МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка базы данных с использованием технологий шифрования данных для приложения обмена сообщениями Sweater»

Выполнил студент Храмых Владислав Олегович

(Ф.И.О.)

Руководитель проекта пр-стаж. Нистюк О.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультанты пр-стаж. Нистюк О.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер пр-стаж. Нистюк О.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2019

**Реферат**

Пояснительная записка курсового проекта состоит из 33 страницы, 19 рисунков, 1 приложение, 10 источников литературы.

Java, Spring Framework (Boot, Security, Data), Hibernate, Bootstrap, Font awesome 4, JavaScript, FreeMarker, Patterns (Repository, MVC, Dao), MySQL, метод хеширования SHA-512.

Основная цель курсового проекта: проектирование базы данных для студенческого приложения Sweater и изучения технологии шифрования данных.

Пояснительная записка состоит из введения, 8 глав, заключения.

Во введении представлена общая информация, дающая представление о предстоящей работе.

В первом разделе рассматриваются технические решения и формируются задачи к проектированию.

Во втором разделе рассматривается модель базы данных.

В третьем разделе рассматривается проектирование программного средства.

В четвертом разделе предоставлена информация о разработанных объектах базы данных.

В пятом разделе описываются функции импорта/экспорта данных в/из формат xml.

В шестом разделе описывается тестирование производительности базы данных.

Седьмой раздел содержит информацию о примененной технологии.

Восьмой раздел содержит руководство пользователя для разработанного клиентского приложения.

В заключении описывается результат работы над курсовым проектом, выполненные цели.

**Содержание**

[Введение 4](#_Toc27193639)

[1 Обзор технических решений и формирование требований 5](#_Toc27193640)

[1.1 Обзор средств разработки 5](#_Toc27193641)

[1.2 Постановка задачи 6](#_Toc27193642)

[2 Разработка модели базы данных 7](#_Toc27193643)

[3 Проектирование программного средства 10](#_Toc27193644)

[3.1 Обобщенная структура проекта и детализация функций проектируемого программного средства 10](#_Toc27193645)

[3.2 Описание разрабатываемой функциональности программного средства 11](#_Toc27193646)

[4 Разработка необходимых объектов 12](#_Toc27193647)

[4.1 Таблицы 12](#_Toc27193648)

[4.2 Пользователи 12](#_Toc27193649)

[4.3 Хранимые процедуры 13](#_Toc27193650)

[5 Описание процедур импорта и экспорта данных 16](#_Toc27193651)

[5.1 Описание процедуры импорта данных 16](#_Toc27193652)

[5.2 Описание процедуры экспорта данных 16](#_Toc27193653)

[6 Тестирование производительности 17](#_Toc27193654)

[6.1 Описание типов индексов 17](#_Toc27193655)

[6.2 Пример использования индексов в базе данных 18](#_Toc27193656)

[7 Описание технологий 19](#_Toc27193657)

[7.1 Описание и реализация технологии шифрования данных 19](#_Toc27193658)

[7.2 Описание и реализация технологии хранения мультимедийных типов 19](#_Toc27193659)

[8 Руководство пользователя 21](#_Toc27193660)

[8.1 Для администратора 23](#_Toc27193661)

[8.2 Для пользователя 25](#_Toc27193662)

[Заключение 30](#_Toc27193663)

[Список используемых источников 31](#_Toc27193664)

[Приложение А 32](#_Toc27193665)

ФИО

Подпись

Дата

Лист

1

КП 00.01.ПЗ

Разраб.

Агапкина Д.С.

Провер.

Бондарчик Е.Н.

Н. контр.

Бондарчик Е.Н.

Утверд.

Бондарчик Е.Н.

Реферат

Лит.

Листов

1

БГТУ 71172546, 2019

У

# Введение

База данных (БД) — это организованная структура, предназначенная для хранения, изменения и обработки взаимосвязанной информации.

В контексте баз данных стоит рассмотреть понятие СУБД. Система управления базами данных (СУБД) — это комплекс программных средств, необходимых для создания структуры новой базы, ее наполнения, редактирования содержимого и отображения информации. Кроме того, СУБД имеет определенные режимы работы и систему команд. На основе СУБД создаются и функционируют информационные системы.

Основные функции СУБД:

* определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки;
* предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация);
* обеспечение логической и физической независимости данных;
* управление полномочиями пользователей на доступ к базе данных;
* синхронизация работы нескольких пользователей;
* управление ресурсами среды хранения;
* поддержка языков базы данных.

В современном мире большое значение имеет работа с данными. Для хранения данных используются различные системы управления базами данных: MS SQL Server, Oracle, MySQL и так далее. Практически все большинство крупных приложений и сервисов используют эти СУБД для хранения данных.

Для хранения большого количество данных приложение «Sweater» должно взаимодействовать с базой данных. В качестве СУБД будет использоваться программный продукт MySQL 5.7. Соответственно, для данного программного средства будет создана база данных, в которой будут созданы необходимые для корректного функционирования приложения объекты.

В приложении необходимо реализовать возможность добавления, оценивания, изменения сообщений, подписок/отписок от пользователей. Для удобства работы пользователя с приложением нужно привести руководство пользователя.

Необходимо создать основные объекты, такие как таблицы, процедуры, функции, пользователи, индексы, триггеры. В базе данных должна быть реализована возможность экспорта и импорта в/из xml данных. Также после создание объектов и добавления данных в таблицы, нужно протестировать базу данных на производительность, вследствие чего добавить необходимые индексы для таблиц. В качестве технологии для курсового проекта была выбрана технология шифрования данных. [1]

# Обзор технических решений и формирование требований

## Обзор средств разработки

Языком разработки пользовательского интерфейса для курсового проекта является Java. При выполнении курсового проекта были использованы принципы и приемы ООП, а также технологии Spring Framework (Boot, Security, Data), Hibernate, Bootstrap.

Spring Framework — один из самых популярных фреймворков для создания веб-приложений на Java.

**Spring Boot** — это полезный проект, целью которого является упрощение создания приложений на основе Spring. Он позволяет наиболее простым способом создать web-приложение.

Spring Security – это framework, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework.

Spring Data — дополнительный удобный механизм для взаимодействия с сущностями базы данных, организации их в репозитории, извлечение данных, изменение, в каких то случаях для этого будет достаточно объявить интерфейс и метод в нем, без имплементации.[2]

Bootstrap - это инструментарий с открытым исходным кодом для разработки адаптивных приложения с современным интерфейсом с помощью HTML, CSS и JS.

Если традиционные средства JDBC позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Hibernate представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Эта технология предоставляет нам набор классов, через которые мы можем отправлять запросы к базам данных, устанавливать подключения, получать ответ от базы данных и производить ряд других операций.

MySQL — [свободная](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%9F%D0%9E) [реляционная система управления базами данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94). Разработку и поддержку MySQL осуществляет корпорация [Oracle](https://ru.wikipedia.org/wiki/Oracle), получившая права на торговую марку вместе с поглощённой [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems). Продукт распространяется как под [GNU General Public License](https://ru.wikipedia.org/wiki/GNU_General_Public_License), так и под собственной коммерческой лицензией. MySQL является решением для малых и средних приложений. Входит в состав серверов [WAMP](https://ru.wikipedia.org/wiki/WAMP), [AppServ](https://ru.wikipedia.org/w/index.php?title=AppServ&action=edit&redlink=1" \o "AppServ (страница отсутствует)), [LAMP](https://ru.wikipedia.org/wiki/LAMP) и в портативные сборки серверов [Денвер](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BD%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0)), [XAMPP](https://ru.wikipedia.org/wiki/XAMPP), [VertrigoServ](https://ru.wikipedia.org/w/index.php?title=VertrigoServ&action=edit&redlink=1" \o "VertrigoServ (страница отсутствует)). Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.[

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа [MyISAM](https://ru.wikipedia.org/wiki/MyISAM" \o "MyISAM), поддерживающие полнотекстовый поиск, так и таблицы [InnoDB](https://ru.wikipedia.org/wiki/InnoDB" \o "InnoDB), поддерживающие транзакции на уровне отдельных записей. [3]

Немаловажно и то, что данная СУБД распространяется бесплатно и представляет собой программное обеспечение с открытым кодом. За счет этого вы можете вносить свои изменения и модифицировать код, что весьма полезно для веб-мастеров.

Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

Эта СУБД может хранить огромное количество записей в своих таблицах. Этим обеспечивается возможность создания крупных проектов. Немаловажным показателем является также простая и эффективная система безопасности MS SQL. Ну и, конечно, бесспорным плюсом данной системы является то, что она распространяется совершенно бесплатно.

Приложение было реализовано с помощью шаблона проектирования MVС (Model-View-Controller).

Паттерн MVC позволяет отделить логику приложения от визуальной части (представления). Данный паттерн является архитектурным, то есть он задает общую архитектуру приложения. MVC состоит из трех компонентов: модели (Model), контроллера (View) и представления (View).

Итогом применения паттерна MVС является функциональное разделение приложения на три компонента, которые проще разрабатывать и тестировать, а также в дальнейшем модифицировать и поддерживать.

## Постановка задачи

Основной задачей курсового проекта является проектирование и разработка базы данных для приложения Sweater, которое ориентировано абсолютно для любого человека, который хочет делиться своими впечатлениями от чего-либо, или который хочет просто что-то показать или сказать. Благодаря механизму подписок/отписок пользователю предоставляется возможность отслеживания новых постов от тех пользователей, которые ему интересны, и также он может оценить то либо иное сообщение.

Таким образом, в данном курсовом проекте требовалось реализовать следующие задачи:

* Сохранять пользовательскую информацию в базе данных;
* Получать информацию из базы данных;
* Создание пользовательского интерфейса для взаимодействия с базой данных;
* Регистрация и авторизация пользователей;
* Удаление при необходимости информации из базы данных;
* Получение информации в xml-формате;
* Разделение ролей пользователей;
* Применение технологии шифрования данных.

# Разработка модели базы данных

## Модель базы данных

Модель базы данных — то же, что и схема базы данных, то есть описания содержания, структуры и ограничений целостности, используемые для создания и поддержки базы данных. С помощью модели данных могут быть представлены информационные объекты и взаимосвязи между ними.

При проектировании реляционной базы данных необходимо исследовать предметную область с целью определения объектов, нормализовать данные и установить связи между ними.

Нормализация данных − это процесс, в результате выполнения которого таблицы базы данных проверяются на наличие зависимостей между столбцами таблицы. Если такие зависимости существуют, то таблица разделяется на несколько таблиц. Если одна из этих сгенерированных таблиц все еще содержит зависимости данных, то процесс нормализации должен повторяться, пока не будут устранены все зависимости. Данные подлежат нормализации.

Чтобы таблица соответствовала 1-й нормальной форме (1NF), необходимо, чтобы все значения ее полей были неделимыми и не вычисляемыми, а все записи уникальными (не должно быть полностью совпадающих строк).

Чтобы таблица соответствовала 2-й нормальной форме (2NF), необходимо, чтобы она уже находилась в 1-й нормальной форме и все не ключевые поля полностью зависели от ключевого.

Для перехода к 3-й нормальной форме (3NF), необходимо обеспечить, чтобы все таблицы БД находились во 2-й нормальной форме, все не ключевые поля в таблицах зависели только от ключа таблицы и не зависели друг от друга.

Для базы приложения Sweater было разработано 8 таблиц.

Таблица USER содержит информацию о ролях пользователя:

* ID – первичный ключ, идентификатор пользователя;
* ACTIVATION\_CODE – код активации аккаунта;
* ACTIVE ­­­­– показывает, активен ли пользователь;
* EMAIL ­– адрес электронной почты;
* PASSWORD – содержит хэш пароля пользователя;
* USERNAME – содержит уникальный логин пользователя;

Таблица MESSAGE содержит информацию о сообщениях:

* ID – первичный ключ, идентификатор сообщения;
* FILENAME– содержит название прикреплённого файла;
* TAG– тег сообщения;
* TEXT–текст сообщения;
* USER\_ID – идентификатор пользователя, разместившего сообщение.

Таблица USER\_ROLE содержит информацию ролях пользователей:

* USER\_ID – идентификатор пользователя;
* ROLES – присвоенные пользователю привилегии.

Таблица USER\_SUBSCRIPTIONS содержит информацию о подписках пользователей:

* CHANNEL\_ID – идентификатор пользователя, на которого подписываются;
* PROFESSION \_NAME – идентификатор пользователя, который подписывается.

Таблица MESSAGE\_LIKES содержит информацию о группах:

* USER\_ID – идентификатор пользователя, который оценил сообщение;
* MESSAGE\_ID – идентификатор оценённого сообщения.

Таблица LOG\_MESSAGE содержит историю добавленных и отредактированных сообщений:

* ID – идентификатор записи;
* MESSAGE – сообщение о совершении действия;
* TIME – время изменения/добавления.
* ROW\_ID – идентификатор сообщения, надо которым производилось действие.

Таблица LOG\_USER историю редактирования и добавления пользователей:

* ID – идентификатор записи;
* MESSAGE – сообщение о совершении действия;
* TIME – время изменения/добавления.
* ROW\_ID – идентификатор пользователя, надо которым производилось действие.

Таблица LOG\_SUBS содержит историю подписок/отписок пользователей:

* ID – идентификатор записи;
* MESSAGE – сообщение о совершении действия;
* TIME – время совершения действия.

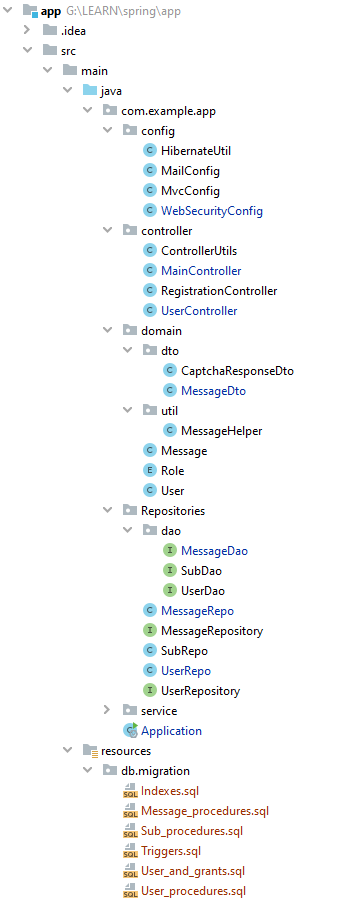
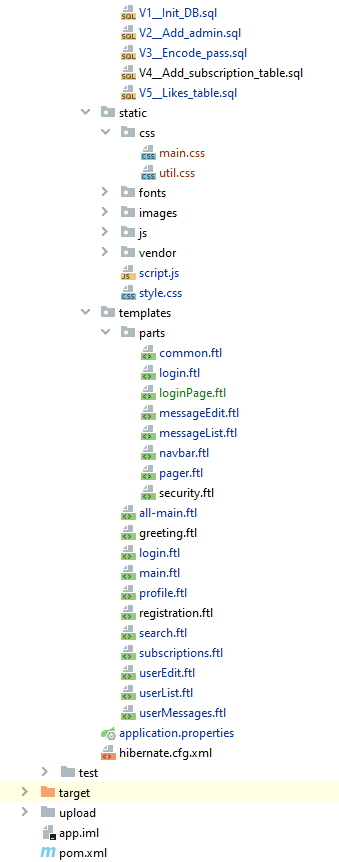
Таким образом, база данных на представленной схеме в полной мере описывает необходимые таблицы для решения поставленной задачи данного курсового проекта.

# Проектирование программного средства

## Обобщенная структура проекта и детализация функций проектируемого программного средства

Решение представлено проектом: Sweater. Для выполнения действий, нужно запустить проект и открыть его в браузере.

Структура проекта программного средства представлена на рисунке 3.1.1.

### Рисунок 3.1.1 – Структура проекта программного средства для администратора

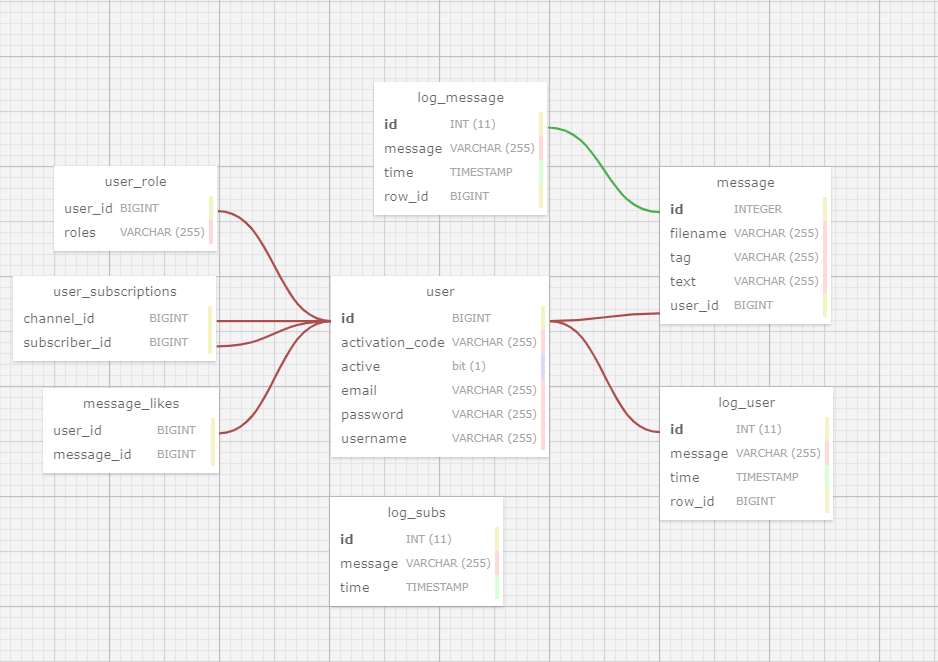
Описание приведенной структуры проекта представлено в таблице 3.1.1.

#### Таблица 3.1.1 – Описание структуры проекта

|  |  |
| --- | --- |
| Имя файла | Содержание |
| config | Пакет, в котором находятся классы конфигурации разных частей приложения. |
| controller | Пакет, в котором находятся контроллеры, которые взаимодействуют с представлением и моделью. Классы, принимающие запросы, обрабатываемые данные и возвращающие ответы в представление. |
| domain | Пакет, в котором находятся модели данных, которые соответствуют таблицам в базе данных. |
| domain.dto | Пакет, в котором находятся классы, которые были специально сформированы для корректного и простого отображения. |
| domain.util | Пакет, в котором находятся классы, которые облегчают и совершенствуют работу с классами domain. |
| repositories | Пакет, в котором находятся классы, с помощью которых происходит работа с базой данных. |
| repositories.dto | Пакет, в котором находятся интерфейсы, которые реализуют классы пакета domain |
| service | Пакет, в котором находятся сервисные классы. |
| resources | Директорий ресурсов. |
| db.migration | Директорий, содержащий скрипты миграций базы данных, а также скрипты создания таблиц, триггеров, процедур, функций, индексов. |
| static | Содержит статические элементы страницы, а также код CSS и JS. |
| templates | Содержит шаблоны страниц. |
| templates.parts | Содержит встраиваемые части для страниц. |
| application.properties | Файл конфигурации Spring. |
| hibernate-cfg.xml | Файл конфигурации Hibernate. |
| pom.xml | Файл, который содержит информацию о деталях проекта, и конфигурации используемых для создания проекта на Maven. |

Таким образом, приведённая таблица содержит название каждого пакета, а также имеет краткое его назнчение, что помогает понять стркутуру проектируемого программного средства.

## Структура модели базы данных



### Рисунок 3.1.2 – Схема модели базы данных

## Описание разрабатываемой функциональности программного средства

На рисунке 3.2.1 представлена UML-диаграмма вариантов использования.

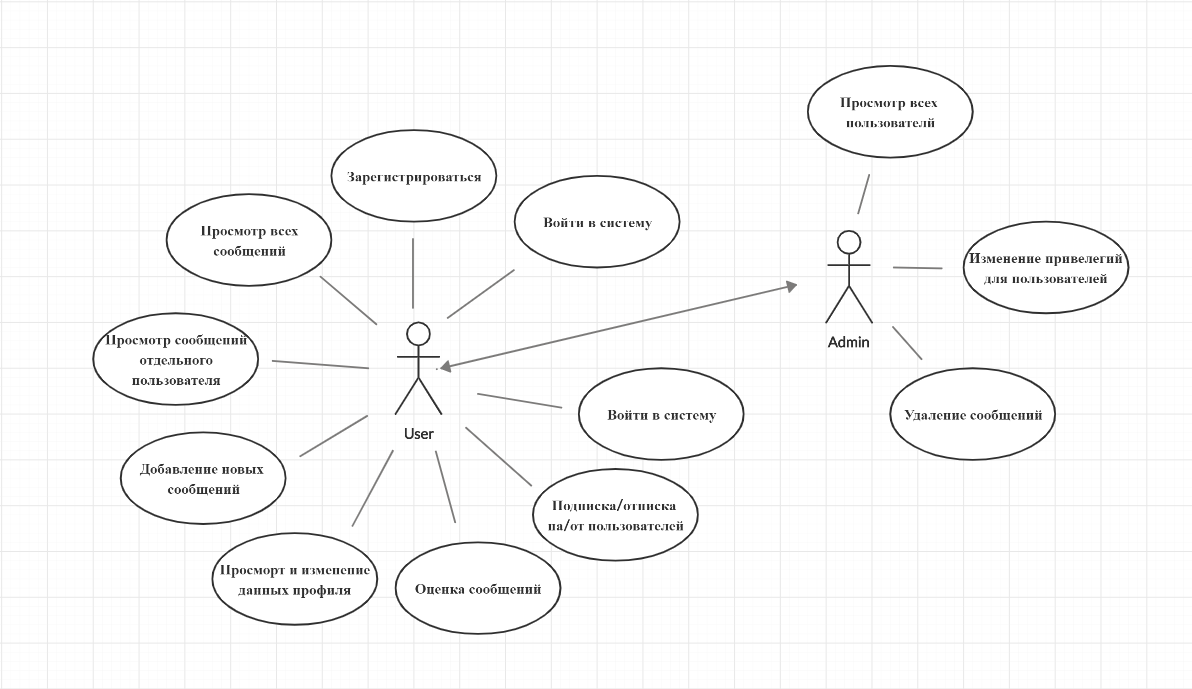


Рисунок 3.2.1 – UML-диаграмма вариантов использования

Данная диаграмма отражает функциональность программного средства.

# Разработка необходимых объектов

Логическая структура базы данных определяет структуру таблиц, взаимоотношения между ними, список пользователей, хранимые процедуры, правила, умолчания и другие объекты базы данных [2].

Логически, данные в MySQL организованы в виде объектов. База данных данного курсового проекта содержит следующие объекты: таблицы, хранимые процедуры, пользователи, функции, триггеры.

**4.1 Таблицы**

Все данные в SQL содержатся в объектах, называемых таблицами. Таблицы представляют собой совокупность каких-либо сведений об объектах, явлениях, процессах реального мира. Никакие другие объекты не хранят данные, но они могут обращаться к данным в таблице.

Таблицы в MySQL имеют такую же структуру, что и таблицы всех других СУБД и содержат:

* cтроки; каждая строка (или запись) представляет собой совокупность атрибутов (свойств) конкретного экземпляра объекта;
* столбцы; каждый столбец (поле) представляет собой атрибут или совокупность атрибутов. Поле строки является минимальным элементом таблицы. Каждый столбец в таблице имеет определенное имя, тип данных и размер.

База данных данного курсового проекта содержит 8 таблиц, 7 связей, которые описаны в главе 2 и 3. Так же в базе данных создались таблицы, которые нужны для работы некоторых технологий, используемых в проекте.

Листинг SQL-кода для создания таблиц находится в приложении А.

**4.2 Пользователи**

Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. При проектировании базы данных понадобилось как минимум 2 пользователя (ADMIN и USER). У каждого пользователя есть свои привилегии, предназначенные для выполнения соответствующих ему требований. Листинг SQL-кода создания пользователей и назначения им привилегий приведен в приложении А.

Пользователь USER наделен правами, имитирующего пользователя, который может регистрироваться, просматривать сообщения, добавлять сообщения, отслеживать сообщения конкретных пользователей, оценивать сообщения, просматривать свой личный профиль и изменять, в случае необходимости.

Пользователь ADMIN наделён правами как пользователя, так и правами, имитирующими администратора, который может просматривать список пользователей, а также может назначать, снимать им роли.

**4.3 Хранимые процедуры**

Хранимая процедура – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Таким образом, вместо того, чтобы хранить часто используемый запрос, клиенты могут ссылаться на соответствующую хранимую процедуру. Это обеспечивает лучшую производительность, поскольку данный запрос должен анализироваться только однажды и уменьшается трафик между сервером и клиентом.

Их использование в курсовом проекте предназначено для ограничения доступа пользователей к таблицам базы данных, оставив пользователям только разрешения на выполнение некоторых процедур, обеспечивающих косвенный и строго регламентированный доступ к данным.

Пользователю данного приложения доступны следующие процедуры:

* AddMessage (in filename varchar(255), in tag varchar(255), in text varchar(255), in user\_id bigint) – добавить сообщение;
* ActivateUser (in id bigint) – активировать пользователя (выслать код активации на email);
* AddUser (in activation\_code varchar(255), in active bit, in email varchar(255), in password varchar(255), in username varchar(255)) – зарегистрировать пользователя;
* DislikeMessage(in u\_id bigint, in m\_id bigint) ­– снятие оценки с сообщения от конкретного пользователя;
* GetAllMessageBySubsAndTagDto (in cur\_id bigint, in tag\_in varchar(255)) – получить сообщения в определённом формате по тегам от отслеживаемых пользователя;
* GetAllMessageBySubsDto (in cur\_id bigint) – получить сообщения отслеживаемых пользователей в определённом формате;
* GetAllMessageByTagDto (in cur\_id bigint, tag\_in varchar(255)) – получить сообщения по тегу в определённом формате;
* GetAllMessageByUserDto (in user\_id bigint, in cur\_id bigint) – получить сообщения конкретного пользователя в определённом формате;
* GetAllMessageDto ()– получить все сообщения в определённом формате;
* GetAllSimilarUsersByUsername (in username varchar(255)) – получить список всех пользователей по логину;
* GetCountOfMessage (in id bigint) – получить количество сообщений конкретного пользователя;
* GetCountSubscribers (in id bigint) – получить количество подписчиков пользователя;
* GetCountSubscriptions (in id bigint) – получить количество подписок пользователя;
* GetSubscribersByUserId (in id bigint) – получить пользователей, которые отслеживают сообщения конкретного пользователя;
* GetSubscriptionsByUserId (in id bigint) – получить пользователей, которых отслеживает конкретный пользователь;
* GetUserByActivationCode (in code varchar(255)) – получить пользователя по коду активации;
* GetUserById (in id bigint) – получить пользователя по id;
* GetUserByUsername (in username varchar(255)) – получить пользователя по логину;
* LikeMessage (in user\_id bigint, in message\_id bigint) – установить оценку на сообщение от конкретного пользователя;
* Subscribe (in id bigint, in sub\_id bigint) – начать отслеживать сообщение пользователя;
* Unsubscribe (in id bigint, in sub\_id bigint) – перестать отслеживать сообщения конкретного пользователя;
* UpdateMessage (in id bigint, in filename varchar(255), in tag varchar(255), in text varchar(255), in user\_id bigint) – обновить данные сообщения;
* UpdateUser (in id bigint, in email varchar(255), in password varchar(255), in activationCode varchar(255)) – обновить данные пользователя;

Функции:

* meLiked (user\_id bigint, message\_id bigint) – проверяет наличие оценки на конкретном сообщении от конкретного пользователя;

Администратору данного приложения доступны следующие процедуры:

* AddMessage (in filename varchar(255), in tag varchar(255), in text varchar(255), in user\_id bigint) – добавить сообщение;
* DeleteMessage (in id bigint) – удалить сообщение по id;
* ActivateUser (in id bigint) – активировать пользователя (выслать код активации на email);
* AddUser (in activation\_code varchar(255), in active bit, in email varchar(255), in password varchar(255), in username varchar(255)) – зарегистрировать пользователя;
* DislikeMessage(in u\_id bigint, in m\_id bigint) ­– снятие оценки с сообщения от конкретного пользователя;
* GetAllMessageBySubsAndTagDto (in cur\_id bigint, in tag\_in varchar(255)) – получить сообщения в определённом формате по тегам от отслеживаемых пользователя;
* GetAllMessageBySubsDto (in cur\_id bigint) – получить сообщения отслеживаемых пользователей в определённом формате;
* GetAllMessageByTagDto (in cur\_id bigint, tag\_in varchar(255)) – получить сообщения по тегу в определённом формате;
* GetAllMessageByUserDto (in user\_id bigint, in cur\_id bigint) – получить сообщения конкретного пользователя в определённом формате;
* GetAllMessageDto ()– получить все сообщения в определённом формате;
* GetAllSimilarUsersByUsername (in username varchar(255)) – получить список всех пользователей по логину;
* GetCountOfMessage (in id bigint) – получить количество сообщений конкретного пользователя;
* GetCountSubscribers (in id bigint) – получить количество подписчиков пользователя;
* GetCountSubscriptions (in id bigint) – получить количество подписок пользователя;
* GetSubscribersByUserId (in id bigint) – получить пользователей, которые отслеживают сообщения конкретного пользователя;
* GetSubscriptionsByUserId (in id bigint) – получить пользователей, которых отслеживает конкретный пользователь;
* GetUserByActivationCode (in code varchar(255)) – получить пользователя по коду активации;
* GetAllUsers () – получить список всех пользователей;
* GetUserById (in id bigint) – получить пользователя по id;
* GetUserByUsername (in username varchar(255)) – получить пользователя по логину;
* LikeMessage (in user\_id bigint, in message\_id bigint) – установить оценку на сообщение от конкретного пользователя;
* Subscribe (in id bigint, in sub\_id bigint) – начать отслеживать сообщение пользователя;
* Unsubscribe (in id bigint, in sub\_id bigint) – перестать отслеживать сообщения конкретного пользователя;
* UpdateMessage (in id bigint, in filename varchar(255), in tag varchar(255), in text varchar(255), in user\_id bigint) – обновить данные сообщения;
* UpdateUser (in id bigint, in email varchar(255), in password varchar(255), in activationCode varchar(255)) – обновить данные пользователя;

Функции:

* meLiked (user\_id bigint, message\_id bigint) – проверяет наличие оценки на конкретном сообщении от конкретного пользователя;
* GetLastInsertedMessageXML () – получить последнее вставленное сообщение в формате XML;
* GetOneMessagesWithTheMostLikes () – получить сообщение с наибольшем количеством оценок;

Листинг SQL-кода всех процедур находится в приложении А.

1. **Описание процедур импорта и экспорта данных**

В данной курсовой работе реализованы функции экспорта в XML выбранных данных из таблицы MESSAGE и экспорта данных из XML формата в таблицу. При данных операциях работает с строкой, которая представляет данные в XML формате, а разбором и генерацией XML занимается MySQL.

XML (Extensible Markup Language) – расширяемый язык разметки. XML-формат часто используется для обмена данными между компонентами информационных систем. При работе с базами данных важными являются две задачи: преобразование табличных данных в XML-структуры и преобразование XML-структур в строки реляционной таблицы.

Код функций импорта и экспорта представлен в Приложении А.

**5.1 Описание процедуры импорта данных**

Существует несколько вариантов процедуры импорта данных xml MySQL:

* импорт данных xml с помощью программы MySQL Workbench;
* работа с xml с помощью функций MySQL.

ExtractValue – функция MySQL, преобразующая данные непосредственно из строки/файла на XML в данные, которые могут быть использованы для вставки в таблицы базы данных.

Вызов функции ExtractValue с параметрами (@xml, '//message/@tag') «достанет» из переменной XML данные атрибута tag, которые находятся в теге MESSAGE.

Вызов функции ExtractValue с параметрами (@xml, '//message/text') «достанет» данные, которые находятся во внутреннем теге TEXT тега MESSAGE.

**5.2** **Описание процедуры экспорта данных**

Для преобразования результата SELECT-запроса в формат XML в операторе SELECT применяется секция CONCAT, с помощью которой мы сами формируем нужную структуру XML. При этом может формироваться нужная структура для разных нужд.

# Тестирование производительности

Оптимизация запросов — процесс изменения запроса и/или структуры БД с целью уменьшения использования вычислительных ресурсов при выполнении запроса. Один и тот же результат может быть получен СУБД различными способами (планами выполнения запросов), которые могут существенно отличаться как по затратам ресурсов, так и по времени выполнения.

В MySQL Server оптимизация запросов в основном заключается в построение индексов над таблицами, и изменением плана запроса. Индекс – это объект базы данных, предназначенный для ускорения запросов к данным в таблице базы данных. Индекс состоит из набора страниц, узлов индекса, которые организованы в виде древовидной структуры — сбалансированного дерева. Эта структура является иерархической по своей природе и начинается с корневого узла на вершине иерархии и конечных узлов, листьев.

Рекомендации при планировании стратегии индексирования:

* для таблиц, которые часто обновляются используйте как можно меньше индексов;
* если таблица содержит большое количество данных, но их изменения незначительны, тогда используйте столько индексов, сколько необходимо для улучшения производительности ваших запросов;
* уникальность значений в столбце влияет на производительность индекса, в общем случае, чем больше у вас дубликатов в столбце, тем хуже работает индекс;
* для кластеризованных индексов старайтесь использовать настолько короткие поля насколько это возможно. Наилучшим образом будет применение кластеризованного индекса на столбцах с уникальными значениями и не позволяющими использовать NULL. Вот почему первичный ключ часто используется как кластеризованный индекс.

## Описание типов индексов

Есть два основных типа индексов: кластеризованные и некластеризованные индексы [10].

При создании кластеризованного индекса данные индексируемой таблицы располагаются в физическом порядке, соответствующем индексу, и становятся частью кластеризованного индекса. Важной характеристикой кластеризованного индекса является то, что все значения отсортированы в определенном порядке либо возрастания, либо убывания. Таким образом, таблица или представление может иметь только один кластеризованный индекс.

Некластеризованный индекс – это отдельный объект, имеющий указатели на строки таблицы. Максимальное количество некластерированных индексов для одной таблицы не должно превышать 1000.

При создании индекса указывается один или несколько столбцов таблицы, по значениям которых будет построен и поддерживаться индекс. Индекс представляет собой структуру памяти, организованную в виде сбалансированного дерева.

**6.2 Пример использования индексов в базе данных**

В базе данных данного курсового проекта в каждой таблице находится поле с уникальным значением поля типа int, соответственно каждая таблица содержит кластеризованный индекс. База данных содержит множество процедур с выборкой содержащий оператор WHERE сравнивающий строки. Наиболее частое сравнение — это логин и пароль для проверки на корректность запроса от пользователя. Даже при незначительном заполнении базы данных (около 300000 записей в таблице) время выборки начинает занимать существенное время.

Ниже на рисунке 6.2.1 представлена карта запроса при выборке в таблице пользователей.

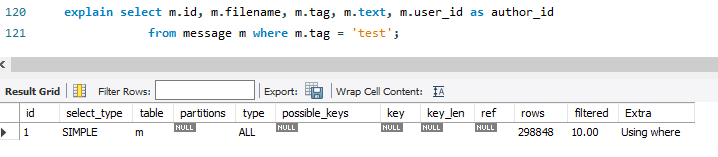


Рисунок 6.2.1 – Карта запроса без оптимизации

Как можно увидеть, количество строк, которые нужно было обработать запросу, чтобы дойти до нужной строки, очень велико. Так как в секции WHERE используется поле TAG, создадим на его основе некластеризованный индекс. При создании индекса указывается один или несколько столбцов таблицы, по значениям которых будет построен индекс. Листинг создания индекса находится в приложении A.

Новый результат выполнения запроса можно увидеть на рисунке 6.2.2.

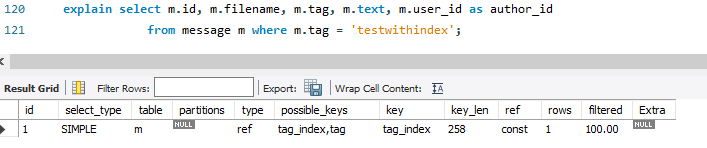


Рисунок 6.2.2 – Карта запроса с некластеризованным индексом

Как можно заметить, количество строк, которые пришлось пройти значительно меньше, так как значение, по которому мы ищем, отлично от всех, и индекс представился в виде отсортированного набора значений. Построение индексов для таблиц весьма эффективна для операций выборки с условием.

# Описание технологий

## Описание и реализация технологии шифрования данных

Актуальная проблема из мира информационной безопасности – обеспечить сохранность данных. Есть ситуации, в которых даже при наличии серьезной защиты системы, сохранность данных оказывается под большим вопросом.

Шифрование — использование технологии шифрования для преобразования информации, хранящейся в базе данных (БД), в шифротекст, что делает ее прочтение невозможным для лиц, не обладающих ключами шифрования.

Естественно, логин и пароль могут храниться в любом виде в базе данных, однако целесообразно шифровать пароли, чтобы злоумышленники не смогли получить доступ к вашей информации. То есть, при регистрации нового пользователя, в БД заносится предварительно зашифрованный пароль пользователя, так сказать в закрытом виде.[10]

Шифрование данных происходит с использованием парольной фразы с использованием алгоритма SHA-512. Из парольной фразы создается симметричный ключ. Исходное сообщение после дополнения разбивается на блоки, каждый блок — на 16 слов. Алгоритм пропускает каждый блок сообщения через цикл с 64 или 80 итерациями (раундами). На каждой итерации 2 слова преобразуются, функцию преобразования задают остальные слова. Результаты обработки каждого блока складываются, [сумма](https://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%BC%D0%BC%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) является значением хеш-функции. Тем не менее, инициализация внутреннего состояния производится результатом обработки предыдущего блока. Поэтому независимо обрабатывать блоки и складывать результаты нельзя.

Реализация технологии шифрования представлена ниже.

update user set password = SHA2(password, 512) where id = 1;

Листинг 7.1.1 – Шифрование пароля

3c9909afec25354d551dae21590bb26e38d53f2173b8d3dc3eee4c047e7ab1c1eb8b85103e3be7ba613b31bb5c9c36214dc9f14a42fd7a2fdb84856bca5c44c2

Листинг 7.1.2 – Полученный новый пароль

# Руководство пользователя

Пользовательское приложение предоставляет собой веб-приложение для взаимодействия с базой данных для администратора и пользователей.

После запуска приложения, мы переходим в браузер и заходим на адрес: http://localhost:8090 открывается страница с кнопками авторизации или регистрации, представленное на рисунке 8.1.

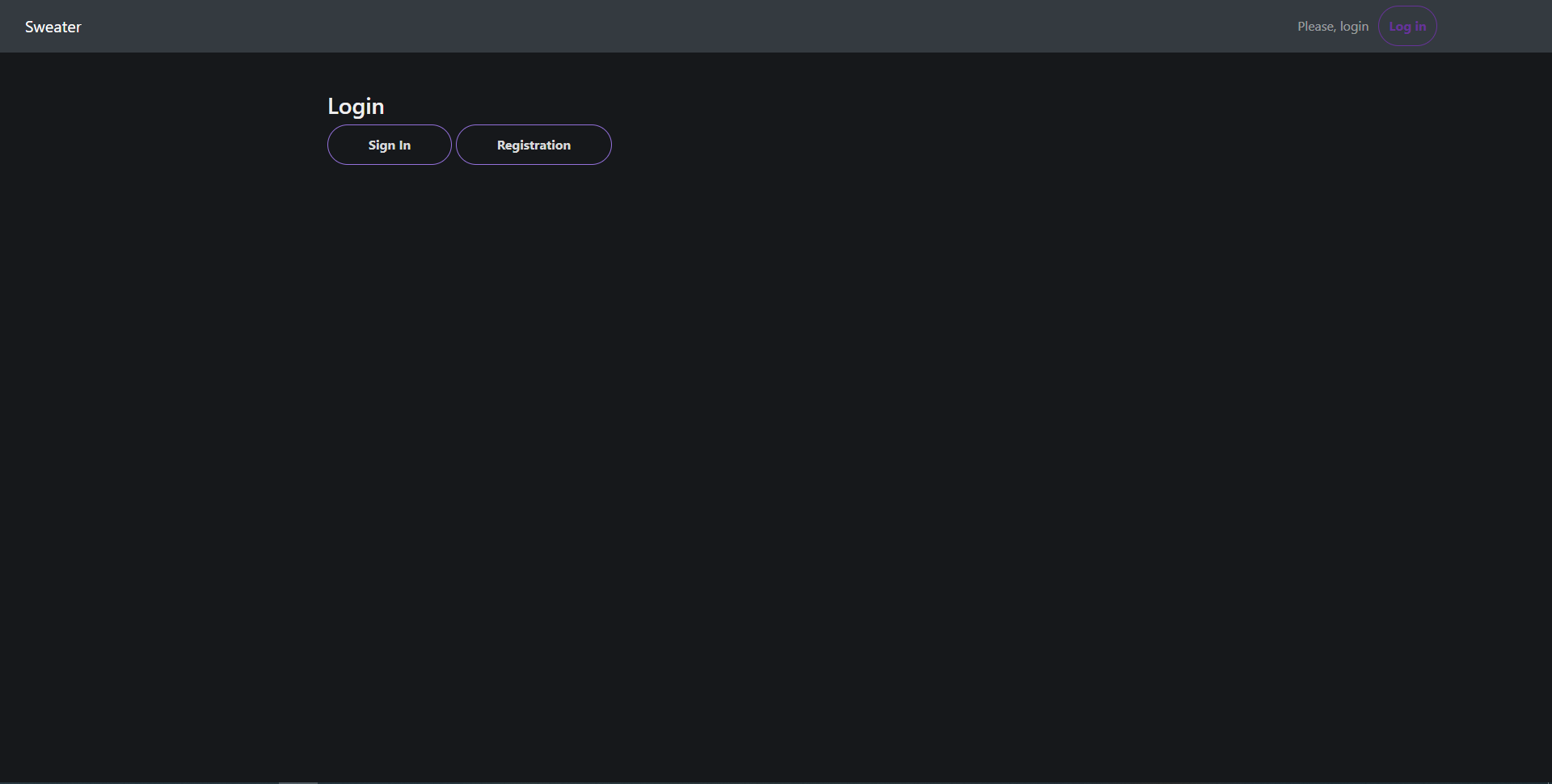


Рисунок 8.1 – Окно авторизации

Окно авторизации требует выбора совершаемого действия. Выбор предоставляется 2-мя кнопками: «SignIn», «Registration». «SignIn» требует ввода логина и пароля для начала работы с приложением. Однако, если у вас нет учетной записи, то ее можно создать, нажав Registration, после чего вам будет показана форма регистрации (Рисунок 8.2), где необходимо заполнить соответствующие поля. В случае неверного ввода данных выведется соответствующее сообщение.

После успешной регистрации необходимо вернуться в окно авторизации и ввести ваши данные, указанные при регистрации.

После ввода логина и пароля на форме авторизации будет осуществляться проверка правильности введенных данных и поиск пользователя с такими данными в базе данных. В случае неверно введенных данных выведется сообщение об ошибке. В ином случае окно авторизации закроется и отроется главное окно, где уже можно непосредственно взаимодействовать с отдельными страницами.

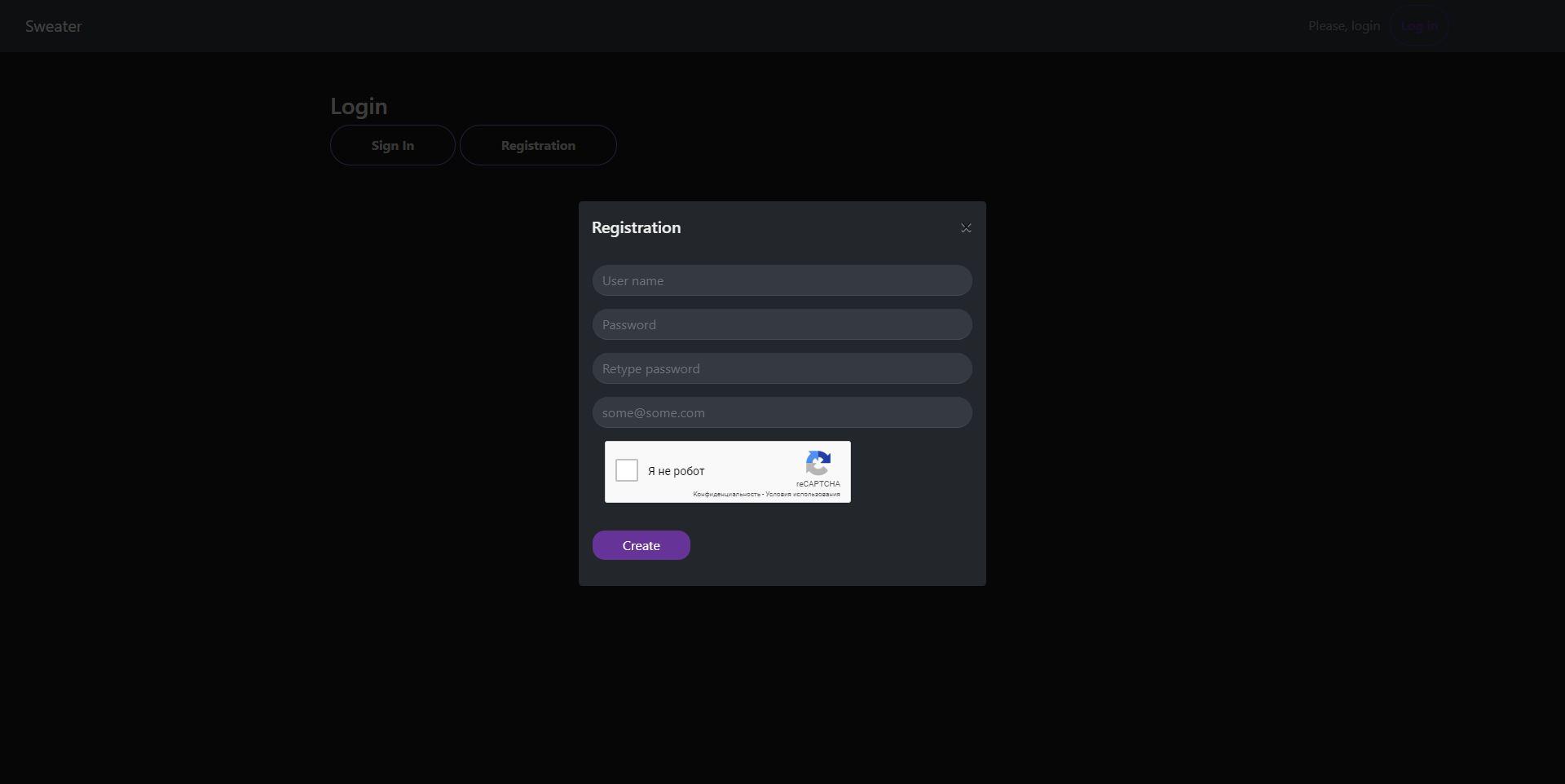


Рисунок 8.2 – Окно регистрации

После успешной регистрации, вам нужно залогиниться в системе, нажав на кнопку «SignIn». Вам будет показано окно, с формой авторизации, представленное на рисунке 8.3.

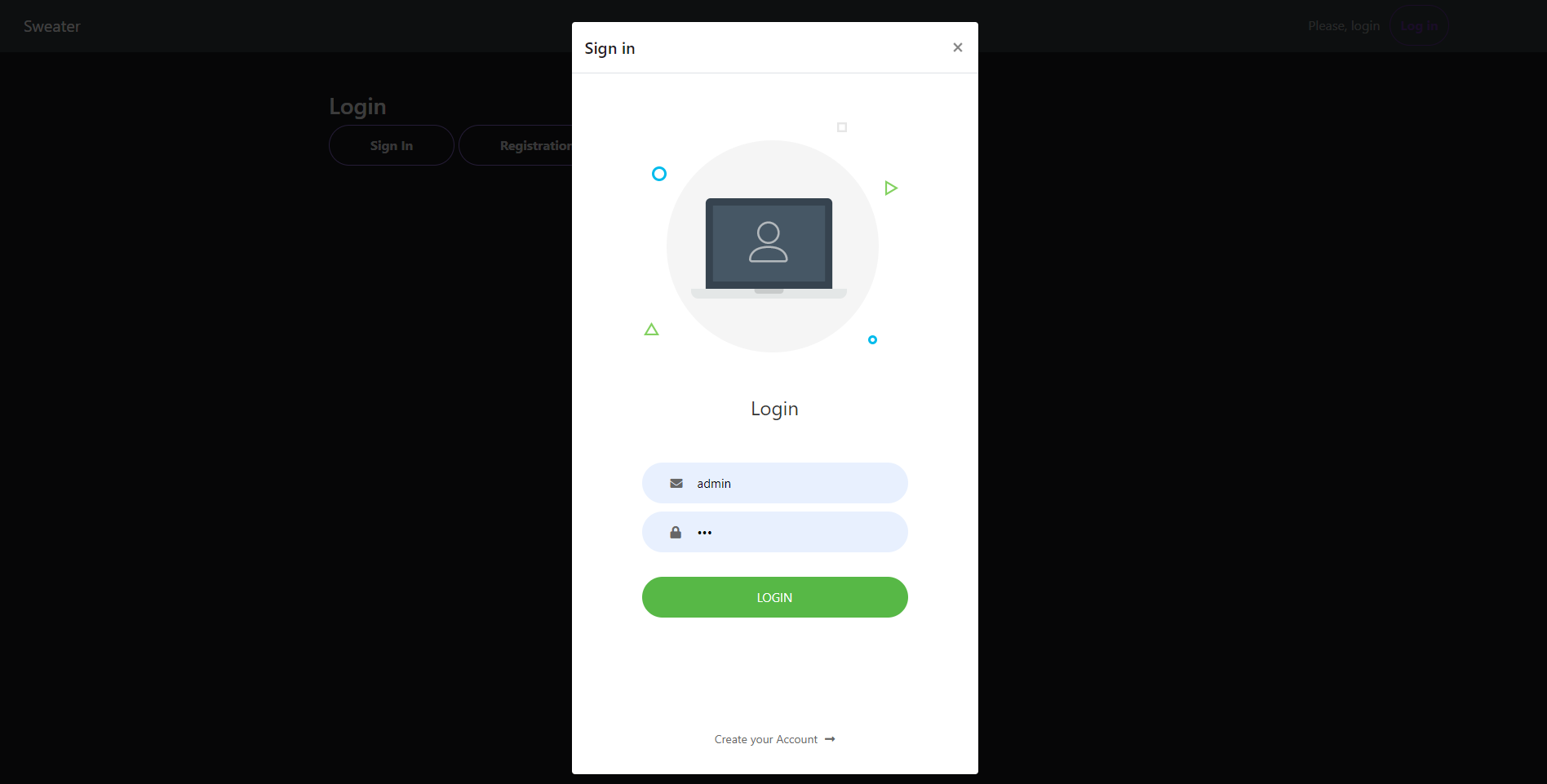


Рисунок 8.3 – Окно авторизации

После того, как вы вошли в систему, вас перекинет на страницу приветствия, изображённую на рисунке 8.4

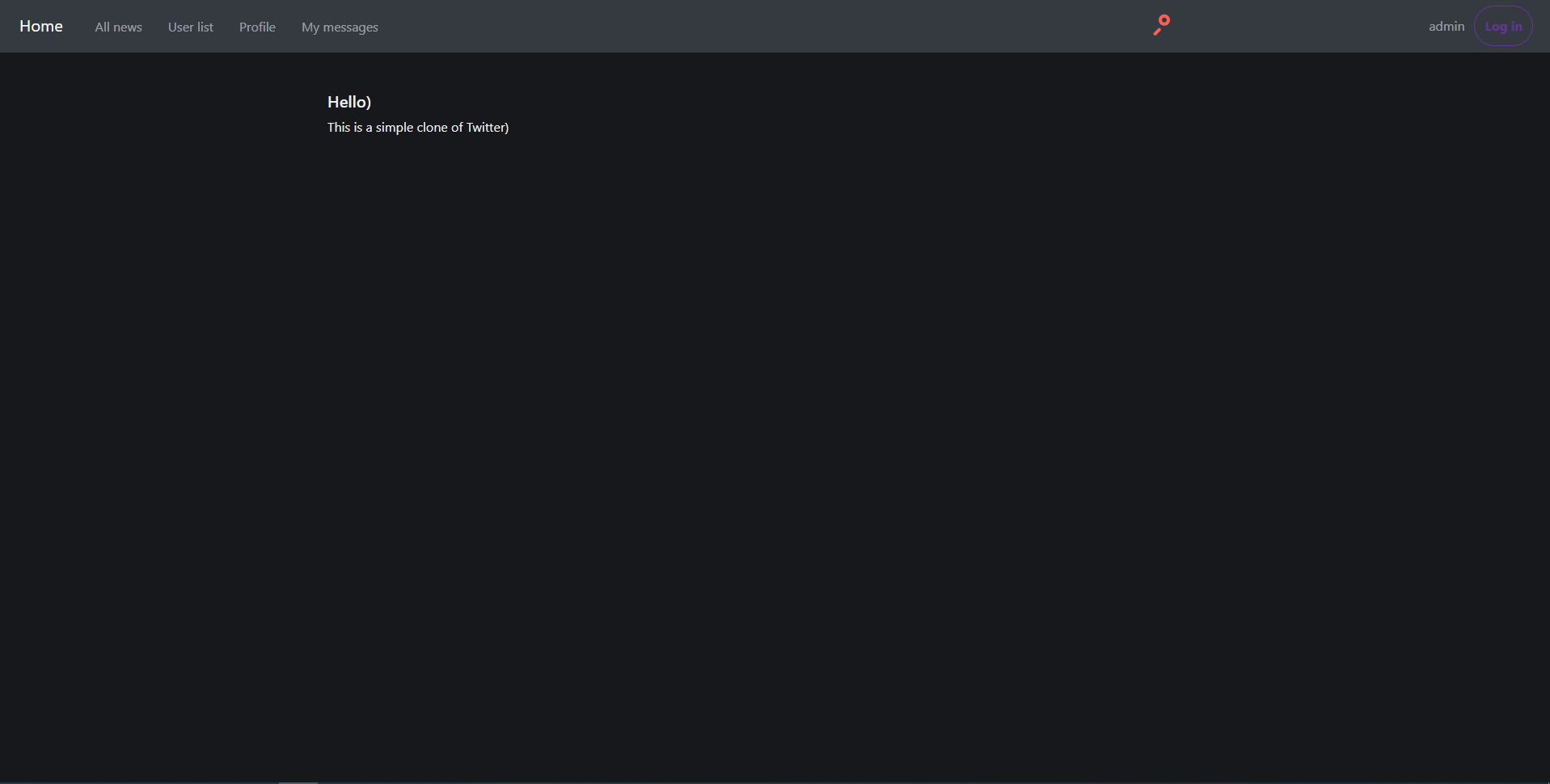


Рисунок 8.4 – Главная страница

Рассмотрим подробнее каждую из страниц, с которой вы можете c столкнуться в ходе работы с приложением.

Вкладка «Home», на которой пользователь может увидеть сообщения тех пользователей, на которых он подписан 8.5. Так же вы можете оценивать то либо иное сообщение, как показано на рисунке.

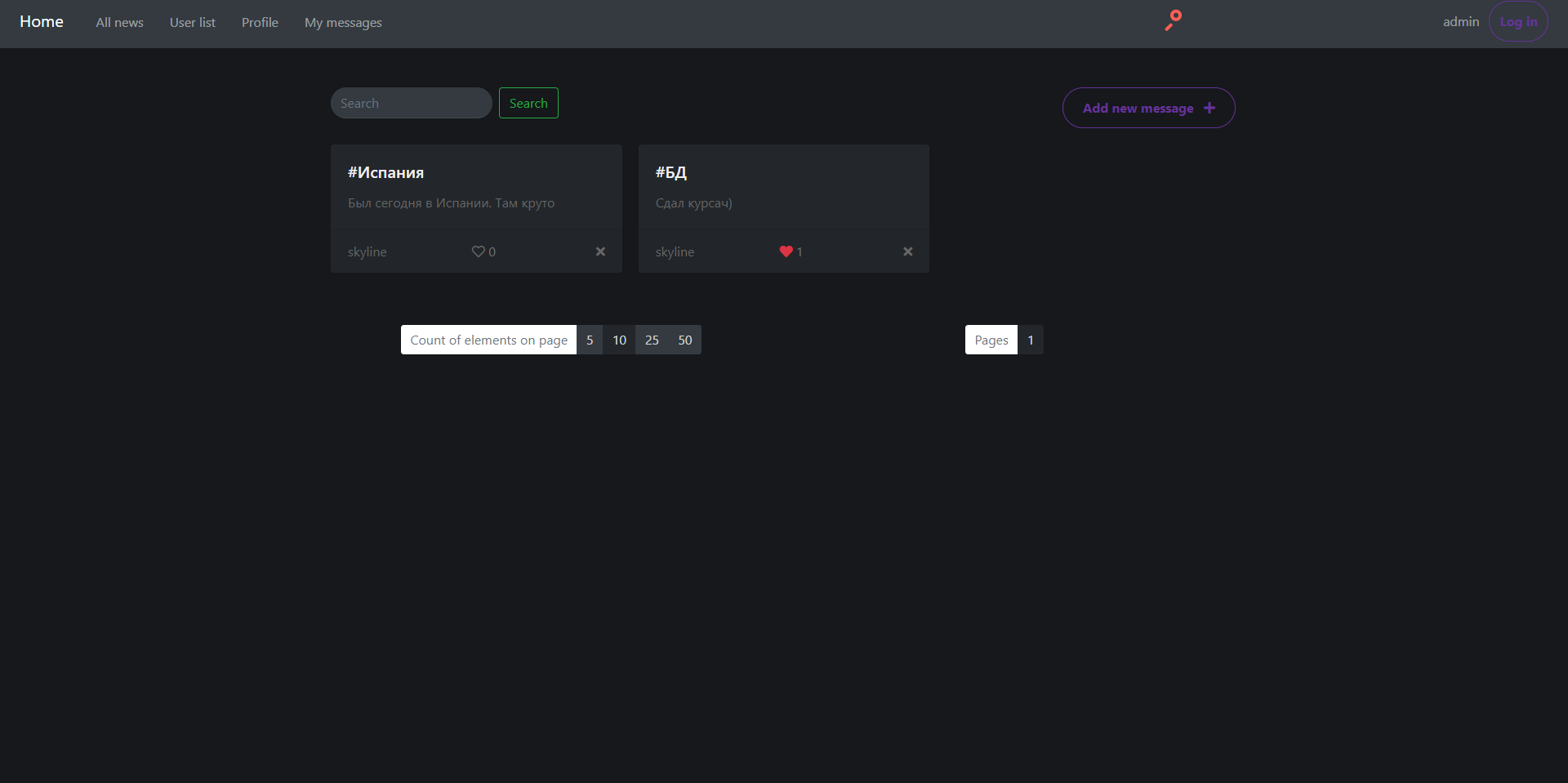


Рисунок 8.5 – Страница Home

Чтобы добавить своё сообщение, нужно нажать на кнопку Add new message. Вам будет показано окно, в котором вы сможете ввести информацию, для заполнения вашего сообщения, а также можете прикрепить картинку.

Окно представлено на рисунке 8.6 По нажатию кнопки «Send» ваше сообщение появится в общей ленте (вкладка All news).

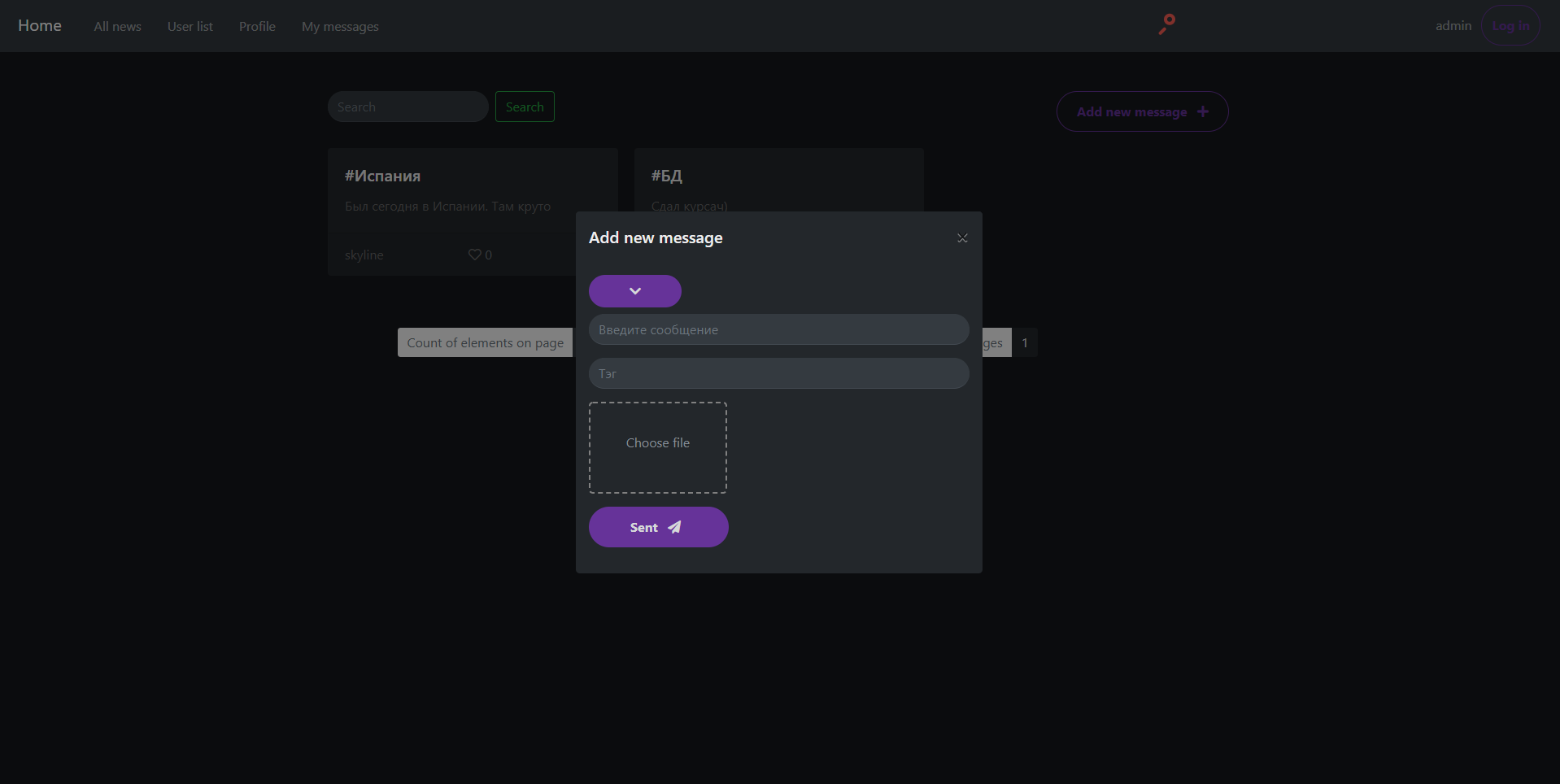


Рисунок 8.6 – Окно добавления сообщения

Чтобы просмотреть сообщение всех пользователей, (чтобы найти для себя новых интересных личностей), вы можете перейти по вкладке «All news». Так же на этих странице есть поиск по тегу (Рисунок 8.7).

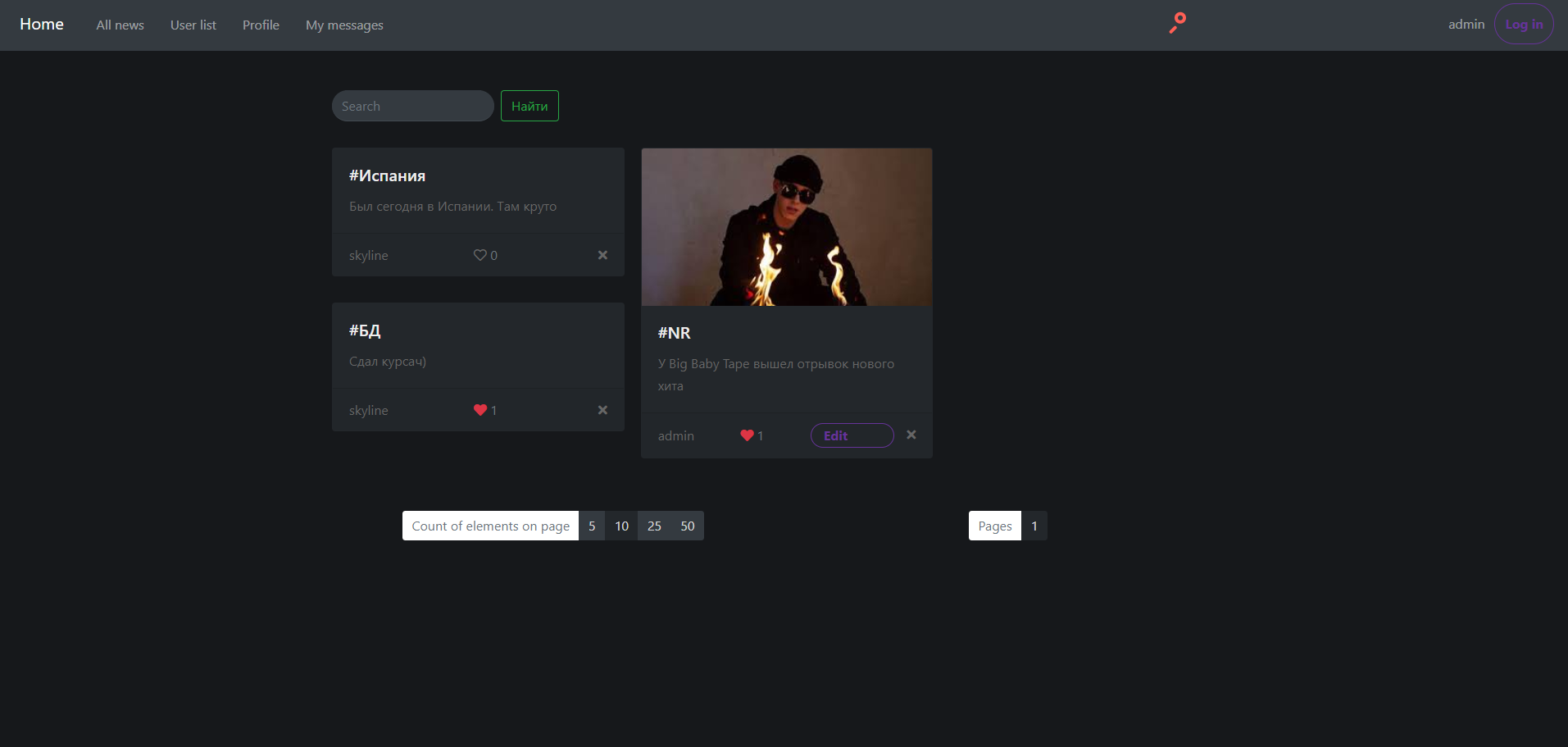


Рисунок 8.7 – Страница всех сообщений

Перейдя по вкладке Profile, вы перейдёте на страницу с данными вашего аккаунта, которые вы можете изменить в случае необходимости (Рисунок 8.8). Такие данные как USERNAME, задаются один раз, при регистрации, и изменение этого поля не предусмотрено. Изменение значение этого поля рассматривается, и возможно, будет добавлено в следующих обновлениях.

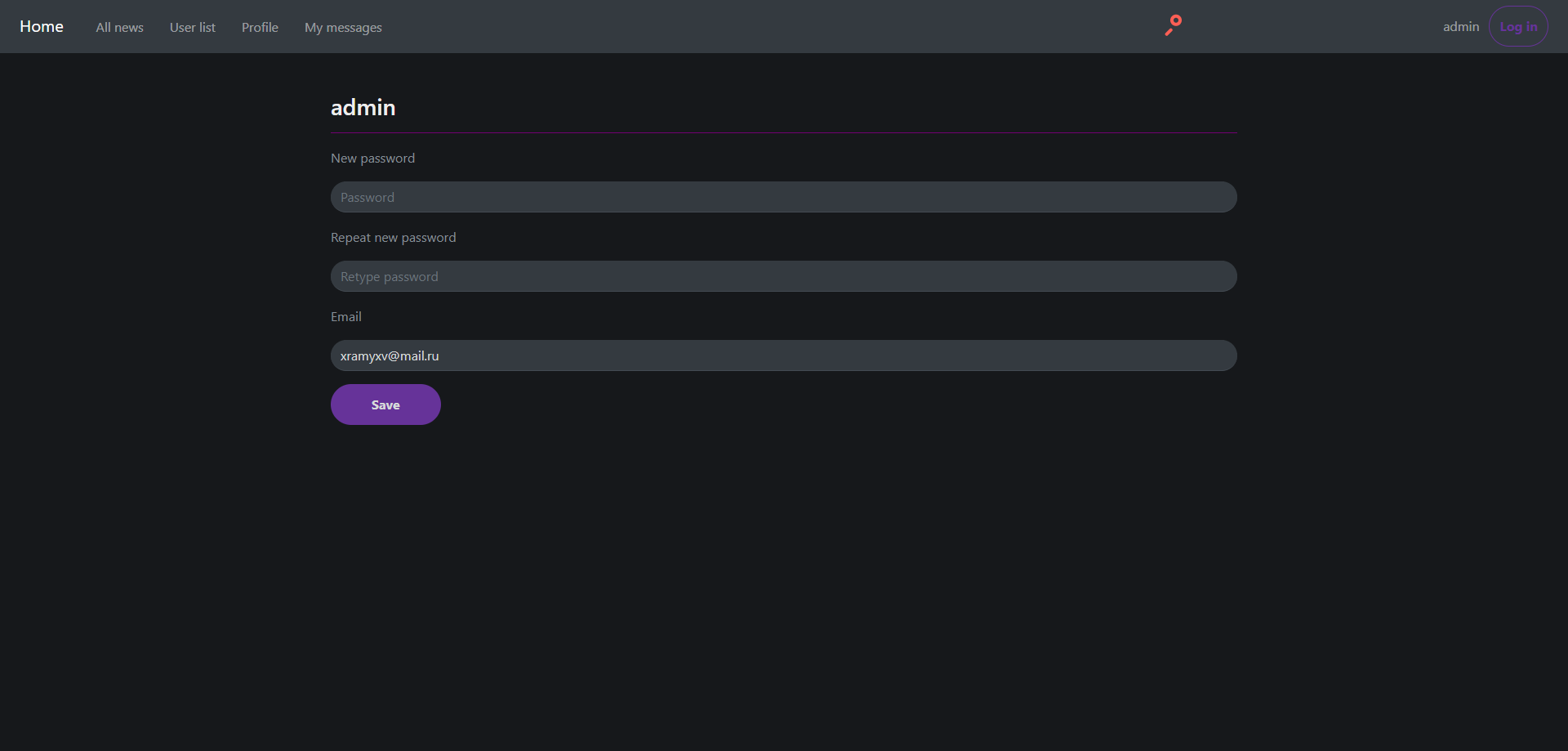
****

Рисунок 8.9 – Страница профиля

Чтобы просмотреть список своих сообщений, а также изменить их содержимое, вы можете перейти по вкладке «My messages» (Рисунок 8.10). Здесь, при нажатии на кнопку «edit» в блоке сообщения, у вас покажутся поля для изменения. После самого изменения, чтобы сохранить сообщение, вам нужно нажать на кнопку «Save». Также на этой странице можно увидеть количество своих подписчиков и подписок. Нажав на это число, вы перейдёте к их списку, с возможностью просмотра профиля каждого пользователя.

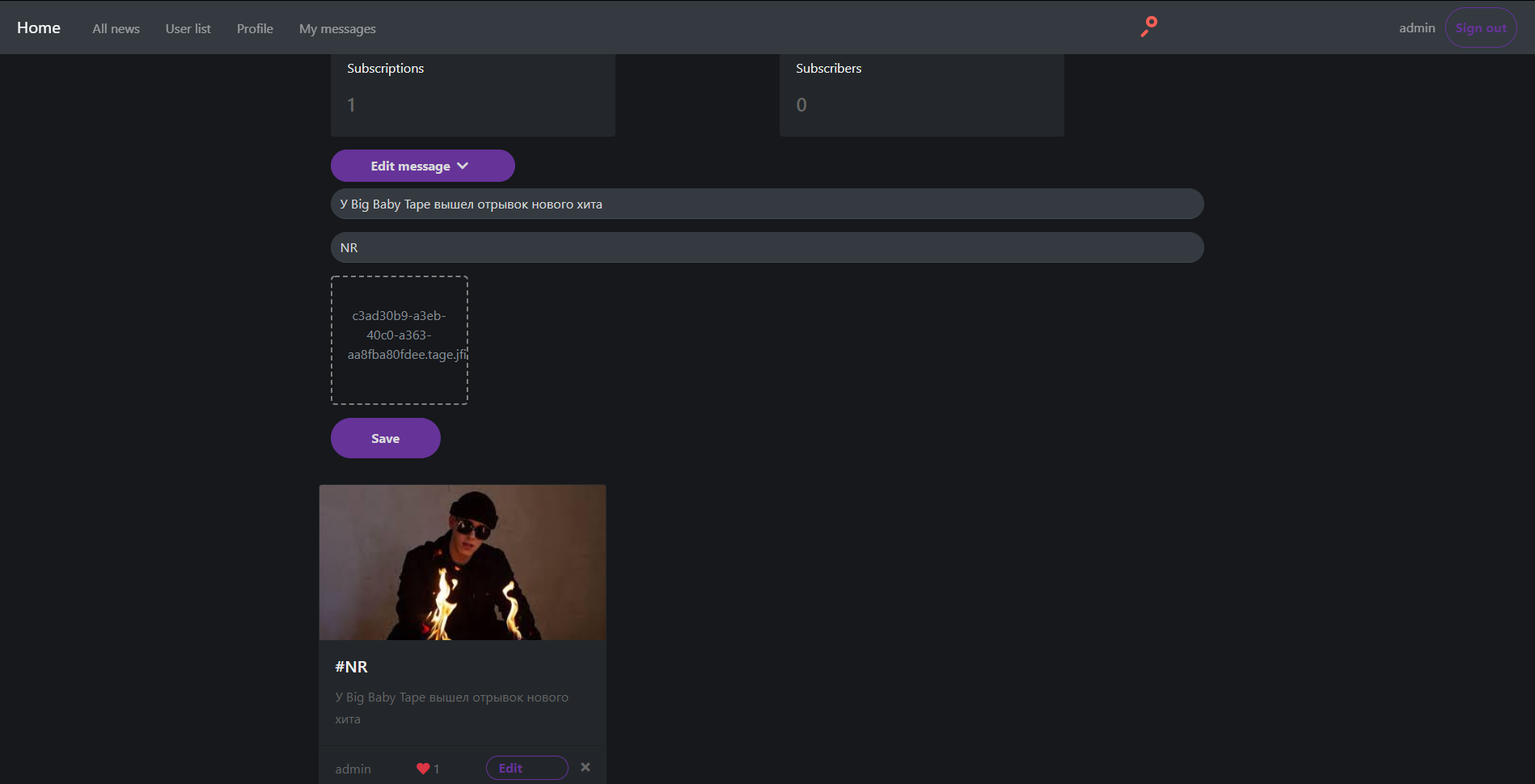


Рисунок 8.10 – Добавление новой задачи

Перейдём к возможностям администратора. У администратора во вкладках есть дополнительная вкладка «User list». Перейдя по этой вкладке, он увидит список всех пользователей, а также получит возможность назначения/удаления привилегий тем или иным пользователям. Эта страница показана на рисунке 8.11.

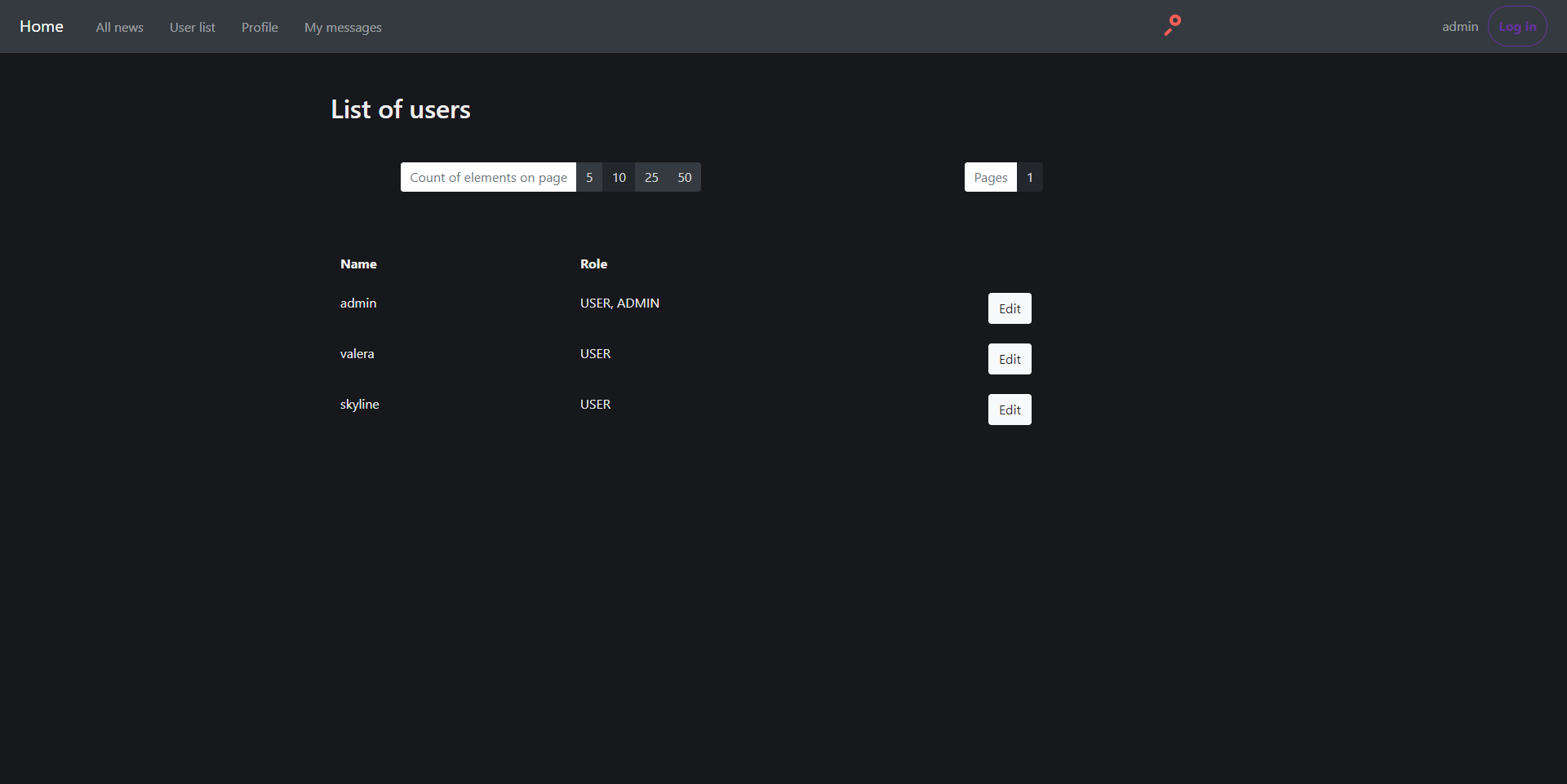


Рисунок 8.11 – Страница администратора

Так же у администратора есть возможность удаления сообщений, если он посчитает их содержание окажется неприемлемо. Это организовано при помощи «крестика». Это было показано на рисунке 8.7.

# Заключение

В данном курсовом проекте была разработана база данных «Sweater», хранимые процедуры, индексы, триггеры, функции, а также настроены пользователь и администратор и их права на выполнение тех или иных процедур.

Были разработаны процедуры для реализации следующих целей:

* Сохранение пользовательской информацию в базе данных;
* Получение информации из базы данных;
* Регистрации и авторизации пользователей;
* Удаления при необходимости информации из базы данных;
* Изменение информации, в базе данных;
* Записи информации в xml-формате;
* Считывания информации в xml-формате и её разбор;

Доступ к данным, а также манипулирование ими осуществляется посредством процедур.

В результате выполнения курсового проекта было получено функционирующее десктопное приложение «Sweater».

Программное средство было успешно протестировано. К программному средству прилагается руководство пользователя.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

В программном средстве были выполнены все пункты из указанного списка основного функционала приложения, включающие в себя роли обычного пользователя и администратора, возможность добавления, удаления, редактирования сообщений, просмотра списка всех пользователей.

По итогу работы продукт был протестирован и отлажен, проанализированы его возможности и, внесены дополнения и улучшения с возможностью дальнейшего расширения проекта.

# 10 Список используемых источников

1. Понятие и функции системы управления базами данных [Электронный ресурс]. — Режим доступа: <https://vuzlit.ru/998046/ponyatie_funktsii_sistemy_upravleniya_bazami_dannyh>. Дата доступа: 13.12.2019.
2. Официальный сайт с документацией по Spring Framework [Электронный ресурс]. https://www.spring.io /. Дата доступа: 14.12.2019.
3. Официальная документация по базе данных MySQL [Электронный ресурс] <https://dev.mysql.com/doc/>. Дата доступа:12.10.2019.
4. MySQL Indexes: Блог, с подробным пояснением по применению индексов базе данных MySQL [Электронный ресурс] <https://ruhighload.com/Индексы+в+mysql> Дата доступа:12.12.2019
5. Easy Java. Java в примерах [Электронный ресурс] <https://easyjava.ru/data/hibernate/> Дата доступа:10.12.2019
6. Подробный анализ хранимых процедур MySQL [Электронный ресурс] <https://ruseller.com/lessons.php?id=1189>. Дата доступа:10.12.2019
7. Основы и способы информационной безопасности в 2017 году [Электронный ресурс] – Режим доступа: <https://habr.com/ru/post/344294/> Дата доступа: 13.12.2019.
8. Примеры массового импорта и экспорта XML-документов [Электронный ресурс] / Режим доступа URL: [https://stackoverflow.com/questions/9672780/importing..](https://vk.com/away.php?to=https%3A%2F%2Fstackoverflow.com%2Fquestions%2F9672780%2Fimporting-xml-data-from-xml-file-to-sql-database&cc_key=)  – Дата доступа: 12.12.2019.
9. [METANIT.COM](https://vk.com/away.php?utf=1&to=http%3A%2F%2FMETANIT.COM) Сайт о программировании [Электронный ресурс] / Режим доступа – URL: [https://metanit.com](https://vk.com/away.php?utf=1&to=https%3A%2F%2Fmetanit.com) – Дата доступа: 12.12.2019

**Приложение А**

**Создание таблиц для базы данных SWEATER**

create database sweater;

use sweater;

create table hibernate\_sequence (next\_val bigint) engine=MyISAM;

insert into hibernate\_sequence values ( 1 );drop table ROLE;

create table user (

id bigint not null auto\_increment,

activation\_code varchar(255),

active bit not null,

email varchar(255),

password varchar(255) not null,

username varchar(255) not null unique,

primary key (id)

) engine=InnoDB;

create table message (

id bigint not null auto\_increment,

filename varchar(255),

tag varchar(255),

text varchar(2048),

user\_id bigint,

primary key (id)

) engine=InnoDB;

alter table message

add constraint message\_user\_fk

foreign key (user\_id) references user (id);

create table user\_role (

user\_id bigint not null,

roles varchar(255)

) engine=InnoDB;

alter table user\_role

add constraint user\_role\_user\_fk

foreign key (user\_id) references user (id);

create table `log\_user` (

`id` int( 11 ) not null auto\_increment primary key,

`message` varchar (255) not null,

`time` timestamp not null default CURRENT\_TIMESTAMP,

`row\_id` bigint not null,

foreign key (row\_id) references user (id)

) ENGINE = InnoDB;

create table `log\_message` (

`id` int( 11 ) not null auto\_increment primary key,

`message` varchar (255) not null,

`time` timestamp not null default CURRENT\_TIMESTAMP,

`row\_id` bigint not null

) ENGINE = InnoDB;

create table `log\_subs` (

`id` int( 11 ) not null auto\_increment primary key,

`message` varchar (255) not null,

`time` timestamp not null default CURRENT\_TIMESTAMP

) ENGINE = InnoDB;

alter table `sweater`.`message`

change column `tag` `tag` varchar(255) character set 'cp1251' null default null ;

alter table `sweater`.`message`

change column `text` `text` varchar(2048) character set 'cp1251' null default null ;

alter table `sweater`.`message`

change column `filename` `filename` varchar (255) character set 'cp1251' null default null ;

create table user\_subscriptions ( channel\_id bigint not null references user(id), subscriber\_id bigint not null references user(id), primary key(channel\_id, subscriber\_id));

create table message\_likes (

user\_id bigint not null,

message\_id bigint not null,

primary key(user\_id, message\_id)

);

alter table message\_likes

add constraint message\_likes\_message\_fk

foreign key (message\_id) references message (id);

**Скрипт создания админа и шифрование его пароля**

insert into user (id, username, email, password, active)

values(1, 'admin','xramyxv@mail.ru', '123', true);

insert into user\_role (user\_id, roles)

values (1,'USER'), (1,'ADMIN');

update user set password = SHA2(password, 512) where id = 1;

**Создание пользователей и назначение им привилегий**

CREATE USER 'user'@'localhost' IDENTIFIED BY 'password';

grant select, insert, update, delete on sweater.\* to 'user'@'localhost';

grant execute on procedure sweater.AddMessage to 'user'@'localhost';

grant execute on procedure sweater.ActivateUser to 'user'@'localhost';

grant execute on procedure sweater.AddUser to 'user'@'localhost';

grant execute on procedure sweater.DislikeMessage to 'user'@'localhost';

grant execute on procedure sweater.GetAllMessageBySubsAndTagDto to 'user'@'localhost';

grant execute on procedure sweater.GetAllMessageBySubsDto to 'user'@'localhost';

grant execute on procedure sweater.GetAllMessageByTagDto to 'user'@'localhost';

grant execute on procedure sweater.GetAllMessageByUserDto to 'user'@'localhost';

grant execute on procedure sweater.GetAllMessageDto to 'user'@'localhost';

grant execute on procedure sweater.GetAllSimilarUsersByUsername to 'user'@'localhost';

grant execute on procedure sweater.GetCountOfMessage to 'user'@'localhost';

grant execute on procedure sweater.GetCountSubscribers to 'user'@'localhost';

grant execute on procedure sweater.GetCountSubscriptions to 'user'@'localhost';

grant execute on procedure sweater.GetSubscribersByUserId to 'user'@'localhost';

grant execute on procedure sweater.GetSubscriptionsByUserId to 'user'@'localhost';

grant execute on procedure sweater.GetUserByActivationCode to 'user'@'localhost';

grant execute on procedure sweater.GetUserById to 'user'@'localhost';

grant execute on procedure sweater.GetUserByUsername to 'user'@'localhost';

grant execute on procedure sweater.LikeMessage to 'user'@'localhost';

grant execute on procedure sweater.Subscribe to 'user'@'localhost';

grant execute on procedure sweater.Unsubscribe to 'user'@'localhost';

grant execute on procedure sweater.UpdateMessage to 'user'@'localhost';

grant execute on procedure sweater.UpdateUser to 'user'@'localhost';

grant execute on function sweater.meLiked to 'user'@'localhost';

CREATE USER 'admin'@'localhost' IDENTIFIED BY 'password';

grant select, insert, update, delete on sweater.\* to 'admin'@'localhost';

grant execute on procedure sweater.AddMessage to 'admin'@'localhost';

grant execute on procedure sweater.ActivateUser to 'admin'@'localhost';

grant execute on procedure sweater.AddUser to 'admin'@'localhost';

grant execute on procedure sweater.DislikeMessage to 'admin'@'localhost';

grant execute on procedure sweater.GetAllMessageBySubsAndTagDto to 'admin'@'localhost';

grant execute on procedure sweater.GetAllMessageBySubsDto to 'admin'@'localhost';

grant execute on procedure sweater.GetAllMessageByTagDto to 'admin'@'localhost';

grant execute on procedure sweater.GetAllMessageByUserDto to 'admin'@'localhost';

grant execute on procedure sweater.GetAllMessageDto to 'admin'@'localhost';

grant execute on procedure sweater.GetAllSimilarUsersByUsername to 'admin'@'localhost';

grant execute on procedure sweater.GetAllUsers to 'admin'@'localhost';

grant execute on procedure sweater.GetCountOfMessage to 'admin'@'localhost';

grant execute on procedure sweater.GetCountSubscriptions to 'admin'@'localhost';

grant execute on procedure sweater.GetCountSubscribers to 'admin'@'localhost';

grant execute on procedure sweater.GetRolesByUserId to 'admin'@'localhost';

grant execute on procedure sweater.GetSubscribersByUserId to 'admin'@'localhost';

grant execute on procedure sweater.GetSubscriptionsByUserId to 'admin'@'localhost';

grant execute on procedure sweater.GetUserByActivationCode to 'admin'@'localhost';

grant execute on procedure sweater.GetUserById to 'admin'@'localhost';

grant execute on procedure sweater.GetUserByUsername to 'admin'@'localhost';

grant execute on procedure sweater.LikeMessage to 'admin'@'localhost';

grant execute on procedure sweater.Subscribe to 'admin'@'localhost';

grant execute on procedure sweater.Unsubscribe to 'admin'@'localhost';

grant execute on procedure sweater.UpdateMessage to 'admin'@'localhost';

grant execute on procedure sweater.DeleteMessage to 'admin'@'localhost';

grant execute on procedure sweater.UpdateUser to 'admin'@'localhost';

grant execute on function sweater.meLiked to 'admin'@'localhost';

grant execute on function sweater.GetLastInsertedMessageXML to 'admin'@'localhost';

grant execute on function sweater.GetOneMessagesWithTheMostLikes to 'admin'@'localhost';

**Скрипты создания индексов**

-- index for search by username or usermail

alter table sweater.user add fulltext index(username, email);

-- index for unsubscribe

create index channel\_id\_subscriber\_id on user\_subscriptions(channel\_id, subscriber\_id);

-- index for dislike

create index user\_id\_message\_id on message\_likes(user\_id, message\_id);

-- index for search users by id

create unique index user\_id on user(id);

-- index for search users by username

create unique index username\_only on user(username);

-- index for search users by activation code

create index activation\_code on user(activation\_code);

-- index for search by sub\_id

create index channel\_id on user\_subscriptions(channel\_id);

-- index for search by sub\_id

create index sub\_id on user\_subscriptions(subscriber\_id);

-- index for search by text

create index text\_index on message(text);

-- index for search by text

create index tag\_index on message(tag);

-- index for serach message by tag

alter table sweater.message add fulltext index(tag, text);

-- index for search message by user\_id

create index message\_user\_id on message(user\_id);

**Скрипты хранимых процедур для работы с таблицей USER**

-- -------------------------------------------------------------------

delimiter $$

create procedure GetAllUsers()

begin

select \* from user;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetUserById(in id bigint)

begin

select

user0\_.id as id,

user0\_.activation\_code as activationCode,

user0\_.active as active,

user0\_.email as email,

user0\_.password as password,

user0\_.username as username

from

user user0\_ where user0\_.id = id;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure UpdateUser(in id bigint, in email varchar(255), in password varchar(255), in activationCode varchar(255))

begin

update user set user.email = email, password = user.password, user.activation\_code = activationCode where user.id = id;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetUserByActivationCode(in code varchar(255))

begin

select user0\_.id as id,

user0\_.activation\_code as activationCode,

user0\_.active as active,

user0\_.email as email,

user0\_.password as password,

user0\_.username as username

from user where user.activation\_code = code;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetUserByUsername(in username varchar(255))

begin

select \* from user where user.username = username;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure AddUser(in activation\_code varchar(255), in active bit, in email varchar(255), in password varchar(255), in username varchar(255))

begin

declare c int unsigned default 0;

declare last\_id int unsigned default 0;

set c := (select count(\*) from user where user.username = username);

if (c != 0) then

signal sqlstate '45001' set message\_text = 'User already exists!';

else

insert into user(activation\_code, active, email, password, username)

values (activation\_code, active, email, password, username);

set last\_id := last\_insert\_id();

insert into user\_role(user\_id, roles) values (last\_id, 'USER');

end if;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure ActivateUser(in id bigint)

begin

update user set activation\_code = null where user.id = id;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetAllSimilarUsersByUsername(in username varchar(255))

begin

select

user0\_.id as id,

user0\_.activation\_code as activationCode,

user0\_.active as active,

user0\_.email as email,

user0\_.password as password,

user0\_.username as username

from

user user0\_ where match (user0\_.username, user0\_.email) against (username in boolean mode);

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetCountSubscriptions(in id bigint)

begin

select count(\*) from user\_subscriptions where user\_subscriptions.channel\_id = id;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetCountSubscribers (in id bigint)

begin

select count(\*) from user\_subscriptions where user\_subscriptions.subscriber\_id = id;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetCountOfMessage(in id bigint)

begin

select count(\*) from message where message.user\_id = id;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetSubscribersByUserId(in id bigint)

begin

select

user0\_.id as id,

user0\_.activation\_code as activationCode,

user0\_.active as active,

user0\_.email as email,

user0\_.password as password,

user0\_.username as username,

roles1\_.roles as roles

from

user user0\_

left outer join

user\_role roles1\_

on user0\_.id=roles1\_.user\_id where user0\_.id in (select subscriber\_id from user\_subscriptions where channel\_id = id);

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetSubscriptionsByUserId(in id bigint)

begin

select

user0\_.id as id,

user0\_.activation\_code as activationCode,

user0\_.active as active,

user0\_.email as email,

user0\_.password as password,

user0\_.username as username,

roles1\_.roles as roles

from

user user0\_

left outer join

user\_role roles1\_

on user0\_.id=roles1\_.user\_id where user0\_.id in (select channel\_id from user\_subscriptions where subscriber\_id = id);

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetRolesByUserId(in id bigint)

begin

select roles.roles as roles

from

user\_role roles

where

roles.user\_id= id;

end $$

delimiter ;

-- -------------------------------------------------------------------

**Процедуры для работы с таблицей MESSAGE**

-- -------------------------------------------------------------------

delimiter $$

create procedure UpdateMessage(in id bigint, in filename varchar(255), in tag varchar(255), in text varchar(255), in user\_id bigint)

begin

update sweater.message set message.filename = filename, message.tag = tag, message.text = text, message.user\_id = user\_id where message.id = id;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure DeleteMessage(in id bigint)

begin

delete from sweater.message where message.id = id;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure AddMessage(in filename varchar(255), in tag varchar(255), in text varchar(255), in user\_id bigint)

begin

insert into message (message.filename, message.tag, message.text, message.user\_id) values (filename, tag, text, user\_id);

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetAllMessageDto(in cur\_id bigint)

begin

declare id bigint default 0;

declare filename varchar(255);

declare tag varchar(255);

declare text varchar(2048);

declare author\_id bigint;

declare countOfLikes bigint;

declare meLiked boolean;

declare author\_name varchar(255);

DECLARE done INT DEFAULT 0;

declare cur cursor for select

m.id, m.filename, m.tag, m.text, m.user\_id as author\_id,

count(ml.user\_id), u.username as author\_name

from message m left join message\_likes ml on m.id = ml.message\_id left join user u on m.user\_id = u.id group by m.id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

drop temporary table if exists `tmp`;

create temporary table `tmp` (

id bigint,

filename varchar(255),

tag varchar(255),

text varchar(2048),

author\_id bigint,

likes bigint,

meLiked boolean,

author\_name varchar(255)

);

open cur;

read\_loop:

loop fetch cur into id, filename, tag, text, author\_id, countOfLikes, author\_name;

if done = 1 then

leave read\_loop;

end if;

set meLiked := meLiked(cur\_id, id);

insert into tmp (id, filename, tag, text, author\_id, likes, meLiked, author\_name) values

(id, filename, tag, text, author\_id, countOfLikes, meLiked, author\_name);

end loop;

close cur;

select \* from tmp;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create function meLiked(user\_id bigint, message\_id bigint)

returns int

begin

declare result boolean;

declare c int unsigned default 0;

set c := (select count(\*) from message\_likes ml where ml.user\_id = user\_id and ml.message\_id = message\_id);

if(c = 0) then

set result := 0;

else

set result := 1;

end if;

return result;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetAllMessageBySubsAndTagDto(in cur\_id bigint, in tag\_in varchar(255))

begin

declare id bigint default 0;

declare filename varchar(255);

declare tag varchar(255);

declare text varchar(2048);

declare author\_id bigint;

declare countOfLikes bigint;

declare meLiked boolean;

declare author\_name varchar(255);

DECLARE done INT DEFAULT 0;

declare cur cursor for select

m.id, m.filename, m.tag, m.text, m.user\_id as author\_id,

count(ml.user\_id), u.username as author\_name

from message m left join message\_likes ml on m.id = ml.message\_id left join user u on m.user\_id = u.id where m.user\_id in (select channel\_id from user\_subscriptions where subscriber\_id = cur\_id) and match (m.tag, m.text) against (tag\_in in boolean mode) group by m.id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

drop temporary table if exists `tmp`;

create temporary table `tmp` (

id bigint,

filename varchar(255),

tag varchar(255),

text varchar(2048),

author\_id bigint,

likes bigint,

meLiked boolean,

author\_name varchar(255)

);

open cur;

read\_loop:

loop fetch cur into id, filename, tag, text, author\_id, countOfLikes, author\_name;

if done = 1 then

leave read\_loop;

end if;

set meLiked := meLiked(cur\_id, id);

insert into tmp (id, filename, tag, text, author\_id, likes, meLiked, author\_name) values

(id, filename, tag, text, author\_id, countOfLikes, meLiked, author\_name);

end loop;

close cur;

SELECT

\*

FROM

tmp;

end $$

-- -------------------------------------------------------------------

delimiter $$

create procedure GetAllMessageBySubsDto(in cur\_id bigint)

begin

declare id bigint default 0;

declare filename varchar(255);

declare tag varchar(255);

declare text varchar(2048);

declare author\_id bigint;

declare countOfLikes bigint;

declare meLiked boolean;

declare author\_name varchar(255);

DECLARE done INT DEFAULT 0;

declare cur cursor for select

m.id, m.filename, m.tag, m.text, m.user\_id as author\_id,

count(ml.user\_id), u.username as author\_name

from message m left join message\_likes ml on m.id = ml.message\_id left join user u on m.user\_id = u.id where m.user\_id in (select channel\_id from user\_subscriptions where subscriber\_id = cur\_id) group by m.id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

drop temporary table if exists `tmp`;

create temporary table `tmp` (

id bigint,

filename varchar(255),

tag varchar(255),

text varchar(2048),

author\_id bigint,

likes bigint,

meLiked boolean,

author\_name varchar(255)

);

open cur;

read\_loop:

loop fetch cur into id, filename, tag, text, author\_id, countOfLikes, author\_name;

if done = 1 then

leave read\_loop;

end if;

set meLiked := meLiked(cur\_id, id);

insert into tmp (id, filename, tag, text, author\_id, likes, meLiked, author\_name) values

(id, filename, tag, text, author\_id, countOfLikes, meLiked, author\_name);

end loop;

close cur;

select \* from tmp;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetAllMessageByTagDto(in cur\_id bigint, tag\_in varchar(255))

begin

declare id bigint default 0;

declare filename varchar(255);

declare tag varchar(255);

declare text varchar(2048);

declare author\_id bigint;

declare countOfLikes bigint;

declare meLiked boolean;

declare author\_name varchar(255);

DECLARE done INT DEFAULT 0;

declare cur cursor for select

m.id, m.filename, m.tag, m.text, m.user\_id as author\_id,

count(ml.user\_id), u.username as author\_name

from message m left join message\_likes ml on m.id = ml.message\_id left join user u on m.user\_id = u.id where match (m.tag, m.text) against (tag\_in in boolean mode) group by m.id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

drop temporary table if exists `tmp`;

create temporary table `tmp` (

id bigint,

filename varchar(255),

tag varchar(255),

text varchar(2048),

author\_id bigint,

likes bigint,

meLiked boolean,

author\_name varchar(255)

);

open cur;

read\_loop:

loop fetch cur into id, filename, tag, text, author\_id, countOfLikes, author\_name;

if done = 1 then

leave read\_loop;

end if;

set meLiked := meLiked(cur\_id, id);

insert into tmp (id, filename, tag, text, author\_id, likes, meLiked, author\_name) values

(id, filename, tag, text, author\_id, countOfLikes, meLiked, author\_name);

end loop;

close cur;

select \* from tmp;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure GetAllMessageByUserDto(in user\_id bigint, in cur\_id bigint)

begin

declare id bigint default 0;

declare filename varchar(255);

declare tag varchar(255);

declare text varchar(2048);

declare author\_id bigint;

declare countOfLikes bigint;

declare meLiked boolean;

declare author\_name varchar(255);

DECLARE done INT DEFAULT 0;

declare cur cursor for select

m.id, m.filename, m.tag, m.text, m.user\_id as author\_id,

count(ml.user\_id), u.username as author\_name

from message m left join message\_likes ml on m.id = ml.message\_id left join user u on m.user\_id = u.id where m.user\_id = user\_id group by m.id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

drop temporary table if exists `tmp`;

create temporary table `tmp` (

id bigint,

filename varchar(255),

tag varchar(255),

text varchar(2048),

author\_id bigint,

likes bigint,

meLiked boolean,

author\_name varchar(255)

);

open cur;

read\_loop:

loop fetch cur into id, filename, tag, text, author\_id, countOfLikes, author\_name;

if done = 1 then

leave read\_loop;

end if;

set meLiked := meLiked(cur\_id, id);

insert into tmp (id, filename, tag, text, author\_id, likes, meLiked, author\_name) values

(id, filename, tag, text, author\_id, countOfLikes, meLiked, author\_name);

end loop;

close cur;

select \* from tmp;

end $$

delimiter ;

-- -------------------------------------------------------------------

**Процедуры для работы с системой ПОДПИСКИ/ОТПИСКИ, ЛАЙК/ДИЗЛАЙК**

-- -------------------------------------------------------------------

delimiter $$

create procedure Subscribe(in id bigint, in sub\_id bigint)

begin

insert into user\_subscriptions(channel\_id, subscriber\_id) values (id, sub\_id);

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure Unsubscribe(in id bigint, in sub\_id bigint)

begin

delete from user\_subscriptions where channel\_id = id and subscriber\_id = sub\_id;

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure LikeMessage(in user\_id bigint, in message\_id bigint)

begin

insert into message\_likes(user\_id, message\_id) values (user\_id, message\_id);

end $$

delimiter ;

-- -------------------------------------------------------------------

delimiter $$

create procedure DislikeMessage(in u\_id bigint, in m\_id bigint)

begin

delete from message\_likes where user\_id = u\_id and message\_id = m\_id;

end $$

delimiter ;

-- -------------------------------------------------------------------

**Скрипты создания триггеров**

delimiter $$

create trigger insert\_user after insert on sweater.user

for each row begin

insert into `log\_user` set message = concat('INSERTED (user\_id/username): ', new.id, '/', new.username, ', usename: ', new.email), row\_id = new.id;

end;

delimiter $$

create trigger subscribe after insert on sweater.user\_subscriptions

for each row begin

insert into `log\_subs` set message = concat('SUBSCRIBED (user\_id): ', new.subscriber\_id, ', up for (user\_id): ', new.channel\_id);

end;

delimiter $$

create trigger add\_message after insert on sweater.message

for each row begin

insert into `log\_message` set message = concat('ADDED: message\_id: ', new.id, ', by whom (user\_id): ', new.user\_id), row\_id = new.id;

end;

delimiter $$

create trigger update\_user before update on sweater.user

for each row begin

insert into `log\_user` set message = concat('UPDATED (user\_id/username): ', new.id, '/', new.username, ',old email:',old.email,' -> new email: ', new.email), row\_id = new.id;

end;

delimiter $$

create trigger unsubscribe before delete on sweater.user\_subscriptions

for each row begin

insert into `log\_subs` set message = concat('UNSUBSCRIBED (user\_id): ', old.subscriber\_id, ', up for (user\_id): ', old.channel\_id);

end;

delimiter $$

create trigger update\_message after update on sweater.message

for each row begin

insert into `log\_message` set message = concat('UPDATED: message\_id: ', new.id, ', by whom (user\_id): ', new.user\_id), row\_id = new.id;

end;

delimiter $$

create trigger delete\_message before delete on sweater.message

for each row begin

insert into `log\_message` set message = concat('DELETED: message\_id: ', old.id, ', by whom (user\_id): ', old.user\_id), row\_id = old.id;

end;

-- -------------------------------------------------------------------

**Скрипты функций работы с XML**

-- ------------------------------------------------- XML Import/Export -----------------------------------------------

set @xml = '<messages>

<message tag="hiFromXml">

<text>This text took from xml</year>

<userId>1</userId>

</message>

</messages>';

select (select ExtractValue(@xml, '//message/@tag')) as tag;

select (select ExtractValue(@xml, '//message/text')) as text;

select (select ExtractValue(@xml, '//message/userId')) as user\_id;

delimiter $$

create function GetLastInsertedMessageXML()

returns nvarchar(2048)

begin

declare msg nvarchar(2048);

set msg := (select concat('<message id="',m.id,'">','<filename>',ifnull(m.filename,'null'),'</filename><tag>',m.tag,'</tag><text>',m.text,'</text><userId>',m.user\_id,'</userId></message>') from message m order by id desc limit 1);

set msg := concat('<messages func="last">',msg,'</messages>');

return msg;

end $$

delimiter ;

delimiter $$

create function GetOneMessagesWithTheMostLikes()

returns varchar(4096)

begin

declare msg varchar(4096);

declare likes bigint;

set msg := (

select concat( '<message id="',m.id,'">',

'<filename>',ifnull(m.filename,'null'),'</filename>',

'<tag>',m.tag,'</tag>',

'<text>',m.text,'</text>',

'<userId>',m.user\_id,'</userId>',

'<likes>',count(ml.message\_id),'</likes>',

'</message>')

from message m left outer join message\_likes ml

on ml.message\_id = m.id group by m.id order by count(ml.message\_id) desc limit 2);

set msg := concat('<messages func="TheMostLikes">',msg,'</messages>');

return msg;

end $$

delimiter ;

select GetLastInsertedMessageXML();

select GetOneMessagesWithTheMostLikes();

-- ------------------------------------------------- XML Import/Export END-----------------------------------------