МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»**

Институт ИТКН

**Кафедра инженерной кибернетики**

Направление подготовки: 01.03.04 прикладная математика

Квалификация (степень): бакалавр

Группа: **БПМ-17-2**

**КУРСОВАЯ РАБОТА**

**на тему:**

*Разработка прототипа корпоративной социальной сети*

Учебная дисциплина «Специальные главы баз данных»

VIII семестр 2020 – 2021 у.г.

**Учащийся: Скибин К.С.**

**Проверил: доц., к.т.н. А.С. Кожаринов**

**Оценка: \_5\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Дата защиты: \_06.04.21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва 2021**

Оглавление

Список используемых основных сокращений [3](#_Toc58608823)

[Введение 4](#_Toc58608823)

[1 Постановка задачи 5](#_Toc58608824)

[2 Использованные средства разработки и системные требования 7](#_Toc58608825)

[2.1 Выбранное архитектурное решение 7](#_Toc58608826)

[2.2 Описание инструментов разработки 8](#_Toc58608827)

[2.3 Системные требования 9](#_Toc58608827)

[3 Модель данных проекта 11](#_Toc58608828)

[3.1 Реляционная модель 11](#_Toc58608829)

[3.2 Документо-ориентированная модель 15](#_Toc58608830)

[3.3 Организация ролевого доступа к данным 17](#_Toc58608831)

[4 Разработанное программное обеспечение 18](#_Toc58608833)

[4.1 Сервлеты 18](#_Toc58608832)

[4.2 Сервисные классы 20](#_Toc58608832)

[4.3 Веб-страницы 22](#_Toc58608832)

5. [Выводы 25](#_Toc58608834)

[Приложение А 26](#_Toc58608835)

# **Список используемых основных сокращений**

NoSQL – not only SQL (не только SQL)

HTTP – HyperText Transfer Protocol (протокол передачи гипертекста)

HTML – HyperText Markup Language (язык гипертекстовой разметки)

MVC – Model-View-Controller (Модель-Представление-Контроллер)

JSP – JavaServer Pages

JSON – JavaScript Object Notation

BSON – Binary JavaScript Object Notation

СУБД – Система управления базами данных

ОМДП – Общая модель данных проекта

# **Введение**

Тема: “Разработка прототипа корпоративной социальной сети”.

В настоящее время всё больше разработчиков программных продуктов начинают отходить от старых стратегий мономодельного хранения данных, отдавая предпочтение многовариантным решениям. Базы данных, построенные в парадигме polyglot persistence позволяют решить сложную задачу, разбив ее на сегменты и применив различные модели баз данных для решения различных задач. При этом такие многовариантные системы открывают возможности использовать различные технологии для обработки различных потребностей в хранении данных.

В рамках данной работы был разработан демонстрационный прототип закрытой социальной сети, решающей задачу коммуникации сотрудников в границах единой корпоративной среды. Разработанный программный продукт можно разбить на:

* Общую модель данных проекта, состоящую из реляционной модели, реализованной посредством MySQL и документо-ориентированной модели на базе MongoDB;
* Веб-приложение, осуществляющее функции веб-сервера и реализующее взаимодействие с разработанной общей моделью данных проекта;
* Веб-сайт, являющийся интерфейсом между пользователем и веб-приложением.

В реляционную модель были внесены данные, требующие жесткой структуры и схемы, имеющие высокие требования к целостности и безопасности.

Документо-ориентированная модель создавалась для данных с нежесткой схемой, которые эффективнее хранить в качестве агрегатов, формализованных как json-подобные записи. Связь между документами NoSQL модели и таблицами реляционной модели осуществляется посредством хранимых внутри них ключей.

Выполнение курсовой работы производилось индивидуально студентом группы БПМ-17-2, Скибином Кириллом.

# **1 Постановка задачи**

В рамках работы требуется разработать демонстрационный прототип закрытой социальной сети, решающей задачу коммуникации сотрудников в границах единой корпоративной среды. Социальная сеть должна обладать следующим функционалом:

* Регистрация и авторизация пользователя;
* Выход пользователя;
* Редактирование информации в профиле пользователя после регистрации;
* Поиск страниц других пользователей;
* Возможность отправки запроса на добавления в друзья и начала чата с любым пользователем;
* Возможность отклонять и принимать заявки в друзья, удалять пользователей из друзей;
* Возможность ведения личных переписок со множеством пользователей посредством множества разделенных чатов;
* Роли пользователя, администратора и руководителя отдела;
* Панель администратора, позволяющая назначать и разжаловать руководителей отделов;
* Автоматическое создание шаблона страницы отдела при назначении нового руководителя отдела;
* Возможность добавлять/удалять/просматривать список сотрудников отдела;
* Написание постов на веб-странице отдела.

Разработка демонстрационного прототипа закрытой социальной сети требует поэтапного выполнения следующего перечня задач:

1. Проектирование и разработка общей модели данных проекта, представляющие из себя логически объединенные документо-ориентированную и реляционную модели;
2. Разработка набора микросервисов в экосистеме веб-приложения для взаимодействия с базами данных;
3. Верстка веб-страниц;
4. Разработка сервлетов, обслуживающих HTTP запросы, поступающие на сервер;
5. Наполнение баз данных записями с последующим тестированием разработанных программных компонентов.

В таблицах реляционной базы данных, являющейся составной частью ОМДП, должны храниться данный с высокими требованиями к целостности и безопасности: логины, пароли, роли зарегистрированных пользователей, основная информация о сотрудниках и отделах, ассоциативные таблицы для определения пар друзей и запросов на добавления в друзья.

В документах документо-ориентированной базы данных, являющейся составной частью ОМДП, должны храниться информация о всех чатах каждого пользователя, а также информация о всех постах на веб-страницах всех зарегистрированных отделов.

# **2 Использованные средства разработки и системные требования**

## **2.1 Выбранное архитектурное решение**

Разработанный программный продукт представляет из себя веб-приложение реализующее функции веб-сервера, обрабатывающего поступающие HTTP запросы, и возвращающего клиенту HTML страницы, которые могут быть просмотрены в любом браузере на Desktop.

В качестве архитектуры веб-приложения используется Model-View-Controller («Модель-Представление-Контроллер») схема. MVC – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер, таким образом, что модификация каждого компонента может осуществляться независимо.



Рисунок 1 – схема MVC

* Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние;
* Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели;
* Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида). За счёт такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем), просто предоставляя доступ к данным и управлению ими. Модель строится таким образом, чтобы отвечать на запросы, изменяя своё состояние, при этом может быть встроено уведомление «[наблюдателей](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B1%D0%BB%D1%8E%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))». Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений для одной «модели».

Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя.

Контроллер обеспечивает «связь» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

## **2.2 Описание инструментов разработки**

Создание реляционной и документо-ориентированной моделей хранения данных осуществлялось посредством использования СУБД MySQL 8.0 и MongoDB 4.4.5.

Разработка веб-приложения велась на языке программирования Java 11. В качестве интегрированной среды разработки использовался Eclipse. Для построения и управления проектом использовался фреймворк Apache Maven. В качестве контейнера сервлетов был задействован Tomcat 9.0. При разработке программного продукта были также использованы следующие фреймворки и технологии:

* Spring Boot – упрощение управления зависимостями, автоматическая конфигурация проекта;
* JSP, JSTL – создание веб-страниц;
* Spring Security – назначение ролей, управление доступом к страницам, управление сессиями и авторизацией;
* Spring MVC – веб-фреймворк, реализующий архитектуру Model-View-Controller;
* Hibernate – реализация ORM в соответствии со спецификацией JPA.
* Spring Data – подключение к СУБД MySQL и MongoDB.

## **2.3 Системные требования**

Для работы веб-приложения требуется одновременная работы двух СУБД: MySQL и MongoDB, работы сервера Tomcat, обрабатывающего запросы, а также наличие широкополосного доступа в интернет с постоянным IP адресом. К ЭВМ предъявляются следующие требования:

|  |  |
| --- | --- |
| Операционная система | Windows 7 x32/x64, MacOS 10.9 |
| Процессор | Intel Core I5 7200U 2.5 ГГц |
| GPU | Nvidia GeForce GTX 960 4 Гб |
| Свободное дисковое пространство | 1 Гб |
| ОЗУ | 8 Гб |

Таблица 1 – Минимальные системные требования

Разработанные веб-страницы не были оптимизированы для мобильных устройств, поэтому рекомендуемыми пользовательскими требованиями являются:

1. Настольный компьютер, ноутбук с операционной системой Windows 7/8/8.1/10, Linux, MacOS;
2. Поддержка веб-браузера.

# **3 Модель данных проекта**

Общая модель данных проекта представляет из себя две логически связанные базы данных: реляционную (MySQL) и документо-ориентированную (MongoDB). Связь между документами NoSQL модели и таблицами реляционной модели является односторонней и реализуется путём добавления в таблиц колонок, имеющих формат binary(12) и хранящих идентификатор документа.

## **3.1 Реляционная модель**

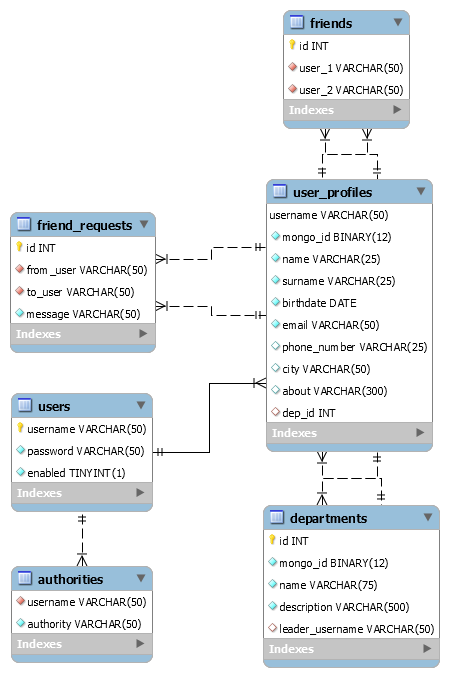


Рисунок 2 – Диаграмма реляционной базы данных

Реляционная модель состоит из 6 таблиц, содержащих информацию для авторизации, распределение ролей, профили сотрудников и отделов, а также две ассоциативные таблицы для запросов на добавления в друзья и для пар друзей.

|  |  |  |
| --- | --- | --- |
| № | Название таблицы | Назначение |
| 1 | users | Хранение пар логин-пароль |
| 2 | authorities | Хранение ролей каждого пользователя |
| 3 | user\_profiles | Хранение основной информации о сотруднике, а также ключа MongoDB документа с чатами |
| 4 | departments | Хранение основной информации об отделе, а также ключа MongoDB документа с постами |
| 5 | friends | Ассоциативная таблица, хранящая пары пользователей, являющихся друзьями |
| 6 | friend\_requests | Таблица, хранящая запросы на дружбу |

Таблица 2 – Общий список таблиц реляционной базы данных

Ниже приведены описания полей всех таблиц:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица “users” | | | | |
| № | Название поля | Тип данных | Ключ (PK/FK) | Назначение |
| 1 | username | varchar(50) | PK | Логин пользователя |
| 2 | password | varchar(50) | - | Пароль для авторизации |
| 3 | enabled | tinyint(1) | - | Флаг: доступна авторизация |

Таблица 3 – Поля таблицы “users”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица “authorities” | | | | |
| № | Название поля | Тип данных | Ключ (PK/FK) | Назначение |
| 1 | username | varchar(50) | PK, FK (users.username) | Логин пользователя |
| 2 | authority | varchar(50) | PK | Роль пользователя. (ROLE\_EMPLOYEE, ROLE\_MANAGER, ROLE\_ADMIN) |

Таблица 4 – Поля таблицы “authorities”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица “user\_profiles” | | | | |
| № | Название поля | Тип данных | Ключ (PK/FK) | Назначение |
| 1 | username | varchar(50) | PK | Логин пользователя |
| 2 | mongo\_id | binary(12) | FK\* | Идентификатор MongoDB документа, содержащего чаты данного пользователя |
| 3 | name | varchar(25) | - | Имя |
| 4 | surname | varchar(25) | - | Фамилия |
| 5 | birthdate | date | - | Дата рождения |
| 6 | email | varchar(50) | - | Адрес электронной почты |
| 7 | phone\_number | varchar(25) | - | Телефонный номер |
| 8 | city | varchar(50) | - | Город проживания |
| 9 | about | varchar(300) | - | Общая информация о сотруднике |
| 10 | dep\_id | int | FK | Идентификатор департамента, к которому принадлежит сотрудник |

Таблица 5 – Поля таблицы “user\_profiles”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица “departments” | | | | |
| № | Название поля | Тип данных | Ключ (PK/FK) | Назначение |
| 1 | id | int | PK | Идентификатор отдела |
| 2 | mongo\_id | binary(12) | FK\* | Идентификатор MongoDB документа, содержащего посты, сделанные руководителем отдела |
| 3 | name | varchar(75) | - | Название отдела |
| 4 | description | varchar(500) | - | Описание отдела |
| 5 | leader\_username | varchar(50) | FK | Логин руководителя отдела |

Таблица 6 – Поля таблицы “departments”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица “friends” | | | | |
| № | Название поля | Тип данных | Ключ (PK/FK) | Назначение |
| 1 | id | int | PK | Идентификатор |
| 2 | user\_1 | varchar(50) | FK | Первый пользователь из пары друзей |
| 3 | user\_2 | varchar(50) | FK | Второй пользователь из пары друзей |

Таблица 7 – Поля таблицы “friends”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица “friend\_requests” | | | | |
| № | Название поля | Тип данных | Ключ (PK/FK) | Назначение |
| 1 | id | int | PK | Идентификатор |
| 2 | from\_user | varchar(50) | FK | Логин пользователя, отправившего запрос дружбы |
| 3 | to\_user | varchar(50) | FK | Логин пользователя, кому был отправлен запрос дружбы |
| 4 | message | varchar(50) | - | Сообщение в запросе |

Таблица 8 – Поля таблицы “friend\_requests”

## **3.2 Документо-ориентированная модель**

Документы MongoDB, являются аналогами записей в реляционных таблицах. Документы хранятся в формате BSON, являющимся надмножеством JSON, включающим дополнительно регулярные выражения, двоичные данные и даты.

Всего в рамках документо-ориентированной модели были созданы 2 коллекции:

1. Пользовательские чаты “users\_chats”. Отдельный документ хранит все чаты данного пользователя в массиве объектов “chats”. Чат представляет из себя объект, где поле “сorrespUsername” – логин собеседника по чату, “pms” – массив объектов личных сообщений. Личное сообщение – это объект, где поле “receipt” обозначает сторону, отославшую сообщение; поле “text” – текст сообщения, после “timestamp” – дату и время отправления сообщения. Документы из данной коллекции логически связаны с реляционной таблицей user\_profiles посредством колонки mongo\_id, которая содержит “\_id” документа;
2. Посты отдела “departments\_blogs”. Отдельный документ хранит все посты, сделанные руководителем отдела в массиве объектов “blogs”. Пост представляет из себя объект, где поле “text” обозначает текст поста, а поле “timestamp” – дату и время публикации поста. Документы из данной коллекции логически связаны с реляционной таблицей departments посредством колонки mongo\_id, которая содержит “\_id” документа.



Рисунок 3 – Пример документа из коллекции “users\_chats” для пользователя admin

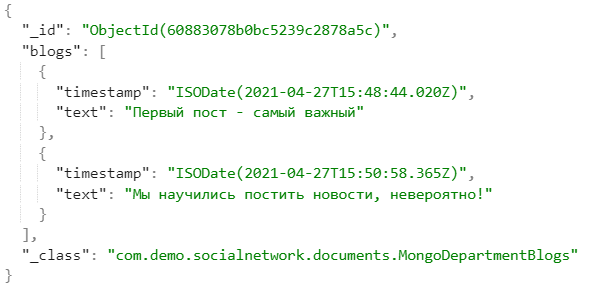


Рисунок 4 – Пример документа из коллекции “ departments\_blogs ” для IT департамента

## **3.3 Организация ролевого доступа к данным**

При проектировании социальной сети было предусмотрено создание 3 ролей, обладающих различными привилегиями:

1. Рядовой сотрудник – может редактировать собственный профиль, добавлять других пользователей в друзья, переписываться с пользователями, просматривать информацию и посты на веб-странице своего отдела. Также может осуществлять поиск других сотрудников по имени и фамилии;
2. Руководитель отдела – может редактировать страницу подведомственного отдела, публиковать посты на веб-странице отдела, добавлять и удалять сотрудников из отдела;
3. Администратор – может назначать и разжаловать руководителей. Имеет доступ к панели администрирования.

Все пользователи без исключения имеют привилегии рядового сотрудника. Пользователь не может быть одновременно администратором и руководителем отдела. Администратор имеет право присваивать или наоборот убирать привилегии руководителя отдела через панель администрирования.

Для реализации системы ролей и авторизации при разработке веб-приложения использовался фреймворк Spring Security.

# **4 Разработанное программное обеспечение**

## **4.1 Сервлеты**

Для обработки поступающих на сервер HTTP запросов в рамках Spring MVC контроллера были разработаны сервлеты, обслуживающие следующие URL пути:

1. @RequestMapping("/") – перенаправляет пользователя на страницу собственного профиля;
2. @RequestMapping("/users/{username}") – страница профиля пользователя с логином username;
3. @RequestMapping("/users/{username}/friends") – страница друзей пользователя с логином username;
4. @RequestMapping("/send\_friend\_request/{username}") – запрос на дружбу от авторизированного пользователя в сторону пользователя с логином username;
5. @RequestMapping("/confirm\_friend\_request/{username}") – добавление пользователя с логином username через одобрение его запроса;
6. @RequestMapping("/decline\_friend\_request/{username}") – отклонение запроса на добавления в друзья, поступившего от пользователя с логином username;
7. @RequestMapping("/remove\_friend/{username}") – удаления пользователя с логином username из друзей;
8. @RequestMapping("/chats") – страница со всеми чатами пользователя;
9. @RequestMapping("/chats/{username}") – страница чата с пользователем с логином username;
10. @RequestMapping("/create\_chat/{username}") – запрос на создание чата с пользователем с логином username;
11. @PostMapping("/chats/{username}/send\_message") – отправка сообщения в чат с пользователем с логином username;
12. @RequestMapping("/search") – страница поиска сотрудников по имени и фамилии;
13. @RequestMapping("/deps/{depId}") – страница отдела с идентификатором depId;
14. @PostMapping("/deps/{depId}/post\_submit") – публикация поста в новостной ленте на странице отдела с идентификатором depId;
15. @RequestMapping("/deps/{depId}/members") – страница со списком участников отдела с идентификатором depId;
16. @RequestMapping("/deps/add\_members") – страница добавления участников в отдел;
17. @RequestMapping("/deps/{depId}/add\_member/{username}") – запрос на добавление пользователя с логином username в отдел с идентификатором depId;
18. @RequestMapping("/deps/{depId}/remove\_member/{username}") – запрос на удаление пользователя с логином username из отдела с идентификатором depId;
19. @GetMapping("/dep\_edit") – страница редактирования информации об отделе;
20. @PostMapping("/dep\_edit\_submit") – запрос на обновление информации об отделе;
21. @RequestMapping("/user\_promote/{username}") – запрос на назначение пользователя с логином username руководителем отдела;
22. @RequestMapping("/user\_demote/{username}") – запрос на снятие пользователя с логином username с поста руководителя отдела;
23. @GetMapping("/user\_edit") – страница редактирования профиля сотрудника;
24. @PostMapping("/user\_edit\_submit") – запрос на обновление профиля сотрудника;
25. @RequestMapping("/admin\_panel") – страница с панелью администрирования;
26. @RequestMapping("/access\_denied") – страница, сообщающая, что у пользователя не хватает прав доступа;
27. @GetMapping("/login") – страница авторизации пользователя;
28. @GetMapping("/register") – страница регистрации пользователя;
29. @PostMapping("/register\_submit") – запрос на регистрацию нового пользователя.

## **4.2 Сервисные классы**

Для взаимодействия с базами данных были разработаны 4 сервиса: AdminService, DepartmentService, RegisterService, UserService.

AdminService содержит в себе основные операции, которые выполняет администратор при нахождении в панели администратора, а именно:

1. promoteUser(String username) – назначение пользователя с логином username руководителем;
2. demoteUser(String username) – снятие пользователя с логином username с поста руководителя.

DepartmentService содержит в себе функционал веб-странц связанных с отделами:

1. getDepartmentInfo(int id) – возвращают основную информацию об отделе с идентификатором id;
2. getDepartmentPosts(String mongoId) – возвращает публикации из документа с идентификатором mongoId;
3. addBlog(int departmentId, MongoBlog blog) – публикует пост blog на странице отдела с идентификатором departmentId;
4. updateDepartment(Department department) – обновляет информацию об отделе department;
5. addMember(int departmentId, String username) – добавляет пользователя с логином username в отдел с идентификатором departmentId;
6. removeMember(int departmentId, String username) – удаляет пользователя с логином username из отдела с идентификатором departmentId.

RegisterService содержим одну единственную функцию регистрации пользователя registerUser(RegisterModel newUser).

UserService содержит в себе функционал веб-странц связанных с пользователями:

1. getAllUserProfiles() – возвращает всех зарегистрированных пользователей;
2. getUserProfile (String username) – возвращает пользователя с логином username;
3. updateUserProfile(UserProfile updatedUserProfile) – обновляет профиль пользователя updatedUserProfile;
4. sendMessage(String usernameFrom, String usernameTo, String text) – отправляет сообщение text от пользователя usernameFrom пользователю usernameTo;
5. createChat(String username1, String username2) – создаёт чат между пользователями username1 и username2;
6. chatExists(String username1, String username2) – возвращает флаг существования чата между пользователями username1 и username2;
7. sendFriendRequest(String usernameFrom, String usernameTo) – отправляет запрос на добавления в друзья от пользователя usernameFrom к пользователю usernameTo;
8. declineFriendRequest(String username, String usernameRequest) – отклоняет запрос на добавления в друзья от пользователя usernameRequest к пользователю username;
9. confirmFriendRequest(String username, String usernameRequest) – подтверждает запрос на добавления в друзья от пользователя usernameRequest к пользователю username;
10. deleteFriendPair(String username1, String username2) – удаляет пару друзей username1 и username2;
11. getAllChatsWithProfilesSortedByLastMessageDate(String mongoId) – возвращает все чаты из документа mongoId, отсортированные по дате и времени последнего сообщения в чате;
12. getChatWithProfileByCorrespUsername(String mongoId, String username) – возвращает чат с пользователем username из документа mongoId;
13. Set<UserProfile> searchUsers(String searchQuery) – возвращает множество пользователей, найденных по запросу searchQuery.

## **4.3 Веб-страницы**

Взаимодействие пользователя с веб-приложением осуществляется посредством веб-бразуера. Всего в рамках проделанной работы было сверстано 14 веб-страниц с использованием HTML, JSP, JSTL. Ниже представлены экранные снимки веб-страниц, загруженных в браузере Google Chrome:

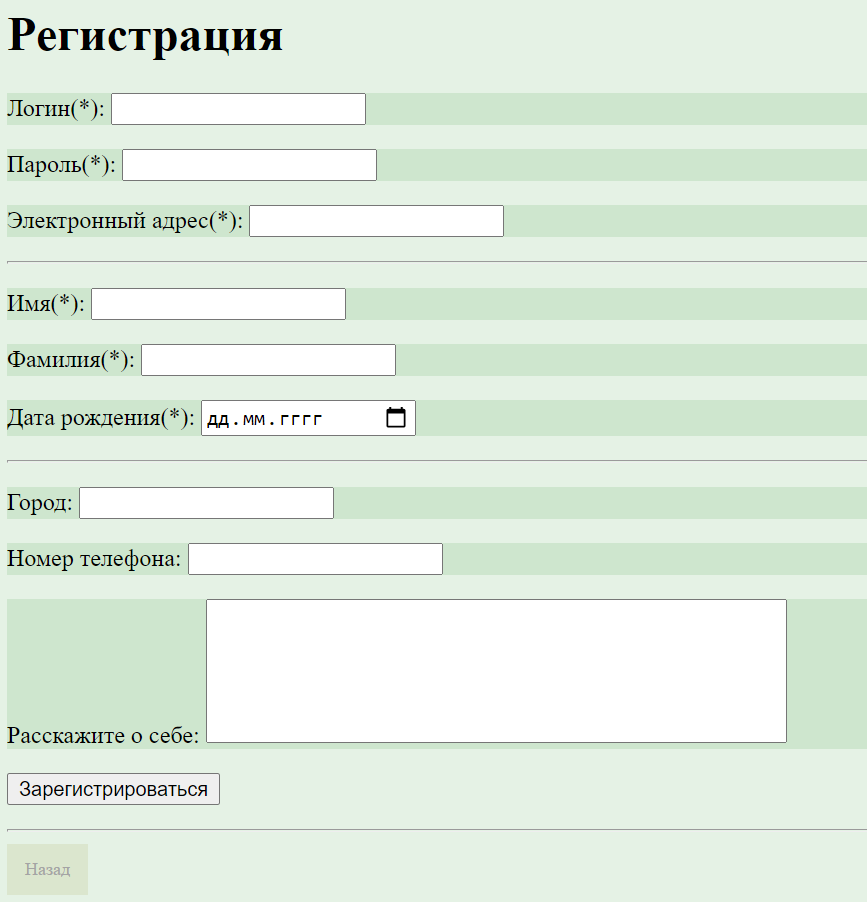


Рисунок 5 – Страница регистрации, URL: /register

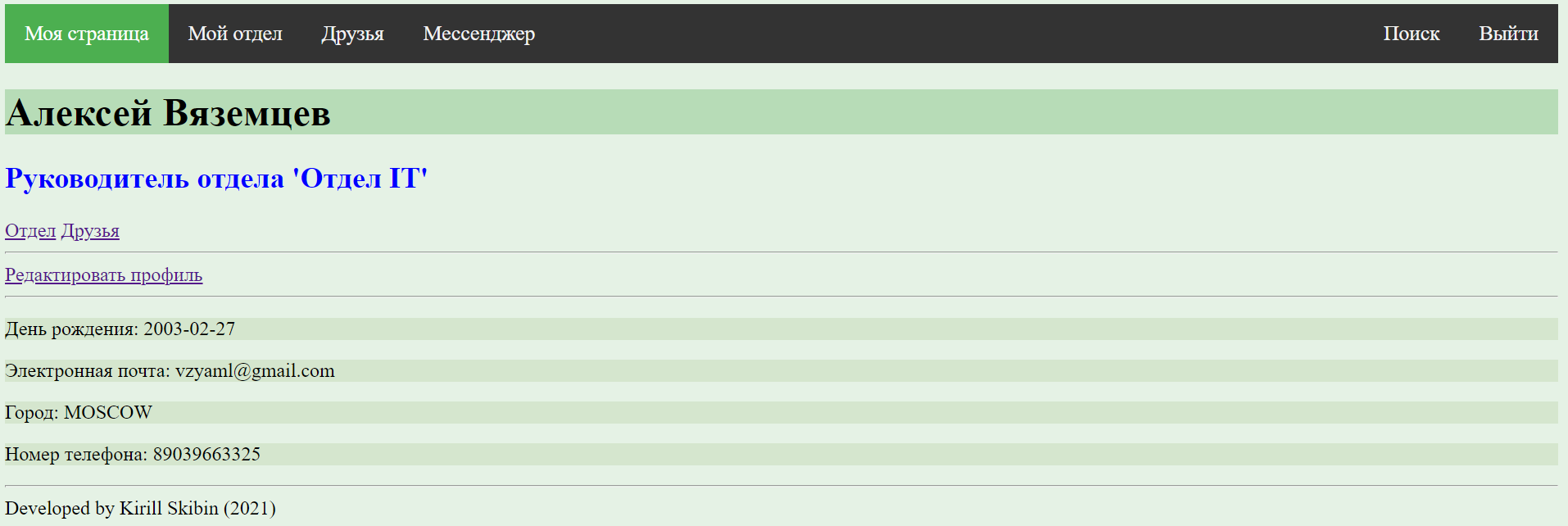


Рисунок 6 – Страница профиля пользователя с логином test, URL: /users/test

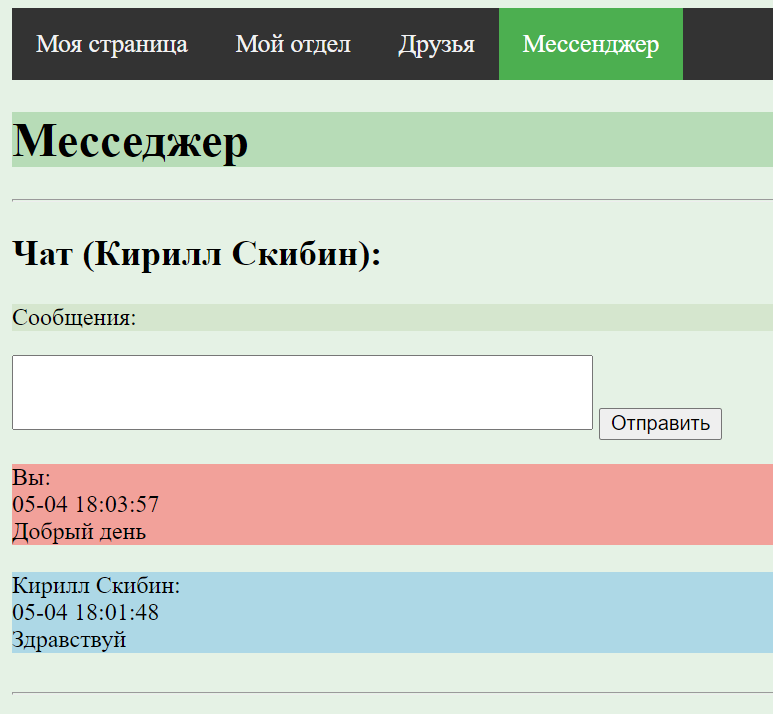


Рисунок 7 – Чат с пользователем admin, URL: /chats/admin

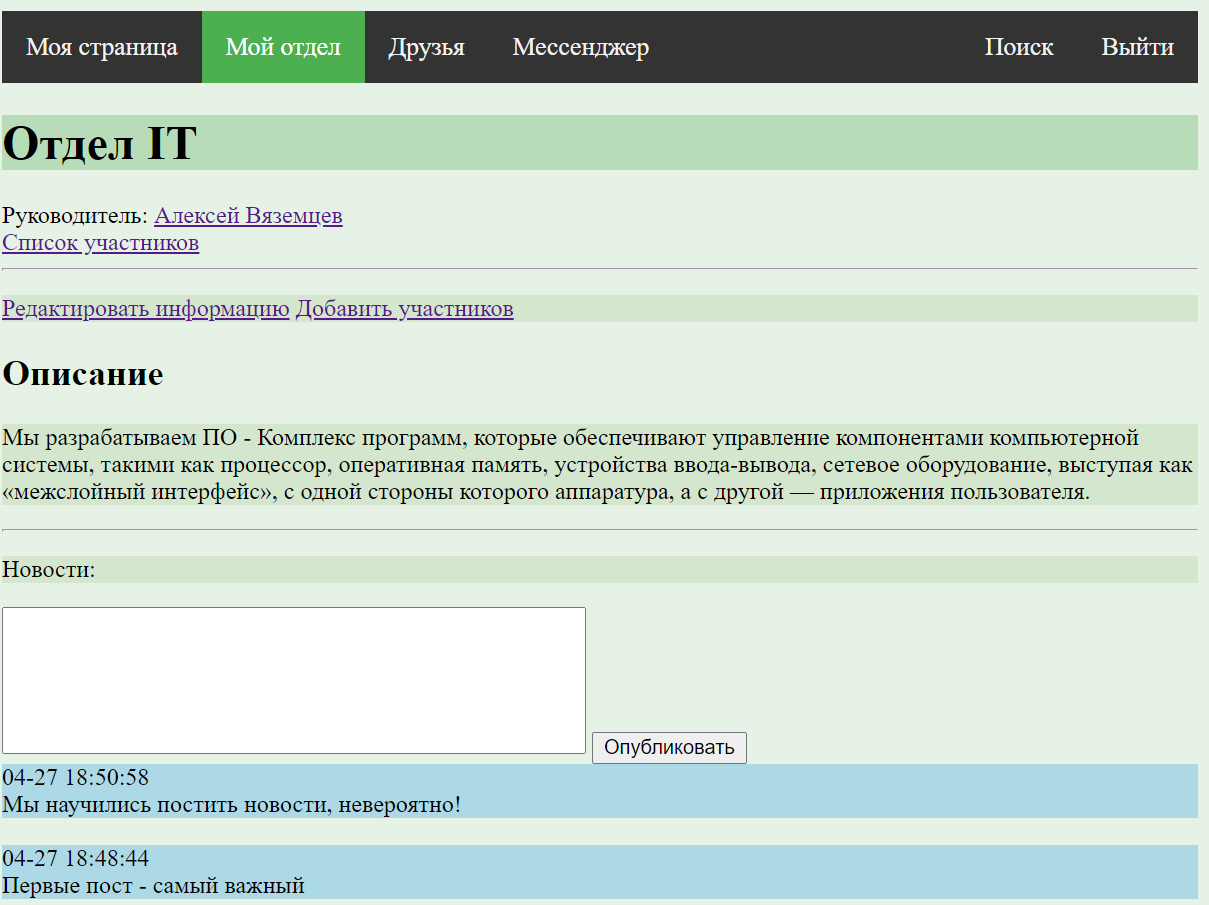


Рисунок 8 – Страница отдела, URL: /deps/25

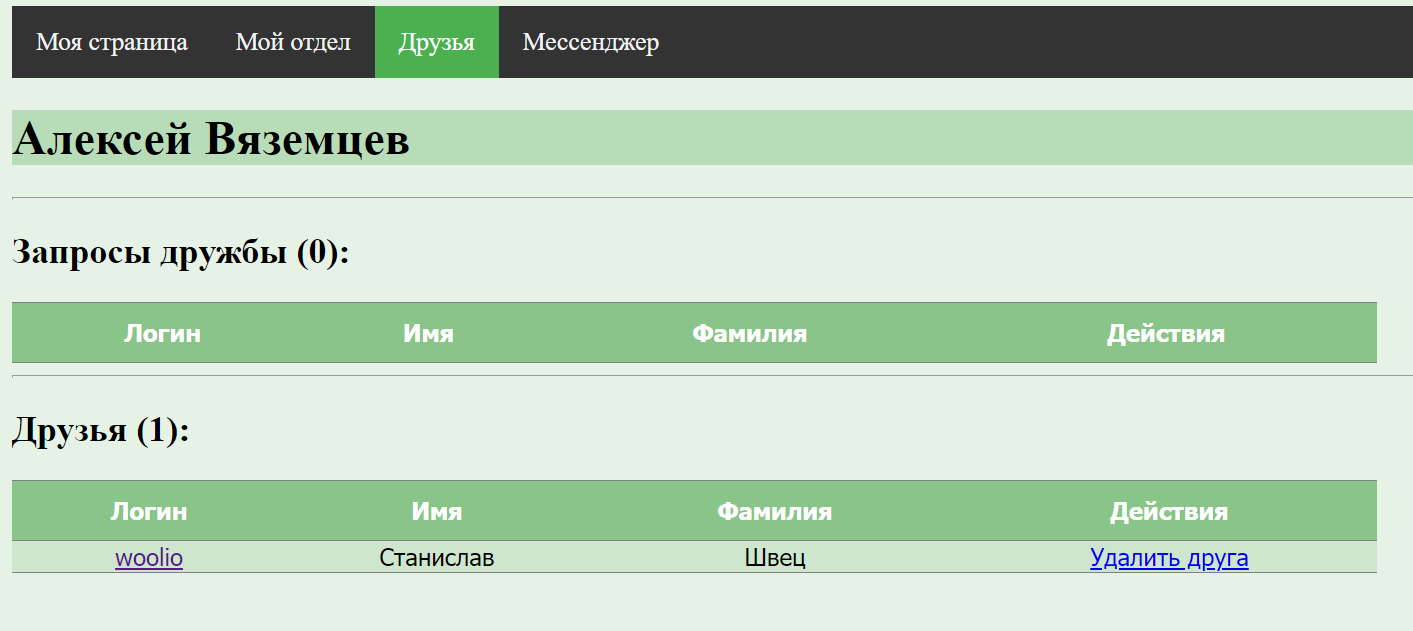


Рисунок 9 – Друзья пользователя с логином test, URL: /users/test/friends

# **5 Выводы**

Таким образом, в данной работе работы был разработан демонстрационный прототип закрытой социальной сети, решающей задачу коммуникации сотрудников в границах единой корпоративной среды.

Для хранения данных была использована парадигма polyglot persistence и создана многовариантная модель, включающая реляционную базу данных на основе MySQL и документо-ориентированную базу данных с использованием СУБД MongoDB. В реляционную модель были внесены данные, требующие жесткой структуры и схемы, имеющие высокие требования к целостности и безопасности. Документо-ориентированная модель создавалась для данных с нежесткой схемой, которые эффективнее хранить в качестве агрегатов.

Для полноценной работы социальной сети были созданы три роли, обладающие различными привилегиями: роли рядового сотрудника, руководителя отдела и администратора.

В ходе разработки ОМДП было создано 6 реляционных таблиц и 2 коллекции документов.

В результате проделанной работы по разработке веб-приложения было создано:

* 29 сервлетов, обслуживающих HTTP запросы;
* 14 веб-страниц для взаимодействия с пользователями;
* 4 микросервиса для взаимодействия с базой данных на уровне сервера.

Разработанная закрытая социальная сеть находится в стадии демонстрационного прототипа и требует дальнейших усовершенствований как на уровне фронтенда, так и бэкенда. Тем не менее, благодаря использованию схемы MVC, работа по модифицированию программного продукта может быть делегирована нескольким специалистам разных профилей, которые смогут работать независимо друг от друга.

# 

# **6 Приложение А**

**SQL скрипт создания реляционной базы данных**

DROP DATABASE IF EXISTS `social\_network`;

CREATE DATABASE IF NOT EXISTS `social\_network`;

USE `social\_network`;

--

-- Table structure for table `users`

--

DROP TABLE IF EXISTS `users`;

CREATE TABLE `users` (

`username` varchar(50) NOT NULL,

`password` varchar(50) NOT NULL,

`enabled` tinyint(1) NOT NULL,

PRIMARY KEY (`username`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--

-- Inserting data for table `users`

--

INSERT INTO `users`

VALUES

('john','{noop}test123',1),

('mary','{noop}test123',1),

('susan','{noop}test123',1);

--

-- Table structure for table `authorities`

--

DROP TABLE IF EXISTS `authorities`;

CREATE TABLE `authorities` (

`username` varchar(50) NOT NULL,

`authority` varchar(50) NOT NULL,

UNIQUE KEY `authorities\_idx\_1` (`username`,`authority`),

CONSTRAINT `authorities\_ibfk\_1` FOREIGN KEY (`username`) REFERENCES `users` (`username`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--

-- Inserting data for table `authorities`

--

INSERT INTO `authorities`

VALUES

('john','ROLE\_EMPLOYEE'),

('mary','ROLE\_EMPLOYEE'),

('mary','ROLE\_MANAGER'),

('susan','ROLE\_EMPLOYEE'),

('susan','ROLE\_ADMIN');

DROP TABLE IF EXISTS `user\_profiles`;

CREATE TABLE `user\_profiles` (

`username` varchar(50) NOT NULL,

`mongo\_id` varchar(128) NOT NULL,

`name` varchar(25) NOT NULL,

`surname` varchar(25) NOT NULL,

`birthdate` date NOT NULL,

`email` varchar(50) NOT NULL,

`phone\_number` varchar(25) DEFAULT NULL,

`city` varchar(50) DEFAULT NULL,

`about` varchar(300) DEFAULT NULL,

`dep\_id` int(11) DEFAULT NULL,

PRIMARY KEY (`username`),

CONSTRAINT `SECURE\_USER`

FOREIGN KEY `SECURE\_USER\_KEY` (`username`) REFERENCES `users` (`username`)

ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `departments`;

CREATE TABLE `departments` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`mongo\_id` varchar(128) NOT NULL,

`name` varchar(75) NOT NULL,

`description` varchar(500) NOT NULL,

`leader\_username` varchar(50) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `LEADER\_KEY` (`leader\_username`),

CONSTRAINT `LEADER`

FOREIGN KEY (`leader\_username`)

REFERENCES `user\_profiles` (`username`)

ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

ALTER TABLE `user\_profiles`

ADD CONSTRAINT `DEP\_CONST` FOREIGN KEY (`dep\_id`)

REFERENCES`departments` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION;

DROP TABLE IF EXISTS `friend\_requests`;

CREATE TABLE `friend\_requests` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`from\_user` varchar(50) NOT NULL,

`to\_user` varchar(50) NOT NULL,

`message` varchar(50) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `FROM\_TO\_USER\_KEY` (`from\_user`,`to\_user`),

CONSTRAINT `FROM`

FOREIGN KEY `FROM\_KEY` (`from\_user`) REFERENCES `user\_profiles` (`username`)

ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `TO`

FOREIGN KEY `TO\_KEY` (`to\_user`) REFERENCES `user\_profiles` (`username`)

ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `friends`;

CREATE TABLE `friends` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`user\_1` varchar(50) NOT NULL,

`user\_2` varchar(50) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `USER\_1\_2\_KEY` (`user\_1`,`user\_2`),

CONSTRAINT `USER1`

FOREIGN KEY `USER1\_KEY` (`user\_1`) REFERENCES `user\_profiles` (`username`)

ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `USER2`

FOREIGN KEY `USER2\_KEY` (`user\_2`) REFERENCES `user\_profiles` (`username`)

ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

**Пример программного кода веб-приложения**

@Override

@Transactional(transactionManager = "transactionManager")

public List<ChatWithUserProfileModel> getAllChatsWithProfilesSortedByLastMessageDate(String mongoId) {

Optional<MongoUserChats> userChatsOpt = mongoUserChatsRepo.findById(mongoId);

if(userChatsOpt.isPresent()) {

MongoUserChats userChats = userChatsOpt.get();

List<MongoChat> chats = userChats.getChats();

chats.sort((MongoChat c1, MongoChat c2) -> c2.getLastMessageTimestamp().compareTo(c1.getLastMessageTimestamp()));

List<ChatWithUserProfileModel> chatsWithProfiles = chats.stream().collect(

Collectors.mapping(

p -> new ChatWithUserProfileModel(p,userProfileRepo.findById(p.getCorrespUsername()).get()),

Collectors.toList()));

return chatsWithProfiles;

}

return null;

}

**Дополнительные экранные снимки веб-страниц**

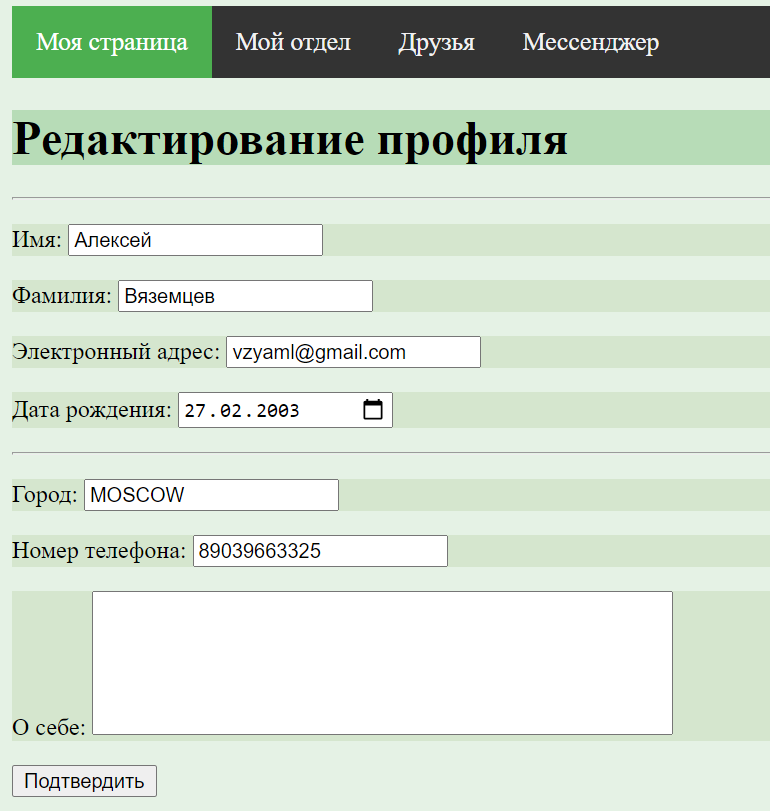


Рисунок 10 – Редактирование профиля пользователя, URL: /user\_edit