("OperationDetails", "Home", operation);

}

public async Task<ActionResult> DeleteComment(int id)

{

Comment comm = await \_db.Comments.Include(c => c.Operation).FirstOrDefaultAsync(c => c.Id == id);

\_db.Comments.Remove(comm);

await \_db.SaveChangesAsync();

return RedirectToAction("OperationDetails", "Home", comm.Operation);

}

}