Министерство науки и высшего образования РФ

Федеральное государственное автономное образовательное учреждение

высшего образования

«Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

Институт Радиоэлектроники и информационных технологий – РТФ

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**

по дисциплине «Виртуализация и облачные технологии»

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ Немтинов А.В.

Студент гр. РИМ-220960 Коваль Д.В.

**СОДЕРЖАНИЕ**

[**ПОСТАНОВКА ЗАДАЧА** 3](#_Toc154449611)

[**РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЕ** 4](#_Toc154449612)

[**Стек технологий** 4](#_Toc154449613)

[**Архитектура веб-сервисов** 4](#_Toc154449614)

[**Средство управления доступом Keycloak** 6](#_Toc154449615)

[**Разработка серверной части** 7](#_Toc154449616)

[**Разработка клиентской части** 12](#_Toc154449617)

[**Разработка базы данных** 17](#_Toc154449618)

[**ИНСТРУКЦИЯ РАЗВЕРТЫВАНИЯ** 22](#_Toc154449619)

[**Docker** 22](#_Toc154449620)

[**Развертывание системы в Docker** 23](#_Toc154449621)

[**Настройка Keycloak** 26](#_Toc154449622)

[**ЗАКЛЮЧЕНИЕ** 29](#_Toc154449623)

# **ПОСТАНОВКА ЗАДАЧА**

Целью практической работы была разработка "облачной" системы информирования пользователей о днях рождениях. При этом была предусмотрена возможность размещения фотографий и сведений о пользователях, а при приближении дня рождения необходимо выводить соответствующие сведения с фотографией на панель информирования. Также предусмотрена возможность отправки сообщения пользователю непосредственно в системе.

# **РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЕ**

Веб-приложения состоит из 4 частей: клиентская часть, серверная часть, PostgreSQL, Keycloak.

**Стек технологий**

Backend-часть:

* Фреймворк – Spring Boot 2.6.1, язык программирования - Java 17;
* Flyway 8.0.4 – инсрумент для управления миграциями БД.

Fronend-часть:

* Фреймворк – Spring Boot 2.6.1, язык программирования - Java 17;
* Thymeleaf 3.0.12-RELEASE – механизм Java-шаблонов для обработки HTML, JavaScript, CSS и простого текста.
* OpenFeign 3.1.5 - декларативный HTTP клиент, для работы с API.

Система контроля версий:

* Git и удалённые репозитории GitHub.

СУБД:

* PostgreSQL 16.0.

Система управления идентификацией и авторизацией:

* Keycloak 22.0.5.

**Архитектура веб-сервисов**

Архитектура веб-сервиса представляет собой взаимодействие между такими компонентами, как пользовательские интерфейсы, мониторы обработки транзакций, базы данных. Все это необходимо для того, чтобы все элементы правильно функционировали.

За основу клиентской и серверной части был взят архитектурный паттерн Model-View-Controller (MVC, «Модель-Представление-Контроллер») для удобного разделения логических компонентов кода и создавать их отдельно друг от друга.

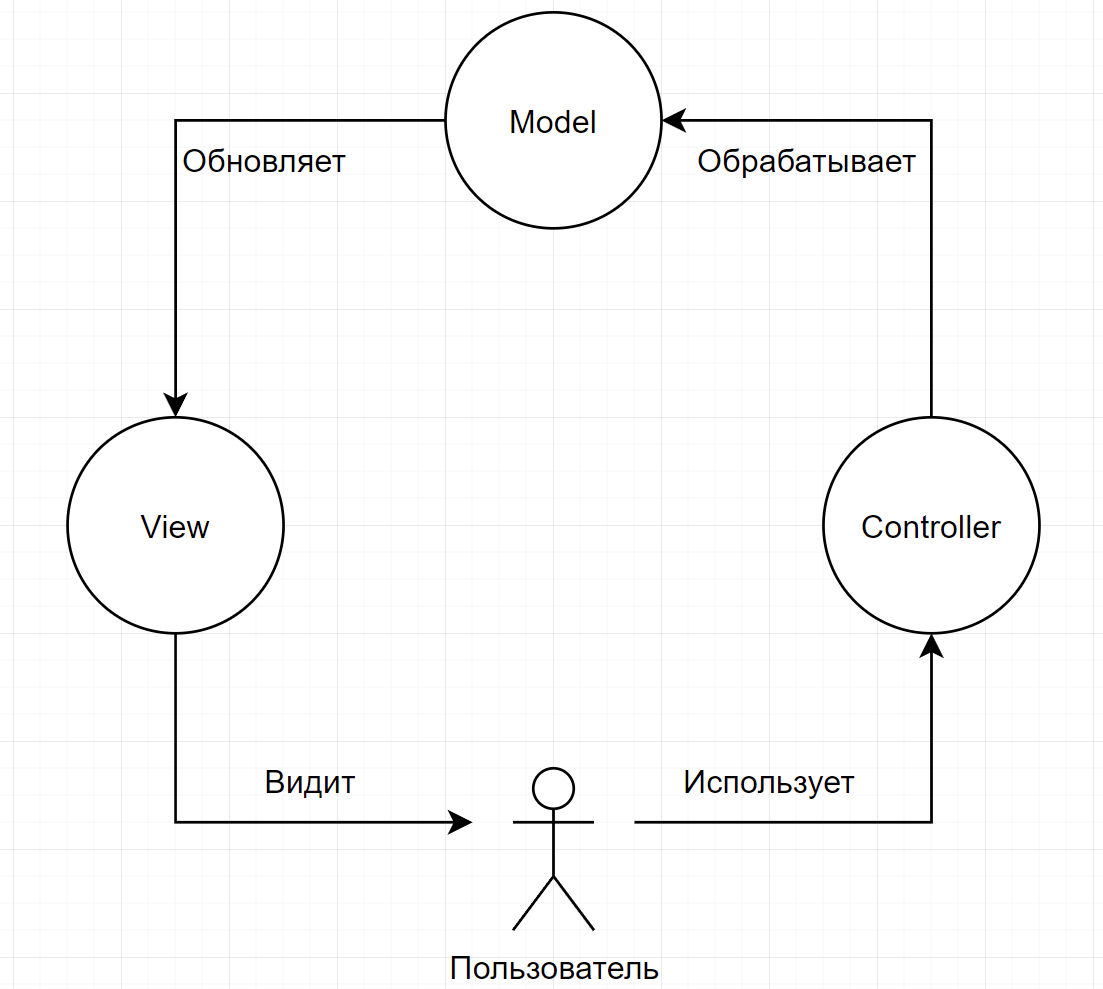


Рисунок 1 – Архитектура MVC

Суть архитектуры заключается в том, что данные сервиса и управления логики разделены на три отдельных компонента: модель, представление и контроллер — таким образом, чтобы модифицирование каждого компонента было независимо от других (рисунок 2).

***Модель –*** предоставляет данные, реагирует на команды контроллера, изменяя свое состояние.

***Представление*** – отвечает за отображение данных модели пользователю.

***Контроллер*** – реализует действия пользователя, уведомляет модель об изменениях.

Для технической реализации серверной части используется язык программирования Java с использованием платформы Spring Boot.

Клиентская часть веб-сайта – это графический интерфейс, который отображается в браузере. Другими словами, это та информация, которую видит пользователь на странице. Именно через интерфейс пользователь взаимодействует с приложением, когда вводит и просматривает информацию, переходит по ссылкам.

Таким образом для корректной работы сервиса невозможно обойтись без базы данных. Структуру базы данных, а также перечень таблиц с типами полей и их описанием рассмотрим в следующем пункте.

**Средство управления доступом Keycloak**

Keycloak – это средство управления доступом и идентификации (IAM), которое предоставляет функции аутентификации, авторизации и управления идентификацией для веб-приложений и служб. Он является открытым исходным кодом и разрабатывается Red Hat.

Keycloak обеспечивает следующие основные функциональности:

* Аутентификация: Пользователи могут войти в систему с использованием различных методов аутентификации, таких как имя пользователя/пароль, OpenID Connect, OAuth и многие другие.
* Авторизация: Keycloak управляет разрешениями и правами доступа, определяя, какие пользователи или группы пользователей имеют доступ к конкретным ресурсам.
* Управление сеансами: Keycloak поддерживает управление сеансами и включает в себя функции однократного входа (Single Sign-On, SSO), что означает, что после успешной аутентификации пользователь может получить доступ ко всем подключенным приложениям без повторной аутентификации.
* Управление пользователями и группами: Keycloak предоставляет административный интерфейс для управления пользователями и группами, что облегчает администрирование системы управления доступом.

Серверная и клиентская часть связаны с Keycloak через протокол авторизации и аутентификации OAuth 2.0.

Процесс выглядит следующим образом:

* *Конфигурация Keycloak*: Настройка клиентов и realm’ов в Keycloak, чтобы определить приложения и ресурсы, которые будут защищены.
* *Аутентификация пользователя*: Клиентская часть отправляет запрос на аутентификацию к Keycloak, который затем взаимодействует с конечным пользователем для проверки его учетных данных.
* *Выдача токена*: После успешной аутентификации Keycloak выдает токен доступа, который клиент может использовать для обращения к защищенным ресурсам на сервере.
* *Проверка токена на серверной части*: Серверная часть может проверить токен, отправив запрос к Keycloak, чтобы удостовериться, что токен действителен и что пользователь имеет необходимые разрешения.

**Разработка серверной части**

Серверная часть (backend) – это часть программного обеспечения, которая обрабатывает бизнес-логику, взаимодействует с базой данных, обеспечивает безопасность и управление доступом, а также выполняет другие серверные задачи. Backend отвечает за обработку запросов от клиентской части и возврат необходимых данных

Серверная часть, как и клиентская были написана на языке Java, с использованием фреймворка – Spring Boot. Он предоставляет удобный способ создания самостоятельных, готовых к работе приложений, которые могут быть запущены "из коробки" без необходимости настройки множества компонентов.

Spring Boot поставляется с большим количеством встроенных функций и библиотек, которые упрощают разработку приложений, таких как автоматическая конфигурация, встроенный сервер приложений, управление зависимостями и многое другое.

Spring Boot также интегрируется с другими фреймворками Spring, такими как Spring MVC (для создания веб-приложения), Spring Data (для обращения к БД) и Spring Security (для обеспечения безопасности), что позволяет создавать мощные и гибкие приложения на основе Java.

Для миграции объектов базы данных используется инструмент Flyway. Он предназначен для автоматизации процесса обновления баз данных в соответствии с изменениями в коде приложения. Flyway позволяет легко управлять изменениями схемы базы данных, контролировать их версии и применять к базе данных в управляемом и контролируемом порядке.

Некоторые ключевые особенности Flyway:

Версионирование миграций: Flyway предлагает версионирование миграций, что означает, что каждая миграция имеет уникальный номер версии. Это обеспечивает последовательный и контролируемый порядок применения миграций к базе данных.

1. *Поддержка многих баз данных*: Flyway совместим с различными базами данных, включая PostgreSQL, MySQL, Oracle, Microsoft SQL Server и другие.
2. *Интеграция с сборкой и развертыванием*: Flyway может быть интегрирован с процессами сборки и развертывания приложения. Это означает, что миграции могут автоматически выполняться при каждом развертывании приложения.
3. *Простота использования*: Flyway работает с использованием SQL-скриптов для определения миграций. Разработчики могут создавать эти скрипты, описывая необходимые изменения в структуре базы данных.
4. *Контроль над миграциями*: Flyway позволяет применять миграции вручную, автоматически или в рамках процесса развертывания. Это дает контроль над тем, когда и как применять изменения к базе данных.

Полный перечень методов, которая реализует серверная часть, представлена на рисунке 2.

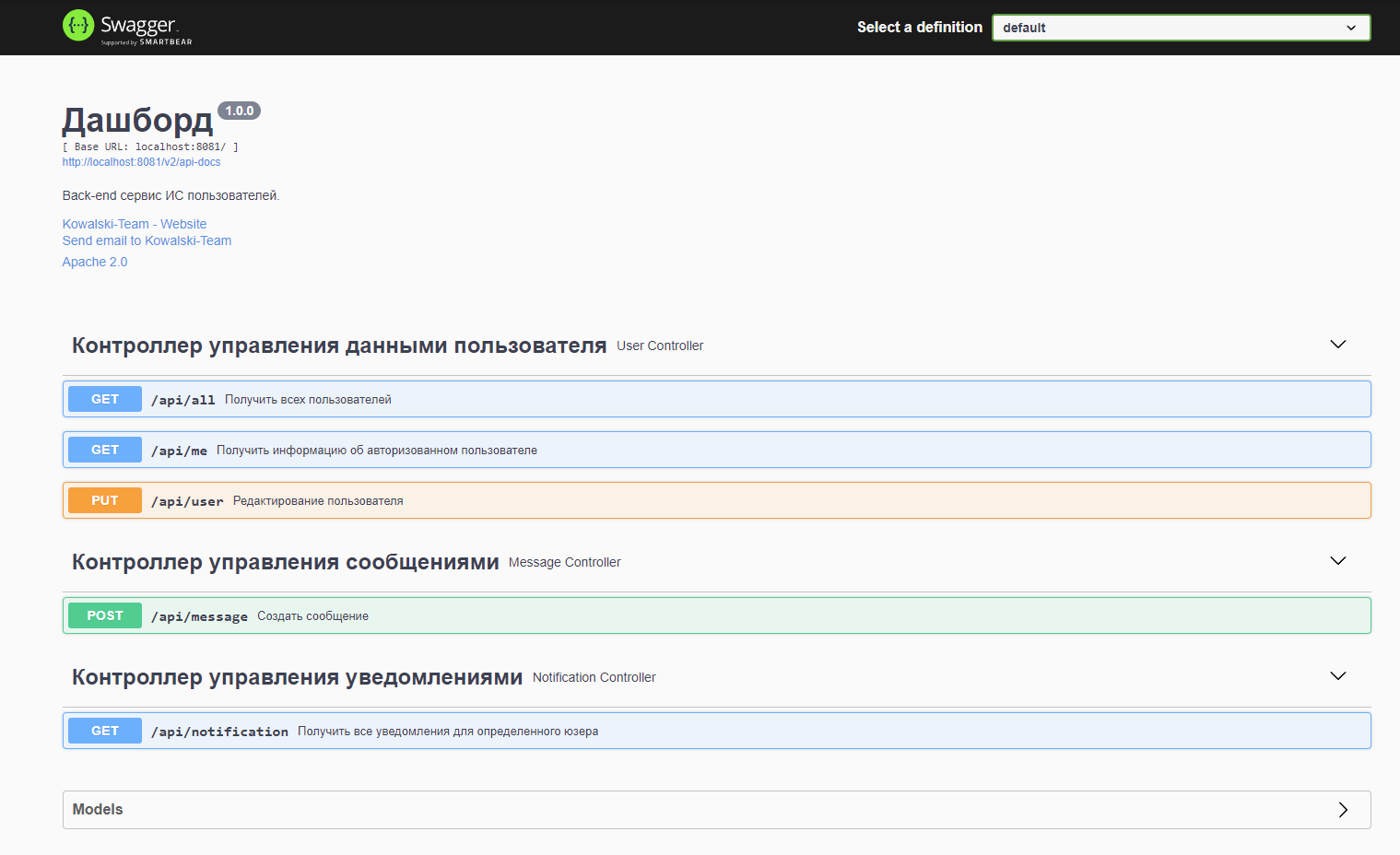
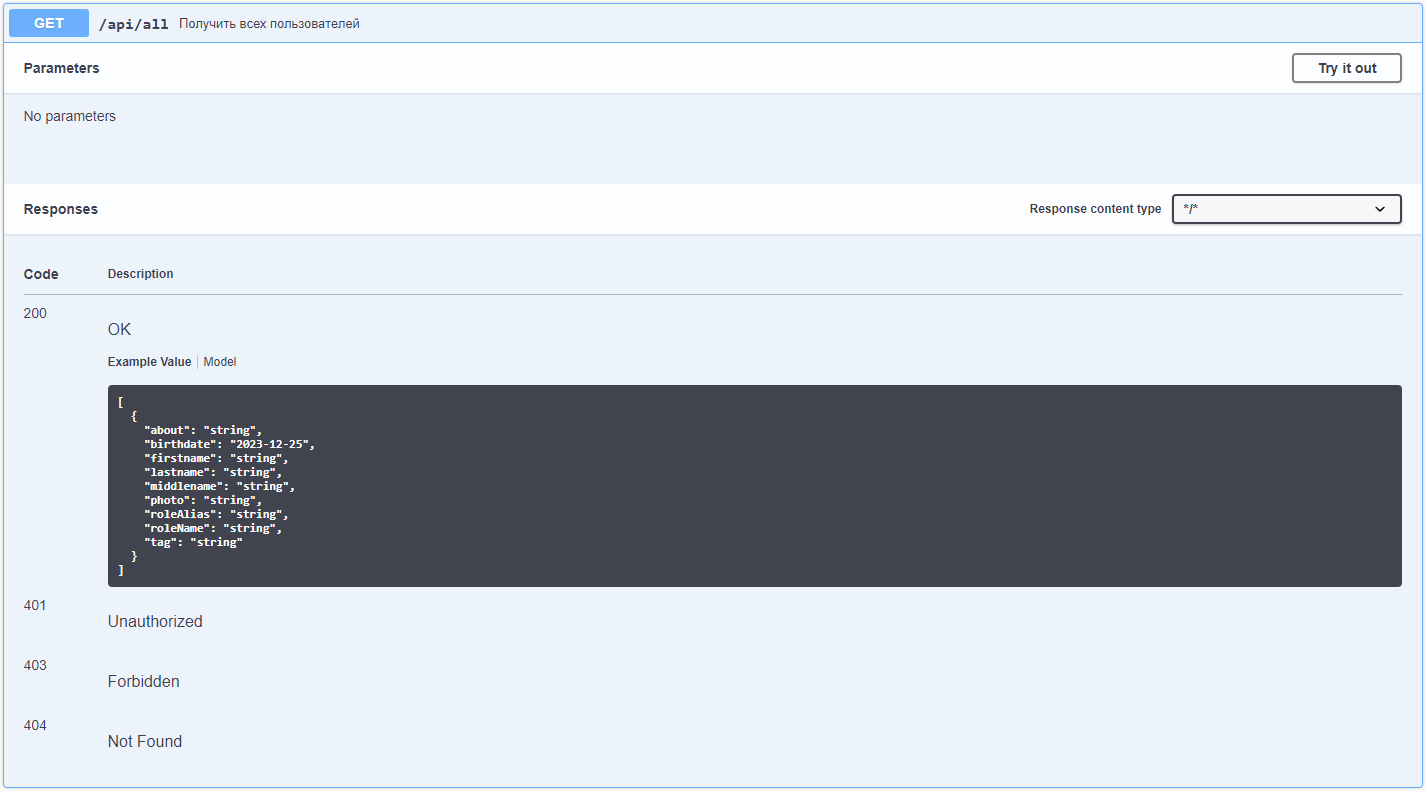


Рисунок 2 – Полный перечень методов на стороне северной части

Полный перечень моделей запросов и ответов представлена на рисунках 3-7.

  
Рисунок 3 – Модель запроса и ответа метода получения всех пользователей

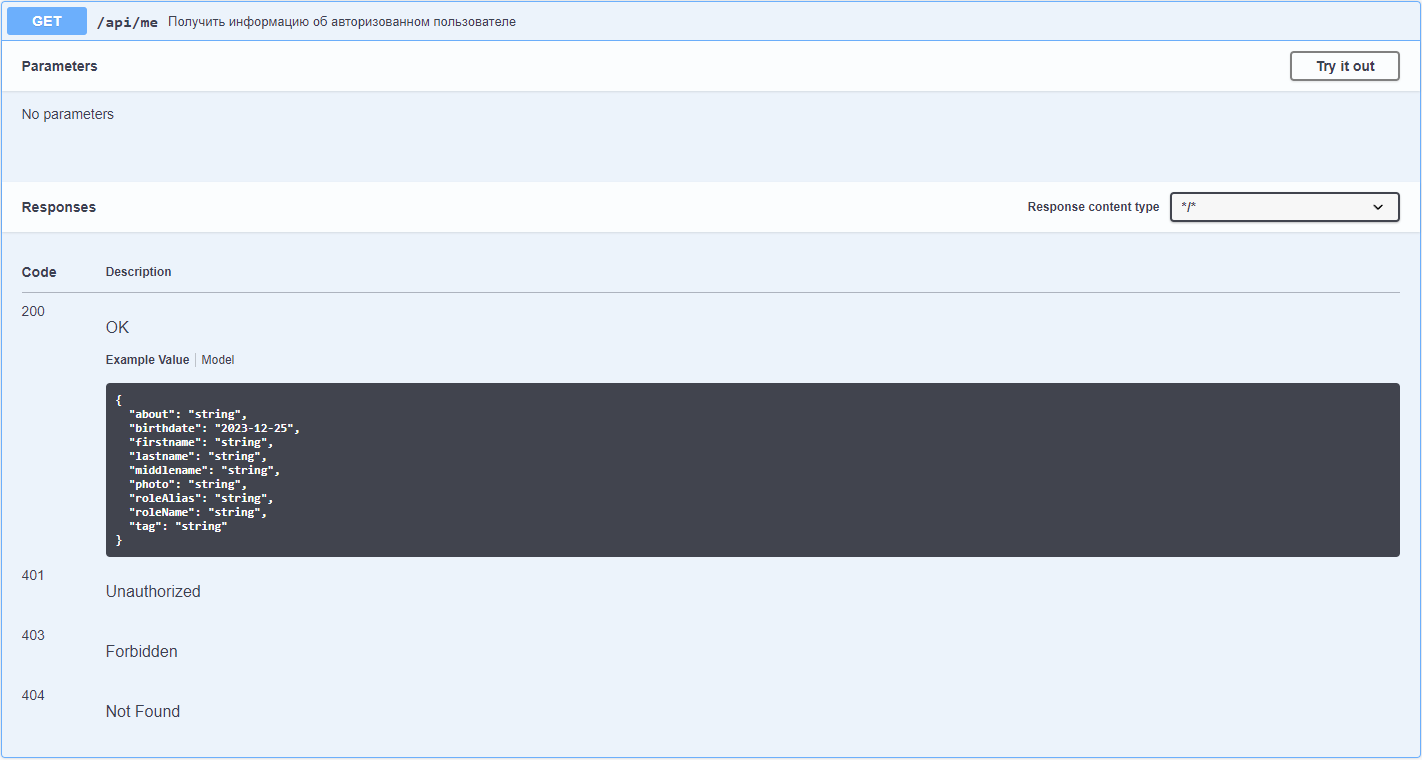


Рисунок 4 – Модель запроса и ответа метода получения информации об авторизованном пользователе

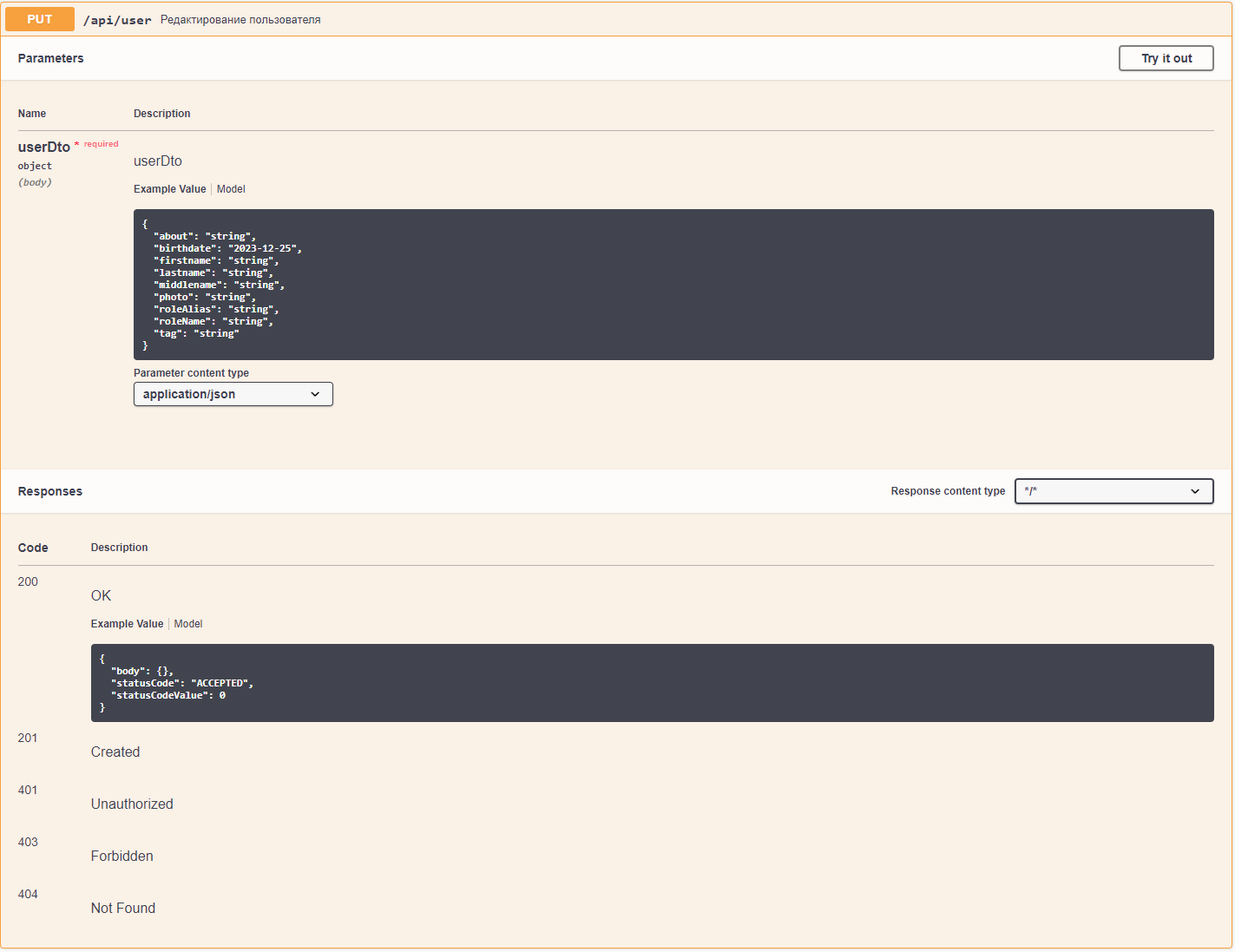


Рисунок 5 – Модель запроса и ответа метода редактирования пользователя

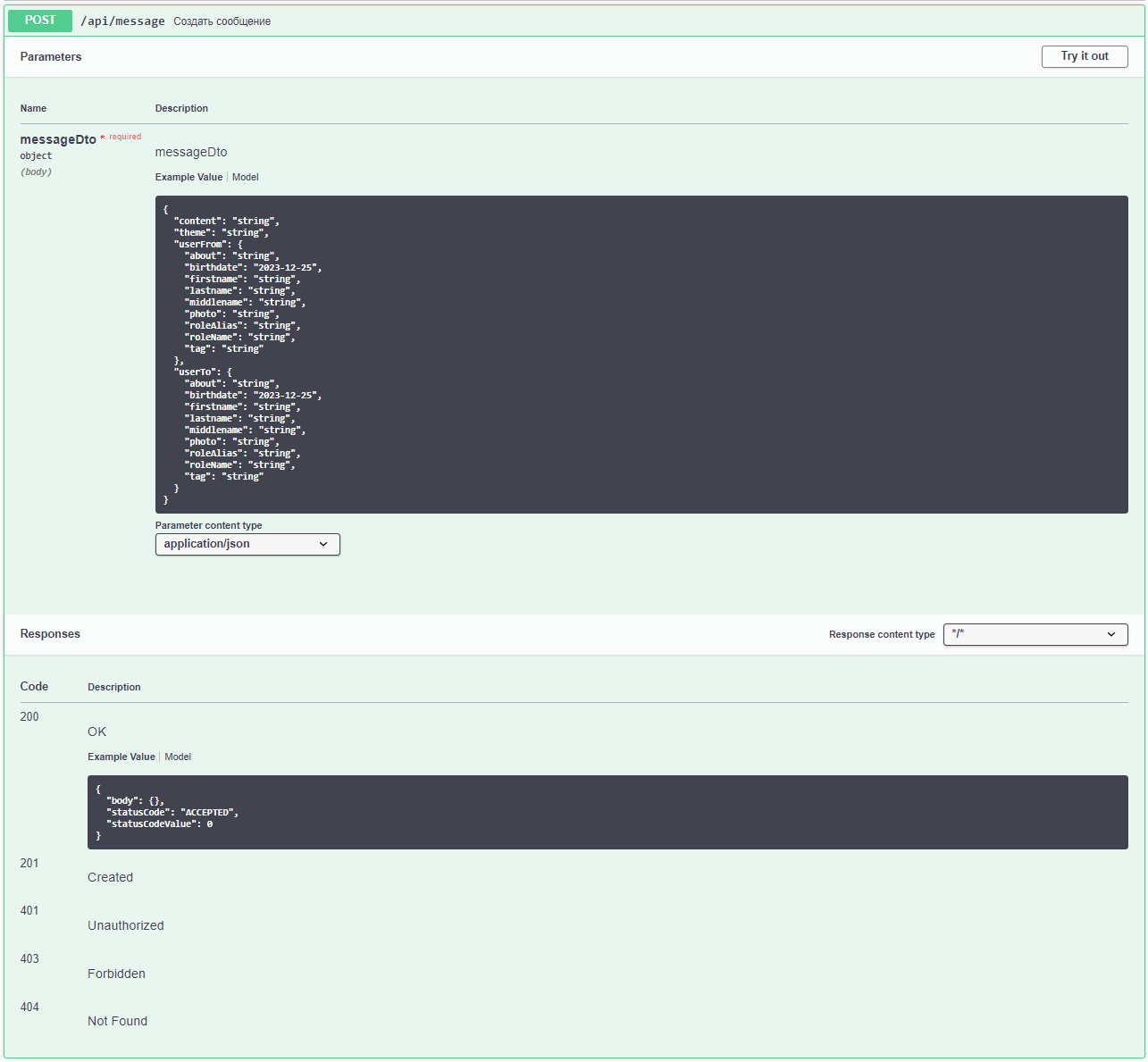


Рисунок 6 – Модель запроса и ответа метода создания сообщения

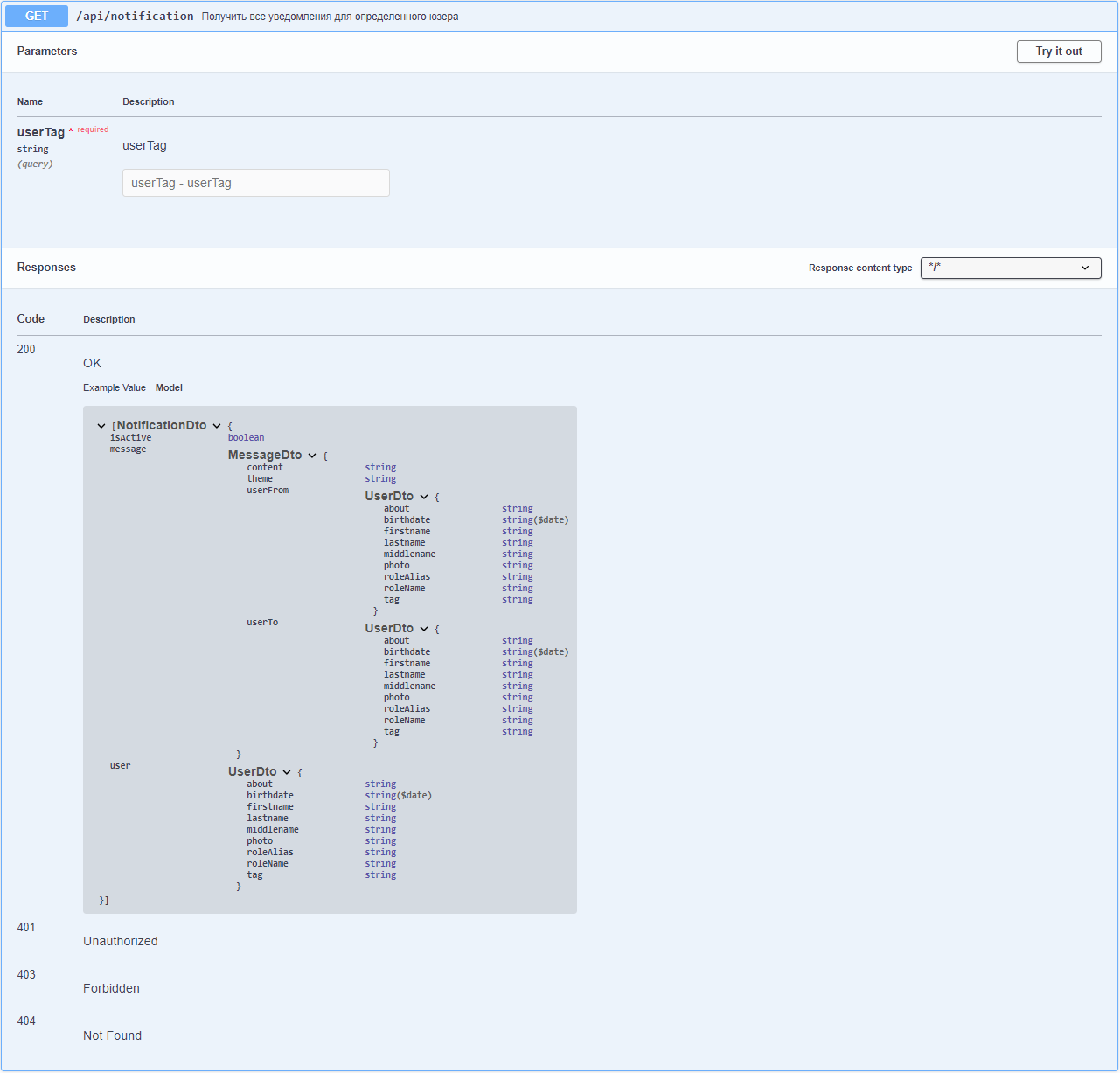


Рисунок 7 – Модель запроса и ответа метода получения всех уведомлений для определенного пользователя

**Разработка клиентской части**

Клиентская часть для взаимодействия с северной использует библиотеку OpenFeign.

OpenFeign - это библиотека для создания клиентов RESTful в Java. Она предоставляет декларативный способ определения HTTP-клиентов, которые могут взаимодействовать с удаленными службами. OpenFeign упрощает создание клиентов для взаимодействия с веб-службами и предоставляет удобные инструменты для обработки запросов и ответов.

В контексте клиентской части приложения OpenFeign используется по следующим причинам:

* *Декларативное определение клиента*: OpenFeign позволяет определить интерфейс, который представляет собой удаленный сервис, используя аннотации. Это делает код более читаемым и легко поддерживаемым, так как вы можете сосредоточиться на логике приложения, а не на деталях конфигурации HTTP-клиента.
* *Интеграция с балансировкой нагрузки и отказоустойчивостью*: OpenFeign интегрируется с инструментами балансировки нагрузки и отказоустойчивости, такими как Ribbon. Это позволяет легко организовывать и обеспечивать отказоустойчивость при взаимодействии с различными экземплярами удаленных сервисов.
* *Легкость использования и интеграция с Spring*: OpenFeign является частью экосистемы Spring Cloud и может легко интегрироваться с другими инструментами Spring. Это обеспечивает единый подход к разработке микросервисов в рамках приложения, построенного с использованием Spring.
* *Поддержка аннотаций для настройки запросов*: OpenFeign предоставляет аннотации для конфигурации запросов, таких как @RequestMapping и @RequestParam, что упрощает формирование запросов и обработку параметров.

Применение OpenFeign устраняет необходимость вручную создавать HTTP-клиенты, обрабатывать кодирование и декодирование запросов и ответов, а также позволяет легко интегрироваться с другими инструментами, используемыми в приложениях, такими как балансировка нагрузки и отказоустойчивость.

Для обработки визуальной части клиентской стороны – используется библиотека Thymeleaf.

Thymeleaf - это шаблонизатор для Java, предназначенный для создания веб-страниц с использованием HTML, XML или других текстовых форматов. Он широко используется в приложениях на основе Spring Framework для построения динамических веб-страниц. Thymeleaf предоставляет простой и интуитивно понятный синтаксис, который легко интегрируется с кодом Java.

Некоторые ключевые особенности Thymeleaf:

1. **Естественный синтаксис:** Thymeleaf разрабатывался с упором на естественность синтаксиса. Это позволяет разработчикам легко читать и понимать шаблоны.
2. **Поддержка HTML и XML:** Thymeleaf может использоваться для создания шаблонов как для HTML, так и для XML документов. Это особенно полезно в контексте веб-приложений, где может потребоваться работа с разными типами документов.
3. **Интеграция с Spring:** Thymeleaf стал популярным выбором для создания представлений в Spring-приложениях. Его можно интегрировать с Spring MVC для разработки веб-приложений.
4. **Поддержка выражений:** Thymeleaf позволяет встраивать выражения в HTML-код, что делает его мощным инструментом для динамического отображения данных. Эти выражения могут быть использованы для отображения данных из модели, выполнения условий и циклов, а также других динамических операций.
5. **Фрагменты и шаблоны:** Thymeleaf поддерживает использование фрагментов и шаблонов, что упрощает создание повторяющихся элементов интерфейса, таких как заголовки, подвалы и меню.

На рисунках 8-12 продемонстрированы HTML-страницы, которые используются на стороне клиентской части.

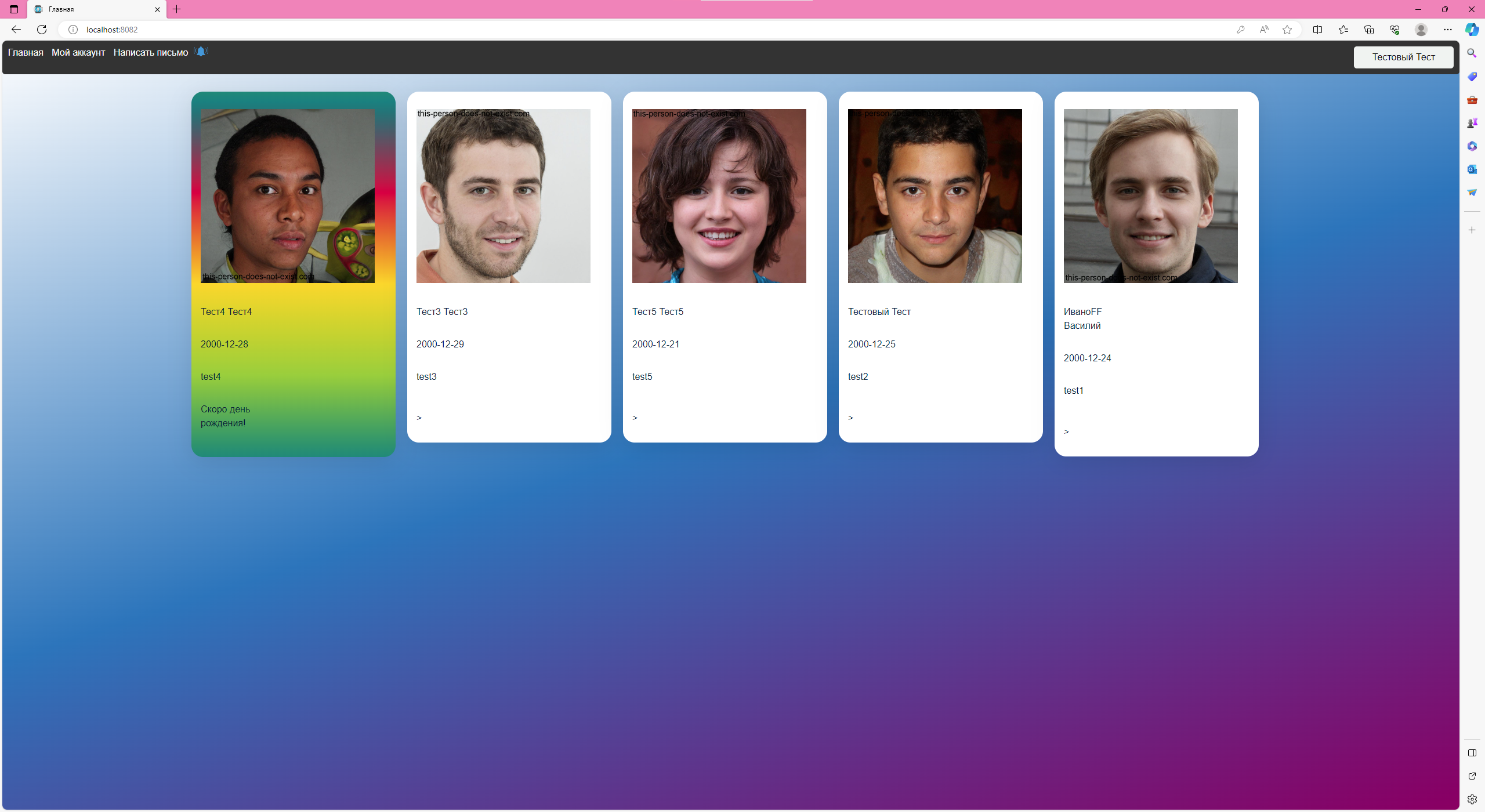


Рисунок 8 – Страница “Главное”

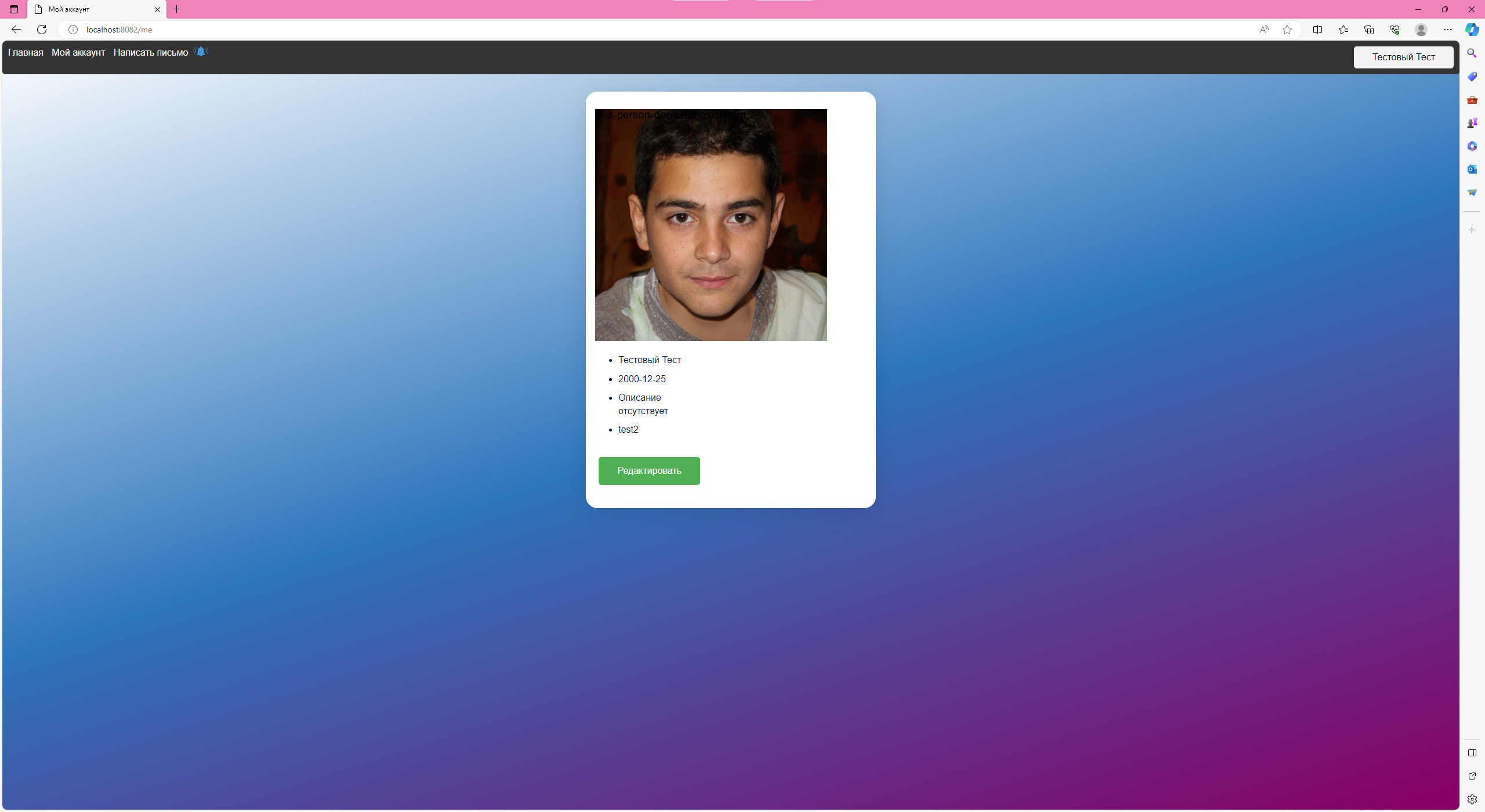


Рисунок 9 – Страница “Мой аккаунт”

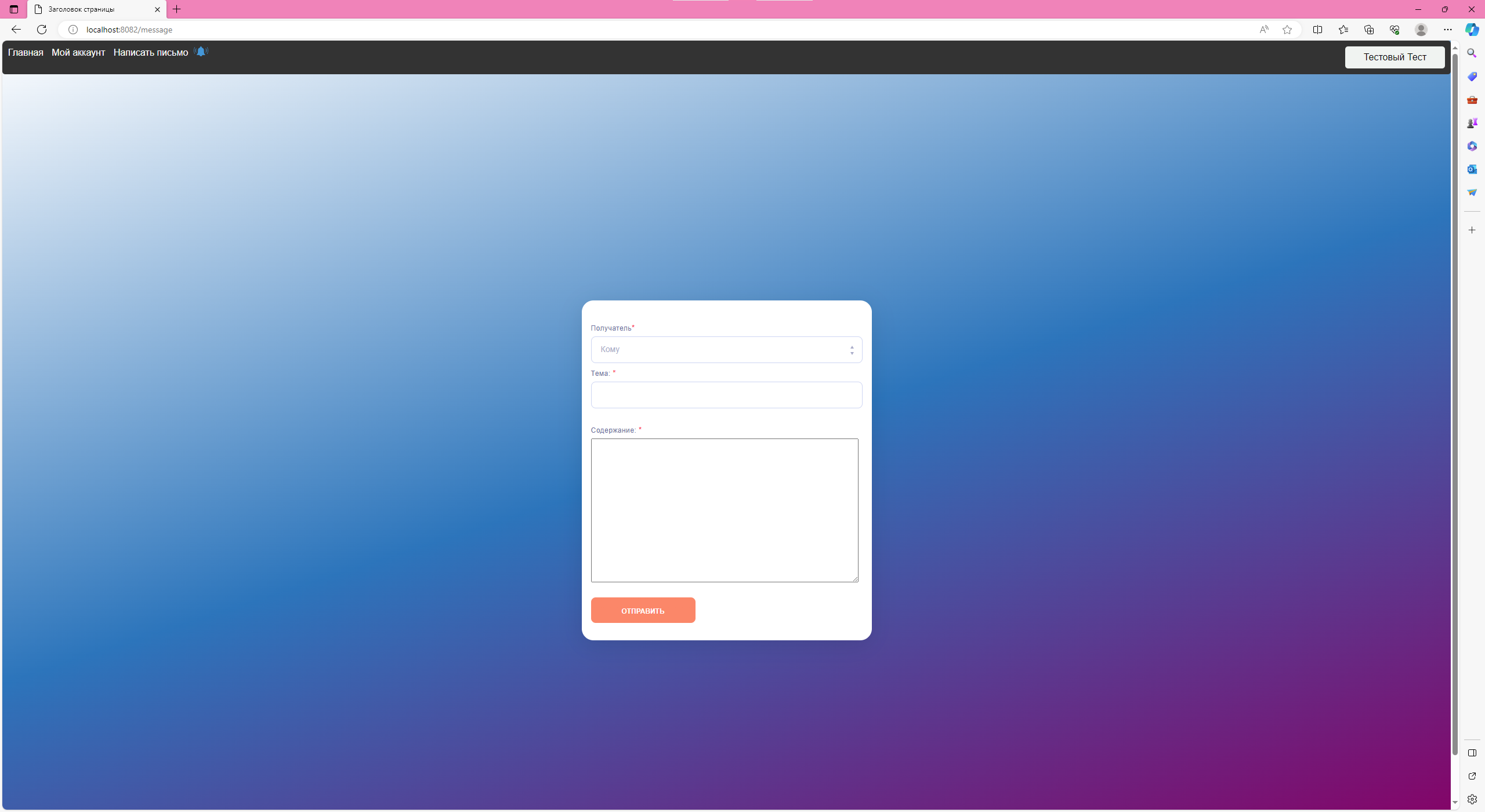


Рисунок 10 – Страница “Написать письмо”

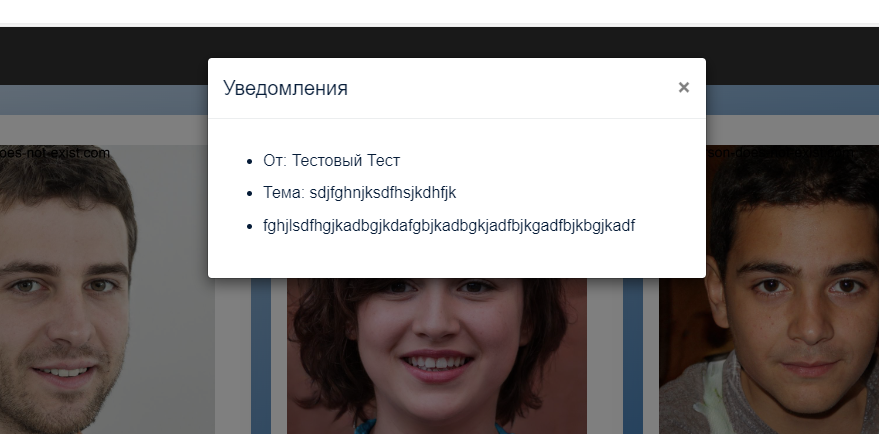


Рисунок 11 – Пример отображения модуля уведомлений

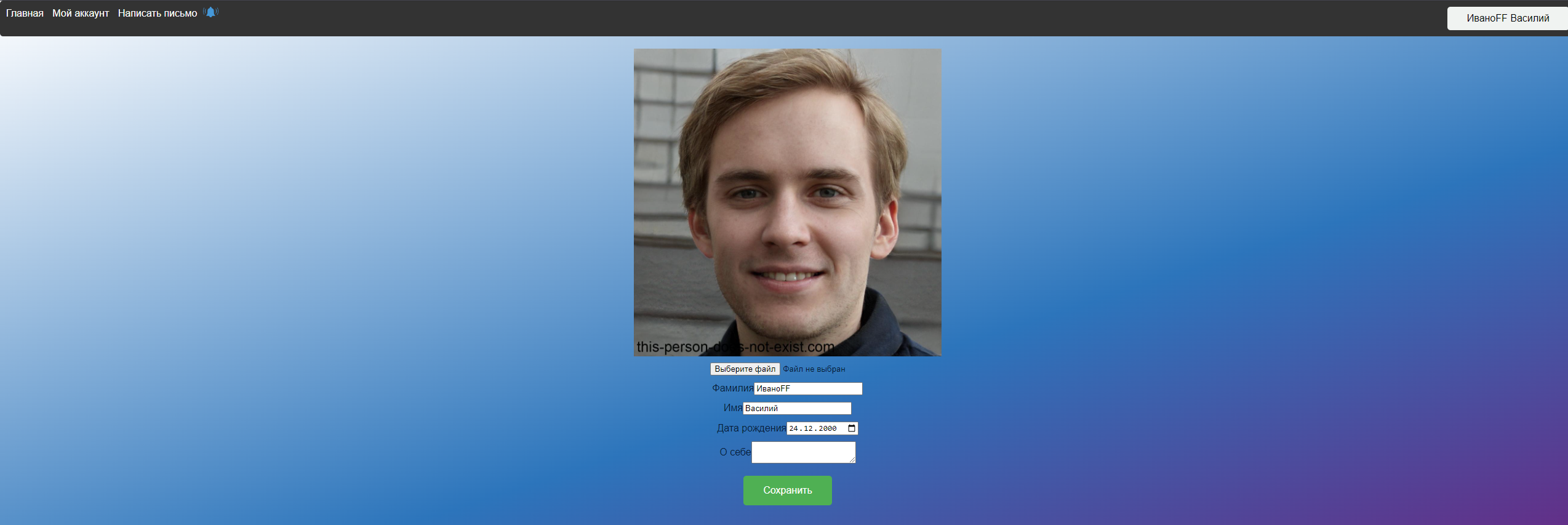


Рисунок 12 – Страница редактирование пользователя

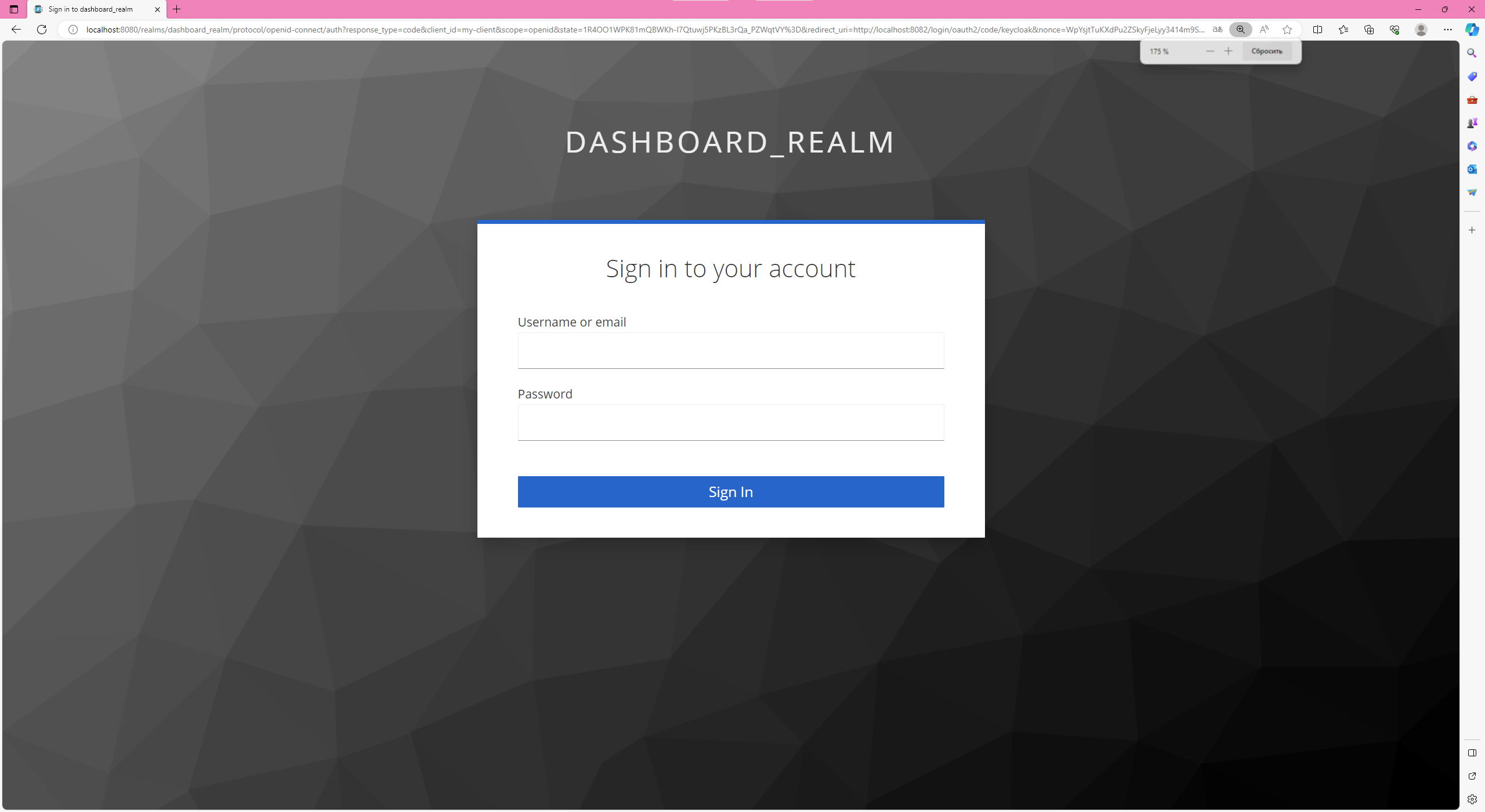


Рисунок 13 – Страница авторизации

**Разработка базы данных**

Для реализации функционала облачного приложения, а именно хранения информации о пользователях, включая дату рождения, фотографию, сообщения, роль, ФИО и тег, была использована СУБД – PostgreSQL.

PostgreSQL (или "Postgres") - это мощная и открытая реляционная система управления базами данных (СУБД). Она разрабатывается, поддерживается и распространяется сообществом разработчиков. Вот некоторые ключевые особенности PostgreSQL:

1. **Реляционная база данных:** PostgreSQL является реляционной базой данных, что означает, что данные организованы в виде таблиц, которые могут быть связаны между собой. Он поддерживает стандарт SQL и предоставляет множество возможностей для работы с данными.
2. **Открытый исходный код:** PostgreSQL распространяется под открытой лицензией, что означает, что его исходный код доступен для свободного использования, модификации и распространения. Это делает его популярным выбором для различных проектов и предприятий.
3. **Множество типов данных:** PostgreSQL предлагает обширный набор встроенных типов данных, включая числа, строки, даты, массивы, JSON и многие другие. Также возможно создание пользовательских типов данных.
4. **Многозадачность и масштабируемость:** PostgreSQL спроектирован для эффективной работы в многозадачной среде и обеспечивает хорошую масштабируемость. Он поддерживает одновременные подключения и может эффективно обрабатывать запросы от множества клиентов.
5. **Транзакционность:** PostgreSQL полностью транзакционна и поддерживает ACID-свойства (Atomicity, Consistency, Isolation, Durability), что обеспечивает надежность и целостность данных.
6. **Расширяемость:** Возможность создания пользовательских функций, операторов и агрегатных функций расширяет функциональность PostgreSQL и позволяет адаптировать его под конкретные требования проекта.
7. **Поддержка расширений:** PostgreSQL поддерживает концепцию расширений, что облегчает добавление дополнительных возможностей и функциональности без необходимости изменения ядра системы.
8. **JSON-поддержка:** PostgreSQL обладает встроенной поддержкой для работы с JSON-данными, что удовлетворяет современным требованиям в области хранения и обработки полуструктурированных данных.

На рисунке 13 предоставлена диаграмма сущностей базы данных.

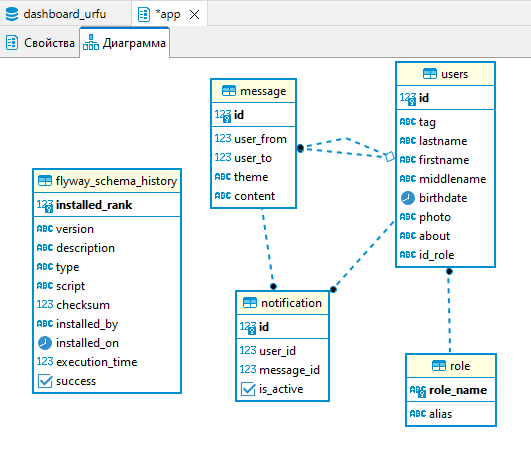


Рисунок 14 – Диаграмма сущностей базы данных

Таблица MESSAGE – сущность, хранящая письма пользователей.

Таблица NOTIFICATION – сущность, хранящая уведомления для пользователей.

Таблица ROLE – сущность, хранящая роли пользователей в системе.

Таблица USERS – сущность, хранящая информацию о пользователях в системе.

Таблица FLYWAY\_SCHEMA\_HISTORY – сущность, хранящая информацию о миграциях, создается и используется инструментом Flyway.

В таблицах 1-4 представлены описания атрибутов сущностей.

Таблица 1 – Описание атрибутов сущности MESSAGE

|  |  |  |
| --- | --- | --- |
| Название поля | Тип поля | Описание поля |
| id | int8 | Идентификатор сообщения |
| user\_to | int8 | Идентификатор пользователя, кому было отправлено сообщение |
| user\_from | int8 | Идентификатор пользователя, от кого было отправлено сообщение |
| theme | varchar(200) | Тема сообщения |
| content | varchar | Содержание сообщения |

Таблица 2 – Описание атрибутов сущности USERS

|  |  |  |
| --- | --- | --- |
| Название поля | Тип поля | Описание поля |
| id | int8 | Идентификатор пользователя |
| tag | varchar(50) | Тег для отправки сообщения в системе |
| lastname | varchar(50) | Фамилия пользователя |
| firstname | varchar(50) | Имя пользователя |
| middlename | varchar(50) | Отчество пользователя |
| birthdate | date | Дата рождения |
| photo | varchar | Фотография пользователя |
| about | varchar(1000) | Описание пользователя |
| id\_role | varchar(255) | Идентификатор роли пользователя |

Таблица 3 – Описание атрибутов сущности ROLE

|  |  |  |
| --- | --- | --- |
| Название поля | Тип поля | Описание поля |
| role\_name | varchar(255) | Наименование роли |
| alias | varchar(1000) | Алиас роли |

Таблица 4 – Описание атрибутов сущности NOTIFICATION

|  |  |  |
| --- | --- | --- |
| Название поля | Тип поля | Описание поля |
| id | int8 | Идентификатор уведомлений |
| user\_id | int8 | Идентификатор пользователя |
| message\_id | int8 | Идентификатор группы |
| is\_active | boolean | Показатель прочтения пользователем письма |

# **ИНСТРУКЦИЯ РАЗВЕРТЫВАНИЯ**

## **Docker**

Docker – это открытая платформа для разработки, доставки и запуска приложений в контейнерах. Контейнеры представляют из себя легковесные, автономные исполняемые пакеты, которые включают в себя все необходимые для работы приложения компоненты, включая код, библиотеки, зависимости и настройки. Docker обеспечивает стандартизированный способ упаковки, доставки и выполнения приложений в различных средах.

Некоторые ключевые концепции и возможности Docker:

1. **Контейнеры:** Контейнер представляет из себя исполняемый пакет, который включает в себя приложение и его зависимости, изолированные от операционной системы и других контейнеров. Docker использует технологию контейнеризации для обеспечения легковесной и быстрой виртуализации.
2. **Образы:** Образ представляет собой статическую версию контейнера, включая все необходимые файлы и настройки. Образы могут быть созданы с использованием Dockerfile, который содержит инструкции для построения образа.
3. **Dockerfile:** Dockerfile - это текстовый файл, который содержит инструкции для создания Docker-образа. Он определяет базовый образ, устанавливает зависимости, копирует файлы приложения и настраивает окружение.
4. **Docker Compose:** Docker Compose - это инструмент для определения и управления многоконтейнерных приложений. С помощью файла **docker-compose.yml** можно описать настройки, зависимости и параметры для развертывания нескольких контейнеров как единого приложения.
5. **Реестр Docker:** Реестр Docker - это хранилище для образов Docker. Docker Hub - это общедоступный реестр, который позволяет разработчикам обмениваться и использовать готовые образы.
6. **Мультиплатформенность:** Docker поддерживает множество платформ, включая Linux, Windows и macOS. Это обеспечивает консистентность окружения разработки, тестирования и развертывания.
7. **Интеграция с оркестраторами:** Docker интегрируется с различными оркестраторами, такими как Kubernetes и Docker Swarm, для управления и автоматизации развертывания контейнеров в распределенных средах.

## **Развертывание системы в Docker**

Для развертывания данной системой, в приложенном архиве находится файл docker-compose.yml. Он используется для разворачивания системы на платформе Docker.

Чтобы развернуть систему, нужно установить Docker Desktop или обычный Docker и ввести команду:

*docker-compose -f docker-compose.yaml up -d*

После развертывания, если используется Docker Desktop, то на странице “Containers” появятся контейнеры приложения (рисунок 14).

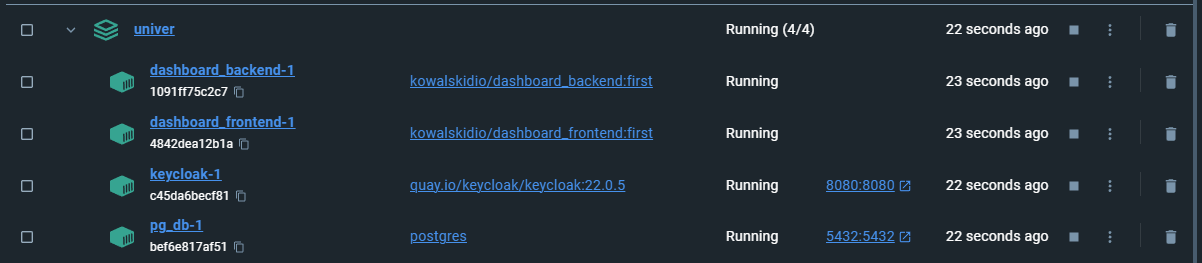


Рисунок 15 – Список развернутых контейнеров

Содержание docker-compose.yml:

*version: "3"*

*volumes:*

*pg\_project:*

*services:*

*pg\_db:*

*image: postgres*

*restart: always*

*environment:*

*- POSTGRES\_PASSWORD=dashboard\_backend*

*- POSTGRES\_USER=dashboard\_backend*

*- POSTGRES\_DB=dashboard\_urfu*

*volumes:*

*- pg\_project:/var/lib/postgresql/data*

*ports:*

*- ${POSTGRES\_PORT:-5432}:5432*

*keycloak:*

*image: quay.io/keycloak/keycloak:22.0.5*

*ports:*

*- "8080:8080"*

*environment:*

*KEYCLOAK\_ADMIN: admin*

*KEYCLOAK\_ADMIN\_PASSWORD: admin*

*command:*

*- start-dev*

*- --import-realm*

*volumes:*

*- /home/keycloak/realm.json:/opt/keycloak/data/import/realm.json*

*dashboard\_backend:*

*image: kowalskidio/dashboard\_backend:first*

*dashboard\_frontend:*

*image: kowalskidio/dashboard\_frontend:first*

В предоставленном файле **docker-compose.yml** определены сервисы для трех компонентов вашего приложения: PostgreSQL, Keycloak, и два компонента фронтенда и бэкенда для дашборда. Давайте рассмотрим каждый блок:

1. PostgreSQL (pg\_db):
   * **image: postgres** - использует официальный образ PostgreSQL из Docker Hub.
   * **environment** - устанавливает переменные окружения для настройки пользователя, пароля и базы данных PostgreSQL.
   * **volumes** - монтирует том (поименованный **pg\_project**) для хранения данных PostgreSQL.
   * **ports** - прокидывает порт PostgreSQL из контейнера в хост (может использовать переменную окружения **POSTGRES\_PORT**, иначе используется 5432).
2. Keycloak**:**
   * **image: quay.io/keycloak/keycloak:22.0.5** - использует образ Keycloak с Docker Hub.
   * **ports** - прокидывает порт 8080 Keycloak из контейнера в хост.
   * **environment** - устанавливает переменные окружения для настройки администратора Keycloak.
   * **volumes** - монтирует файл **realm.json** внутрь контейнера, который будет использован для импорта конфигурации Keycloak.
3. Dashboard Backend (dashboard\_backend):
   * **ports** - прокидывает порт 8081 из контейнера в хост.
   * **image** - используется образ с именем **kowalskidio/dashboard\_backend:first**.
4. Dashboard Frontend (dashboard\_frontend):
   * **ports** - прокидывает порт 8082 из контейнера в хост.
   * **image** - используется образ с именем **kowalskidio/dashboard\_frontend:first**.

## **Настройка Keycloak**

После успешного развертывания приложения, перейдем к настройке Keycloak.

Откроем в браузере страницу Keycloak’a по адресу <http://localhost:8080/> (рисунок 15).

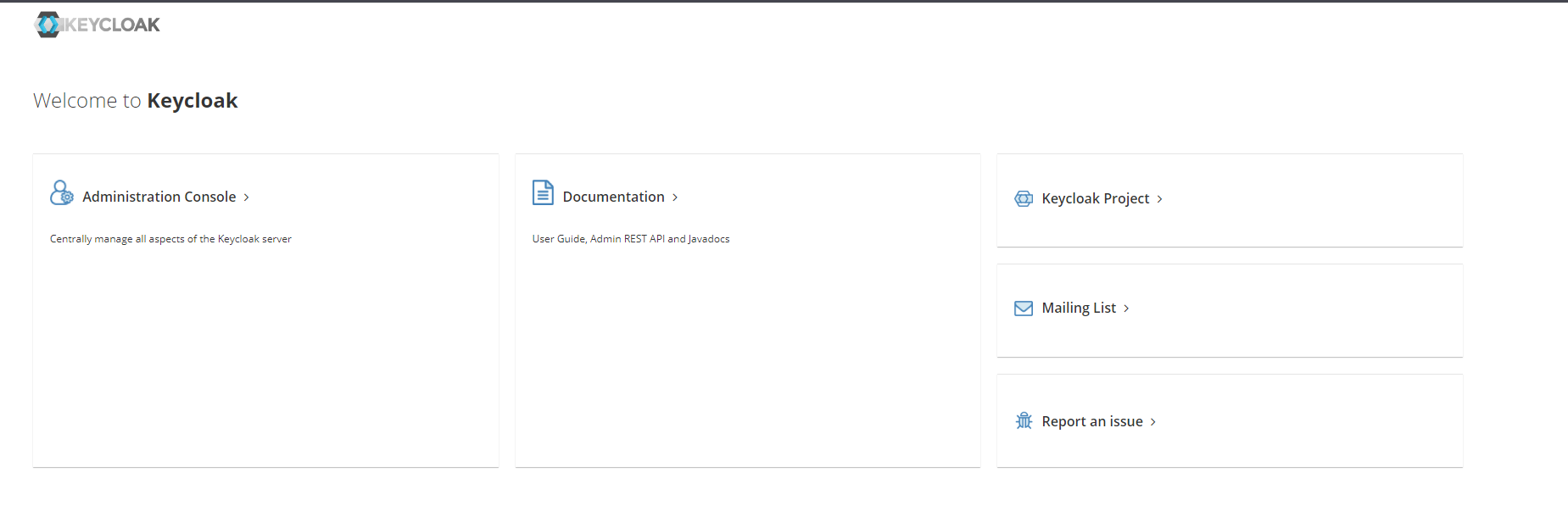


Рисунок 16 – Начальная страница Keycloak

Перейдем во вкладку “Administration Console” и создадим realm “dashboard\_realm” (рисунок 16).

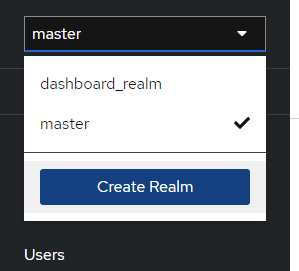


Рисунок 17 – Создание нового realm

Перейдем во вкладку Clients и создадим “my-client” (рисунок 17-18).

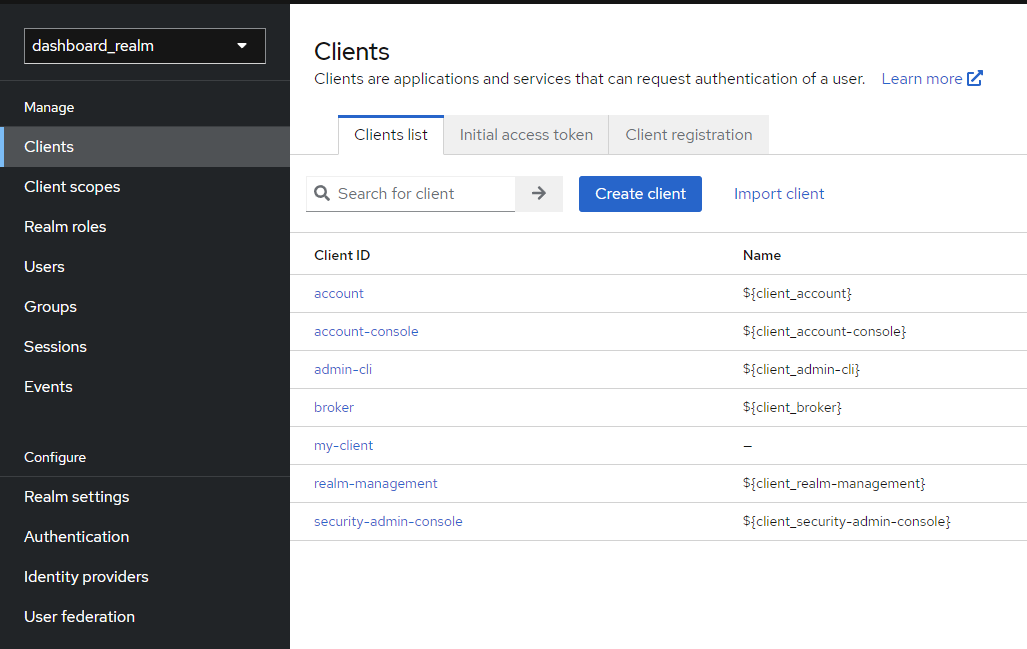


Рисунок 18 – Вкладка “Clients”

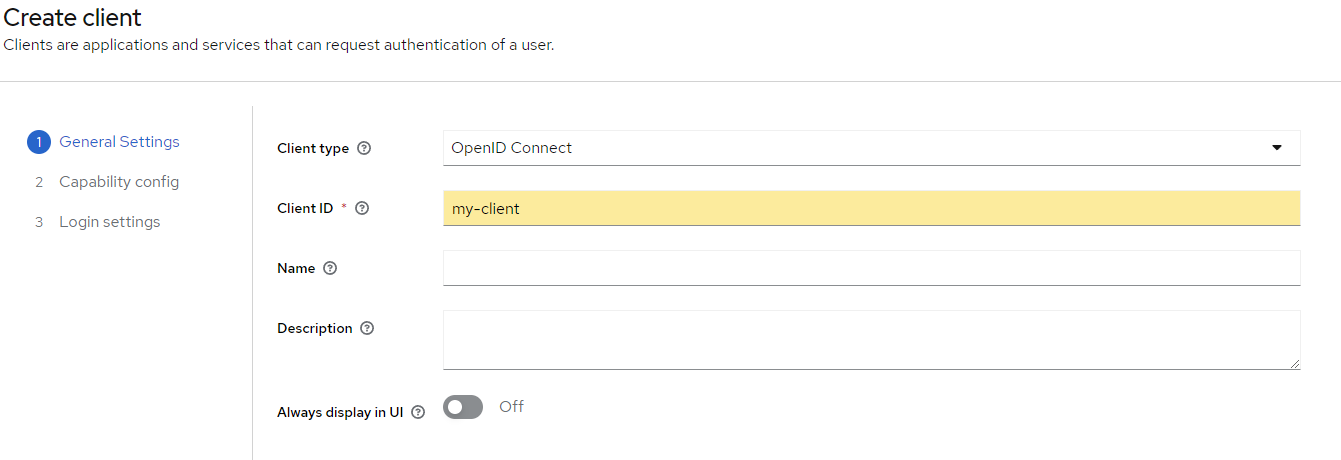


Рисунок 19 – Создание my-client

Перейдем в созданный ранее “my-client” и добавим адреса веб-приложений нашей системы в “Valid redirect URIs” (рисунок 19).

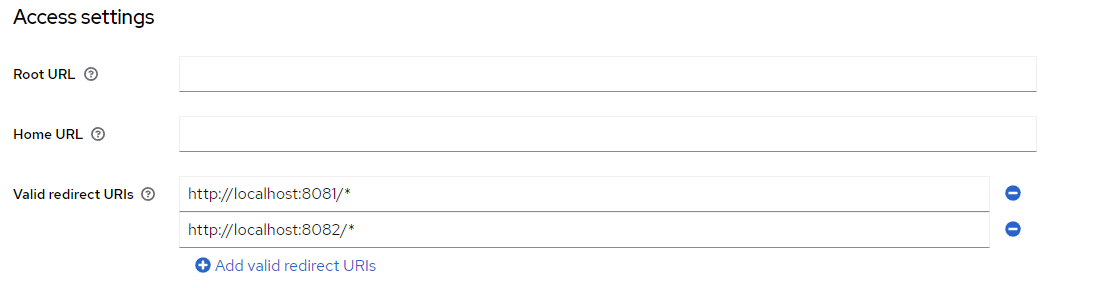


Рисунок 20 – Добавление адресов для перенаправления пользователей в наше веб-приложения

Создадим пользователя в нашем “dashboard\_realm”. Перейдем во вкладку “Users” и нажмем на кнопку “Add user” (рисунок 20).

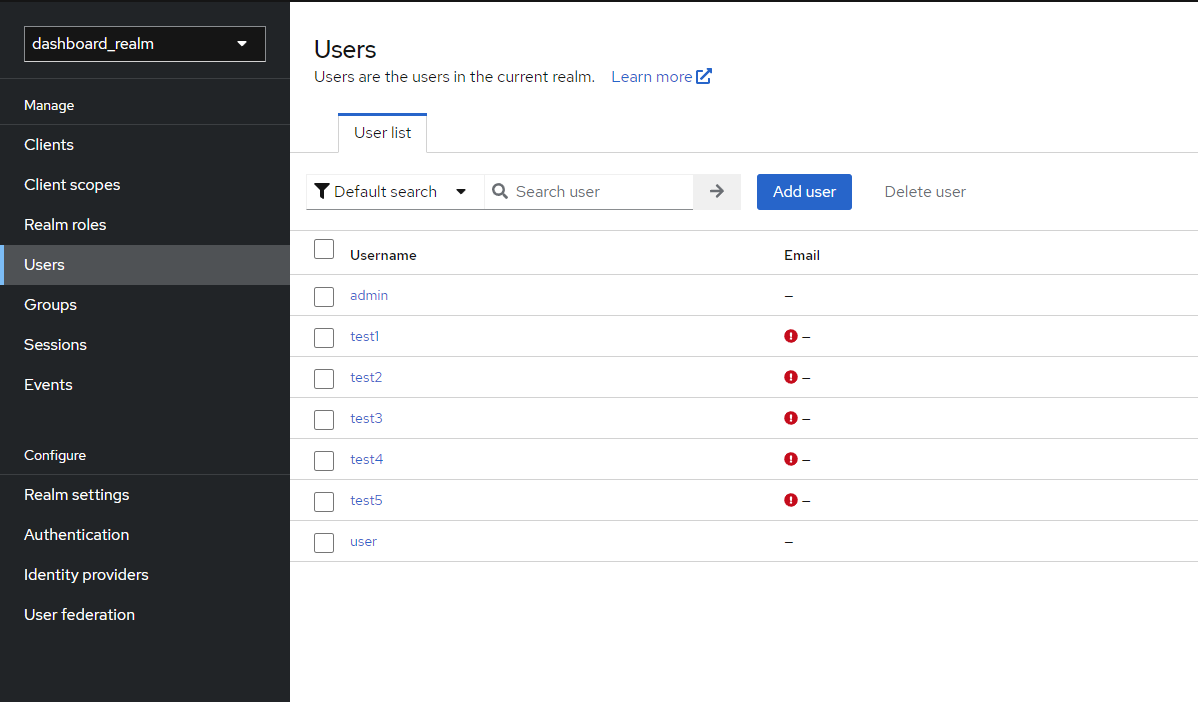
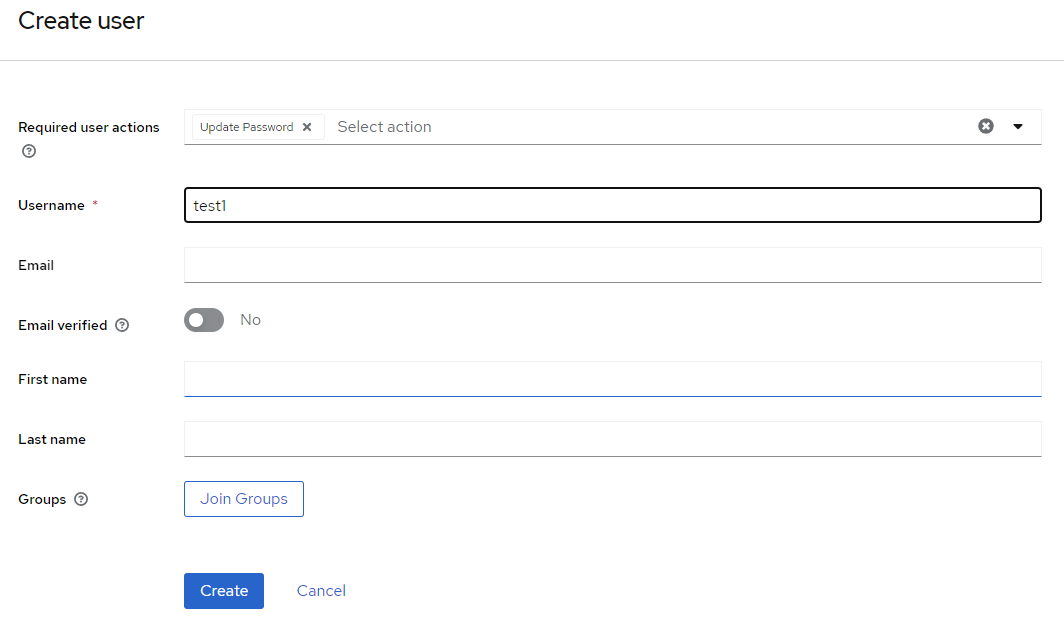


Рисунок 21 – Добавим пользователя в Keycloak

Установим в поле “Required user actions” значение “Update Password”, чтобы наш пользователь мог сам установить собственный пароль. Поле “Username” должен совпадать с наименованием пользователя в БД (рисунок 21).

  
Рисунок 22 – Создание пользователя в Keycloak

Keycloak настроен и система готова к использованию.

# **ЗАКЛЮЧЕНИЕ**

В процессе выполнения работы была разработана "облачная" система информирования пользователей о днях рождениях, в рамках которых были предусмотрены следующие пункты:

1. Возможность размещения фотографий и сведений о пользователях.
2. Отображение пользователя, у которых приближается день рождения, с фотографией на панели информирования.
3. Возможность отправки сообщений пользователям непосредственно в системе