Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**Департамент анализа данных,**

**принятия решений и финансовых технологий**

**ЭТОТ ТИТУЛЬНИК НЕПРАВИЛЬНЫЙ, СКАЗАЛИ ВЫШЛЮТ НОВЫЙ**

на тему:

**Разработка приложения-чата с использованием**

**библиотек Spring Boot и JavaFX**

Выполнил:

студент группы ПИ19-4

Деменчук Георгий Максимович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Научный руководитель:

Никитин Петр Владимирович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

2021

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 4](#_Toc70371879)

[ПОСТАНОВКА ЗАДАЧИ 5](#_Toc70371880)

[ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc70371881)

[АКТУАЛЬНОСТЬ АВТОМАТИЗАЦИИ 8](#_Toc70371882)

[АЛГОРИТМИЧЕСКИЕ РЕШЕНИЯ 9](#_Toc70371883)

[Клиент 9](#_Toc70371884)

[Сервер 11](#_Toc70371885)

[ОПИСАНИЕ ИНТЕРФЕЙСА ПРОГРАММЫ 13](#_Toc70371886)

[Окно авторизации клиента 14](#_Toc70371887)

[Окно регистрации нового пользователя 15](#_Toc70371888)

[Окно восстановления пароля 16](#_Toc70371889)

[Окно успешного действия пользователя 17](#_Toc70371890)

[Главное окно программы 18](#_Toc70371891)

[Окно создания новой чат-комнаты 19](#_Toc70371892)

[Окно с информацией о текущей чат-комнате 19](#_Toc70371893)

[Окно с информацией об авторе и программе 20](#_Toc70371894)

[СОСТАВ ПРИЛОЖЕНИЯ 21](#_Toc70371895)

[Сервер 21](#_Toc70371896)

[База данных 22](#_Toc70371897)

[Клиент 23](#_Toc70371898)

[НАЗНАЧЕНИЕ И СОСТАВ КЛАССОВ ПРОГРАММЫ 25](#_Toc70371899)

[Сервер 25](#_Toc70371900)

[Клиент 29](#_Toc70371901)

[ЗАКЛЮЧЕНИЕ 34](#_Toc70371902)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 35](#_Toc70371903)

[ПРИЛОЖЕНИЯ 36](#_Toc70371904)

[Приложение 1. Исходный код для развертки решения 36](#_Toc70371905)

[Приложение 2. Исходный код сервера 36](#_Toc70371906)

[Приложение 3. Исходный код клиента 78](#_Toc70371907)

# ВВЕДЕНИЕ

Примерно с 2010 годов можно наблюдать эволюцию приложений-мессенджеров. Сначала быструю популярность набрал WhatsApp за счет своей простоты использования и привязки пользователя к его номеру телефона. Затем, в 2011 году социальная сеть Facebook представила гибридный мессенджер Facebook Messenger: обособленное от основной социальной сети приложение, в котором можно было общаться как с пользователями Facebook, так и только по номеру телефона. В том же году был представлен китайский WeChat, который включал в себя не только мессенджер, но и локальную платёжную систему. В 2013 году был представлен Telegram, который одним из первых внедрил официальное API с документацией для ботов в своём сервисе. Через год вышел Signal Messenger, в котором уделялось повышенное внимание конфиденциальности данных пользователей. Таким образом, появилось множество мессенджеров со своими уникальными свойствами и особенностями, но неизменным оставалось одно: возможность предавать текст на любые расстояния.

Но как происходит этот процесс передачи данных? Какие механизмы и алгоритмы используются при передаче и получении сообщений? Целью данной курсовой работы и является получение ответов на эти вопросы.

В качестве демонстрации будет разработан прототип на языке программирования Java, содержащий в себе 2 решения, взаимодействующих между собой с помощью архитектуры REST: серверная часть с использованием фреймворка Spring Boot и технологией Long Poll, а также клиентская часть: с графическим интерфейсом с использованием библиотеки JavaFX. Оба решения будут использовать модель MVC, которая разграничивает управляющую логику программы на отдельные компоненты, а за счёт применения Java и виртуальной машины JVM решение будет кроссплатформенным.

# ПОСТАНОВКА ЗАДАЧИ

В соответствии с выбранной темой требуется разработать клиент-серверное решение с использованием библиотек Spring Boot для сервера и JavaFX для GUI клиента в виде пользовательских классов и таблиц для СУБД.

Со стороны клиента необходимо разработать несколько окон и логику переходов пользователей между ними, а также их дизайн и расположение элементов интерфейса для взаимодействия с пользователем.

Cо стороны бекенда необходимо использовать ORM для связи Spring с СУБД, а также модель MVC для отдельного расположение контроллеров, сервисов и репозиториев с логикой таблиц СУБД.

Решение должно выполнять следующие операции:

* Отображать в таблице данные предметной области;
* Для информационной модели, основанной на БД, таблицы должны быть предварительно заполнены записями;
* Добавлять в БД новые объекты, удалять и редактировать их;
* Фильтровать записи в БД, которые удовлетворяют введенному пользователем сложному критерию;
* Сортировать записи;
* Обновлять изменения источника данных в базе данных;
* Загружать данные из БД;
* Отображать статистические данные.

Решение не должно завершаться аварийно: сообщения о некорректном вводе данных, противоречивых или недопустимых значениях данных, при отсутствии данных по функциональному запросу пользователя и других нештатных ситуациях отображать в окнах сообщений.

Программа должна иметь содержательные комментарии, которые могут генерировать автоматически составляемую документацию при помощи инструмента JavaDoc.

# ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Предметной областью автоматизации является чат-мессенджер. Логика его работы достаточно проста: пользователь создает чат-комнату с указанием имени и выбором пользователей-участников, переходит в неё и начинает общение. Участника в комнате может быть два или более, тогда чат-комната будет диалогом или беседой соответственно. В окне диалога у пользователя существует возможность чтения предыдущих сообщений, отправки и получения новых, а также просмотра информации об участниках чата.

Всего со стороны клиента используются 3 сущности для представления данных:

* Пользователь – сущность представления конкретного пользователя решения. В один времени может быть авторизован только один пользователь со стороны клиента;
* Комната – чат конкретных пользователей-участников. У одного пользователя может быть несколько комнат, у комнаты может быть несколько участников, но только один пользователь-администратор;
* Сообщение – текстовое сообщение пользователя в чате. Содержит информацию об идентификаторе пользователя и чат-комнаты, к которой принадлежит.

Со стороны же сервера дополнительно объявляется сущность лонгпула, которая содержит в себе основную информацию о длинных соединениях клиента с сервером. Более подробно о об атрибутах сущностей можно прочитать в пункте «База данных» главы «СОСТАВ ПРИЛОЖЕНИЯ».

# АКТУАЛЬНОСТЬ АВТОМАТИЗАЦИИ

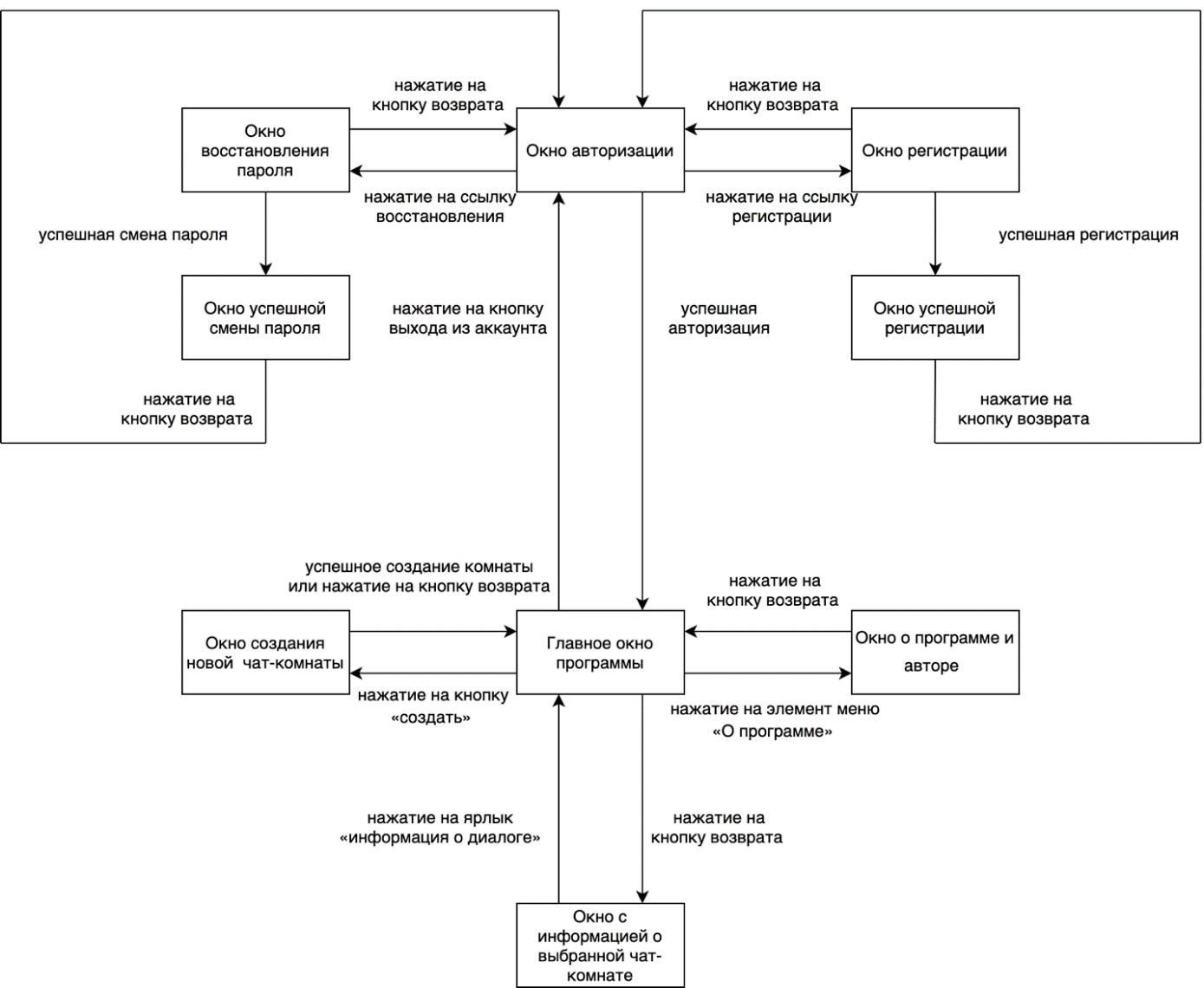
Автоматизация получения сообщений с помощью собственно разработанного решения позволяет оперативно передавать абсолютно любую информацию в текстовом виде, при этом храня все данные исключительно на своих серверах и базах данных, что критично для проектов с повышенным вниманием к конфиденциальности данных, а за счет использования кроссплатформенных решений, запуск приложения-клиента возможен на всех платформах (Windows, OS X, Linux) и архитектурах (x86, ARM).

Если серверная часть решения размещается на территории Российской Федерации, то программа будет исполнять ФЗ-152 «О защите персональных данных», что также важно для компаний, которые работают с данными пользователей РФ.

# АЛГОРИТМИЧЕСКИЕ РЕШЕНИЯ

## Клиент

На рисунке 1 ниже представлены переходы пользователя между данными формами, кроме отдельного окна «Нет подключения к интернету» ‎. Оно не представлено на схеме т.к. переход на него может быть осуществлён абсолютно с любого окна программы и не зависит от действий пользователя. Данный элемент программы вызывается с помощью отдельного потока, который проверяет соединение с интернетом. Когда интернет появляется снова, то осуществляется переход на одно из 2-х окон, в зависимости от флага авторизации пользователя: окно авторизации или главное меню программы. Каждое окно программы использует адаптивное масштабирование элементов интерфейса, в зависимости от разрешения и соотношения сторон экрана пользователя.

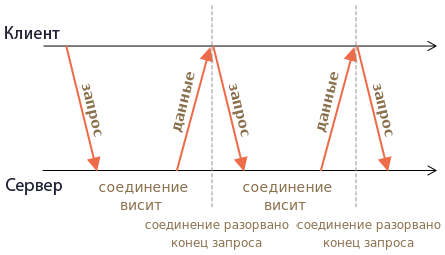


#### Рисунок – Переходы пользователя между формами в программе-клиенте

При запуске программы происходит проверка на существование ключа доступа (токена) в файле .tempdata. Если файла нет, то он создаётся. Если же файл существует и в нем есть информация о токене, то программа попытается передать его на сторону сервера. Сервер принимает запрос, ищет соответствующий токен в коллекции Users в MongoDB и отдаёт ответ клиенту ответ. Если ответ сервера положительный, то откроется главное окно программы, иначе происходит переход на форму авторизации пользователя.

Во время запуска главного окна программы происходит запуск 2 дополнительных потоков: поток проверки интернет-соединения и поток получения новых сообщений с помощью технологии лонгпулинга.

Технология лонгпулинга позволяет получать информацию о новых событиях с помощью длинных запросов. На рисунке 2 представлен алгоритм работы: сервер получает запрос, но отправляет ответ на него не сразу, а лишь только тогда, когда произойдет какое-либо событие, например, поступит новое входящее сообщение или же истечёт заданное время ожидания. Другими словами, получая новый запрос, сервер ждёт, когда произойдет событие, о котором надо уведомить. Когда событие произойдет, то сервер отправляет ответ на запрос, содержащий информацию о случившимся событии.



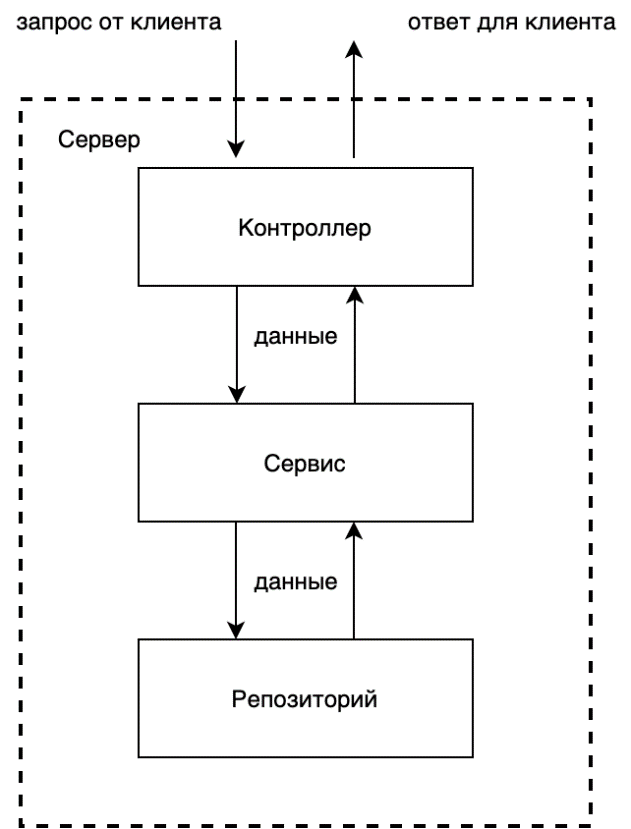
#### Рисунок – Пример работы технологии лонгпулинга

Если клиенту пришло событие о новом сообщении для пользователя, то он отобразит его в соответствующей комнате. Если такой комнаты нет, то добавит её в список чат-комнат пользователя. Если сервер сообщает клиенту, что время ожидания запроса истекло, то клиент обратится заново к серверу с просьбой создания нового соединения.

Поток проверки интернет-соединения проверяет доступность ip-адреса сервера по его открытому порту каждые 5 секунд. В случае, если ответа нет, то происходит синхронизация потоков и отображение окна с ошибкой интернет-соединения. Когда соединение снова возвращается, то программа снова синхронизирует потоки и автоматически переходит на главное окно с чатами.

## Сервер

При получении запроса от клиента, сервер сопоставляет переданный пусть URL с каким-либо из существующих контроллеров. Если такой ассоциации нет, то сервер возвратит ошибку 404 not found, если же контроллер действительно существует, то Spring сопоставляет переданные данные с теми, которые требуются в аргументах контроллера. Если переданные данные некорректные или переданы не в полном объёме, то сервер вернёт ошибку 400 Bad Request. Иерархия обработки и передачи данных со стороны сервера показана на рисунке 3 ниже.



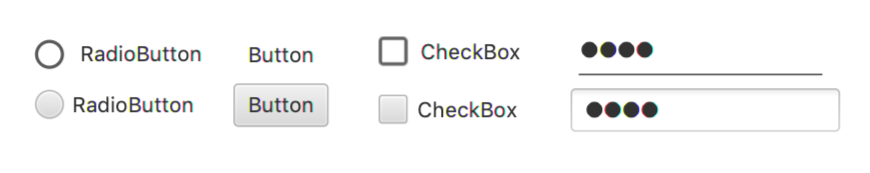
#### Рисунок – Работа с потоками данных со стороны сервера

Если же все переданные данные корректны, то из соответствующего контроллера вызывается сервис с логикой, которая отвечает за обработку полученных данных и передачу их в методы репозитория (класса, который наследуется от интерфейса MongoRepository). После получения данных от репозитория, сервис обрабатывает полученный результат и передаёт данные обратно, на уровень контроллера.

Контроллер, получивший данные от сервиса, осуществляет постобработку и возвращает ответ клиенту, сделавшему соответствующий запрос.

# ОПИСАНИЕ ИНТЕРФЕЙСА ПРОГРАММЫ

Т.к. у сервера не существует графического интерфейса, то очевидно, что в этом разделе речь пойдет о клиенте на JavaFX. Для проектирования интерфейсов использовалась специальная библиотека JFoenix для минималистичного material дизайна, которая расширяет возможности по кастомизации стандартных элементов JavaFX разрабатываемой системы. Ниже представлено сравнение однотипных элементов интерфейса с использованием JFoenix (верхний элемент) и без него (нижний элемент).





#### Рисунок – Сравнение стандартных элементов JavaFX с JFoenix

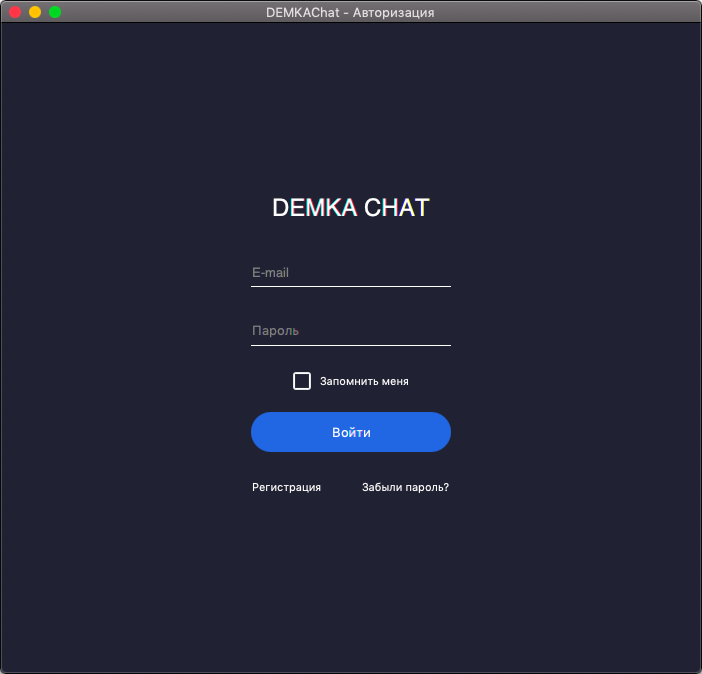
Как видим из рисунка 4 выше, элементы JFoenix выглядят более аккуратно по сравнению со стандартным набором, что особенно хорошо видно на элементах Slider, ProgressBar и TextField.

Всего в клиенте представлено 10 окон:

* Окно авторизации в системе;
* Окно регистрации в системе;
* Окно сброса пароля системы;
* Окна успешного действия пользователя (регистрации или авторизации);
* Главное меню программы;
* Окно «Об авторе» ‎;
* Окно создания новой чат-комнаты;
* Окно с информацией о выбранном чате;
* Окно «Нет подключения к интернету» ‎.

## Окно авторизации клиента

Запрашивает логин/пароль пользователя и предлагает запомнить информацию об авторизации для автоматического входа при следующем запуске программы. Если же пользователь еще не зарегистрирован в системе, то по нажатию на ссылку регистрации будет осуществлён переход на форму регистрации. Если пользователь забыл пароль, то у него есть возможность перехода на форму восстановления пароля через одноимённую ссылку. При корректной авторизации пользователя происходит переход на главную форму программы.



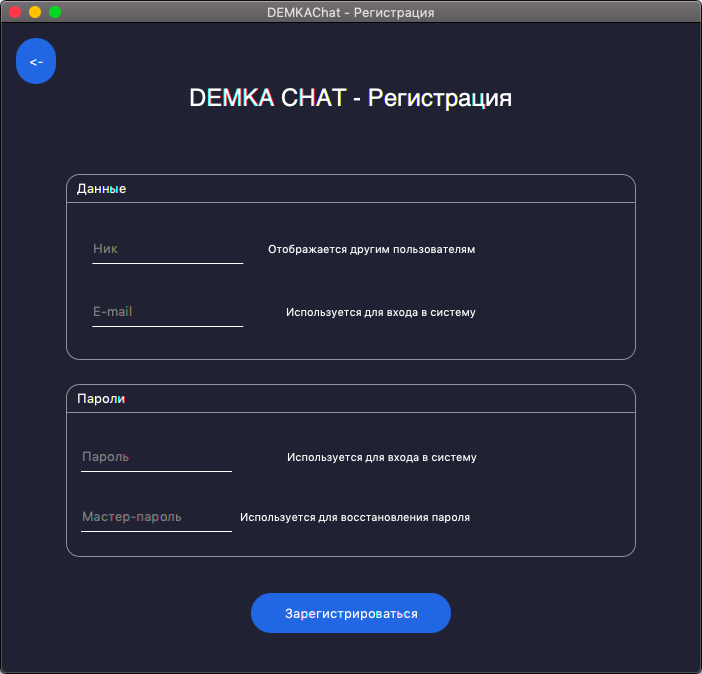
#### Рисунок – Окно авторизации клиента

На форме существует следующие валидаторы входных данных:

* Фильтрация на пустые поля e-mail и пароля (на стороне клиента);
* Фильтрация на некорректную авторизацию по паре логин/пароль (на стороне сервера).

## Окно регистрации нового пользователя

Запрашивает данные для авторизации пользователя: ник (логин), e-mail, пароль и мастер-пароль. Ник необходим для отображения при получении нового сообщения от пользователя, мастер-пароль (он же мастер-ключ) нужен для смены пароля пользователя. Пара e-mail/пароль необходимы для последующей авторизации пользователя в системе.



#### Рисунок – Окно регистрации нового пользователя

При нажатии на кнопку регистрации будет осуществлены следующие проверки на стороне клиента:

* Корректность ввода e-mail;
* Проверка на пустые поля;
* Проверка на длину пароля и мастер-пароля (не менее 8 символов);
* Проверка, чтоб пароль и мастер-пароль не совпадали между собой.

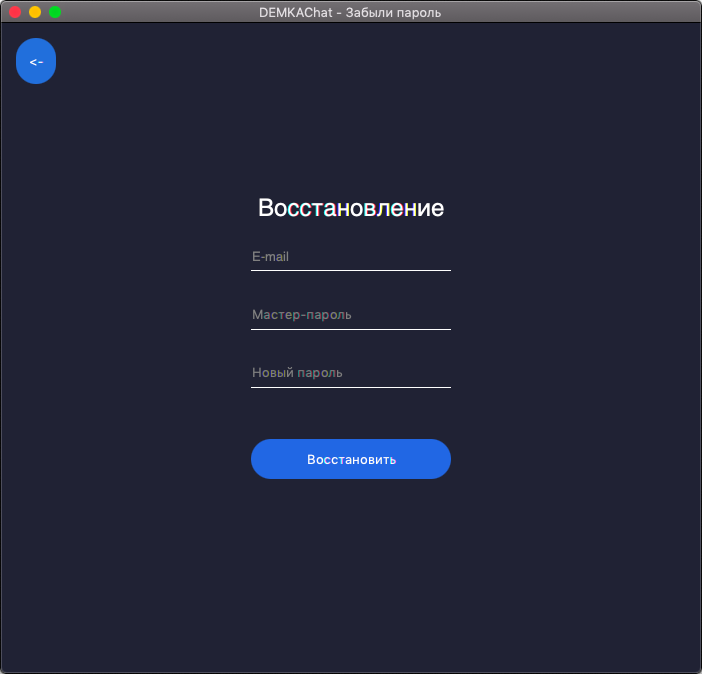
На стороне сервера:

* Проверка на существование пользователя с указанным e-mail или ником;
* Валидация логина (e-mail) пользователя.

При успешной регистрации осуществляется переход на окно с информированием о соответствующем действии, если же валидация данных не прошла, то пользователю отображается соответствующая ошибка.

## Окно восстановления пароля

Необходимо для восстановления пароля пользователя по мастер-ключу, предварительно указанному при регистрации. Запрашивает мастер-ключ, e-mail и новый пароль пользователя. При успешном восстановлении осуществляется переход на окно информирования об успешном действии, также возможен переход на форму авторизации с помощью кнопки возврата.

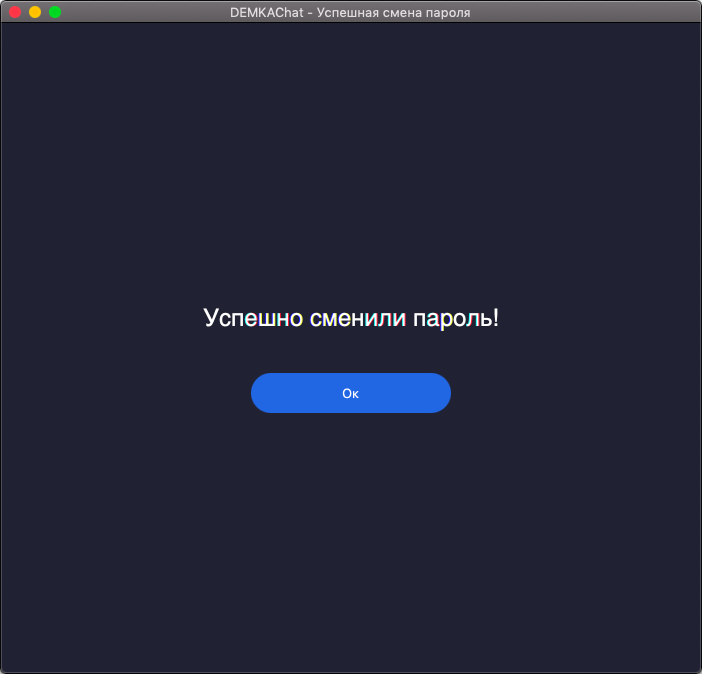
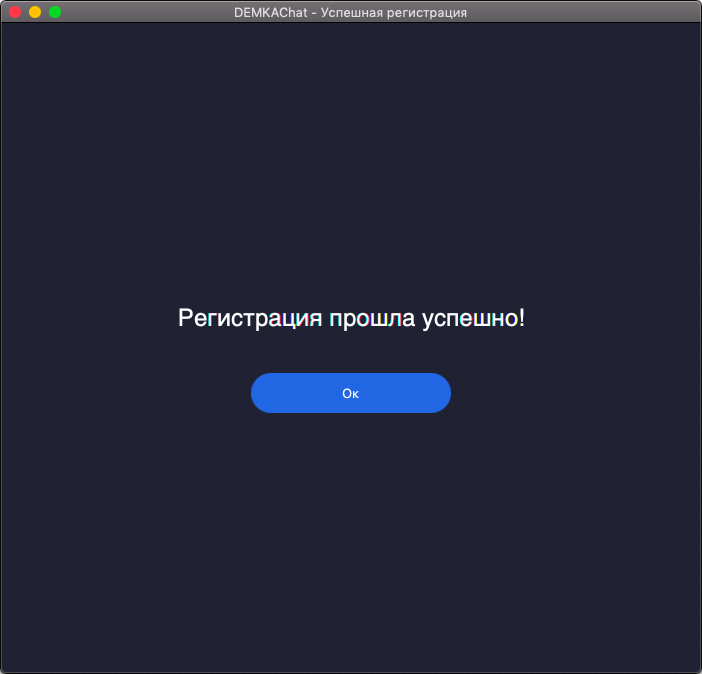


#### Рисунок – Окно восстановления пароля

На стороне клиента осуществляется проверка на пустые поля, корректность e-mail, а также длину паролей. Со стороны сервера реализована проверка на существование пользователя по паре e-mail и мастер-пароля.

## Окно успешного действия пользователя

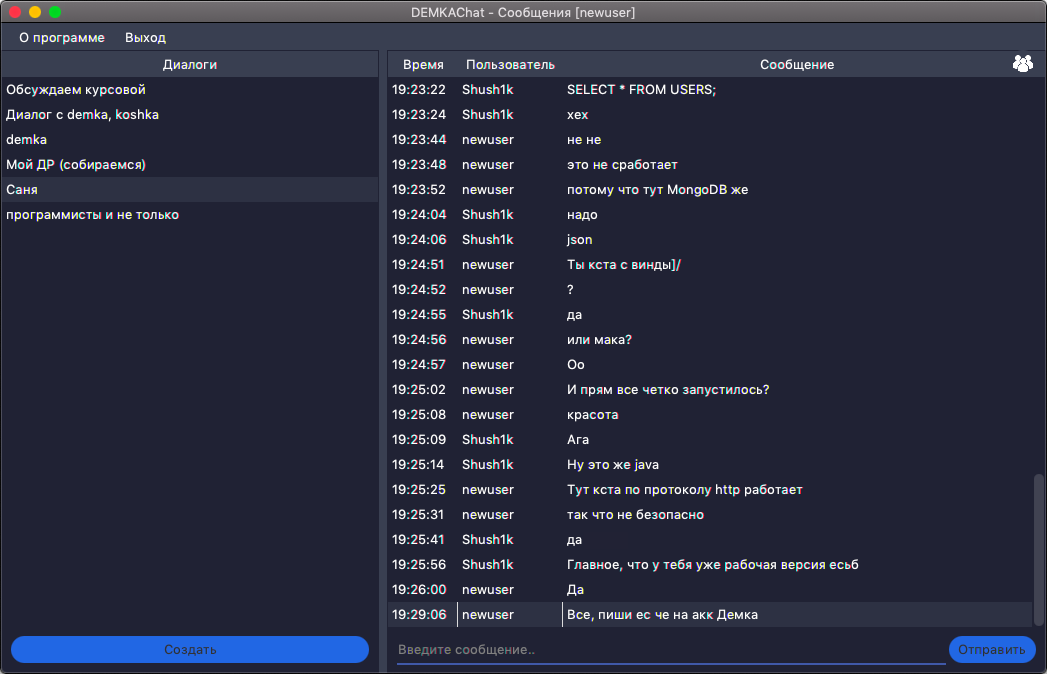
Представляет из себя одно окно с динамическим содержимым, необходимо для информирования об успешной операции: либо регистрации, либо смены пароля. Возможен переход только на форму авторизации путём нажатия единственной кнопки.



#### Рисунок – Окна успешного действия пользователя

## Главное окно программы

Слева окна расположены все чат-комнаты пользователя. При нажатии на какую-либо из них справа отобразятся сообщения, которые содержатся в данной комнате. Все новые сообщения и комнаты добавляются автоматически без каких-либо действий со стороны пользователя. Внизу находятся кнопки создания нового чата и отправки сообщения. Сверху расположены пункты меню «О программе» и «Выход», правее расположен значок информации о выбранном чате. Элементы для информации о чате и отправки сообщения скрываются, если чат не выбран.

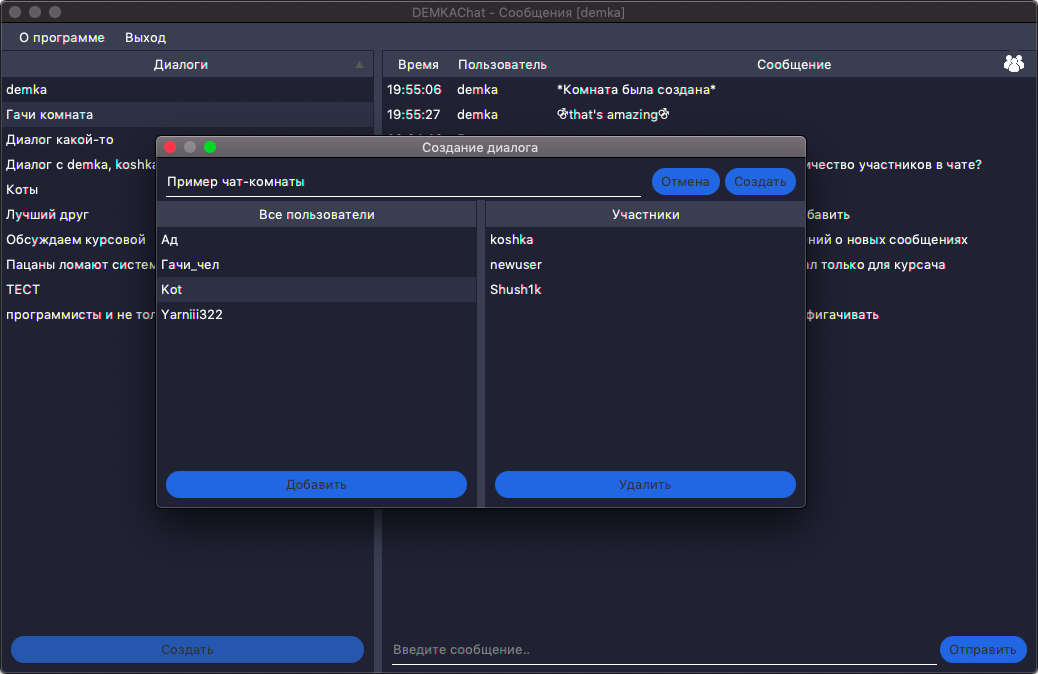


#### Рисунок – Главное окно программы

Если пользователь попытается отправить пустое сообщение, то программа не даст ему это осуществить. С этого окна возможны переходы на формы с информацией о программе, авторизации, создания новой комнаты, а также информацией о текущем чате.

## Окно создания новой чат-комнаты

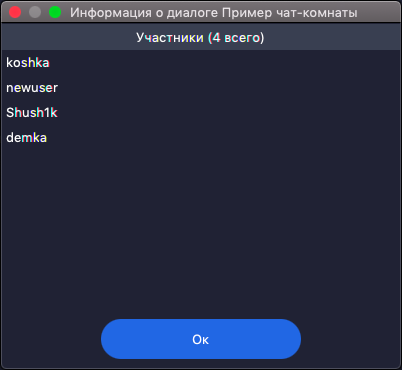
При регистрации пользователя и первом входе в систему, его список диалогов пуст. Для создания нового диалога используется соответствующее окно. Само окно разделено на 2 части: слева показываются все участники системы, а справа те пользователи, которых мы пригласим во вновь созданную комнату. Если название комнаты не было указано, то оно сгенерируется автоматически по шаблону «Чат с НИК\_ПОЛЬЗОВАТЕЛЯ». Если комната была успешно создана, то она появится в общем списке комнат в главном окне программе.



#### Рисунок – Окно создания новой чат-комнаты

## Окно с информацией о текущей чат-комнате

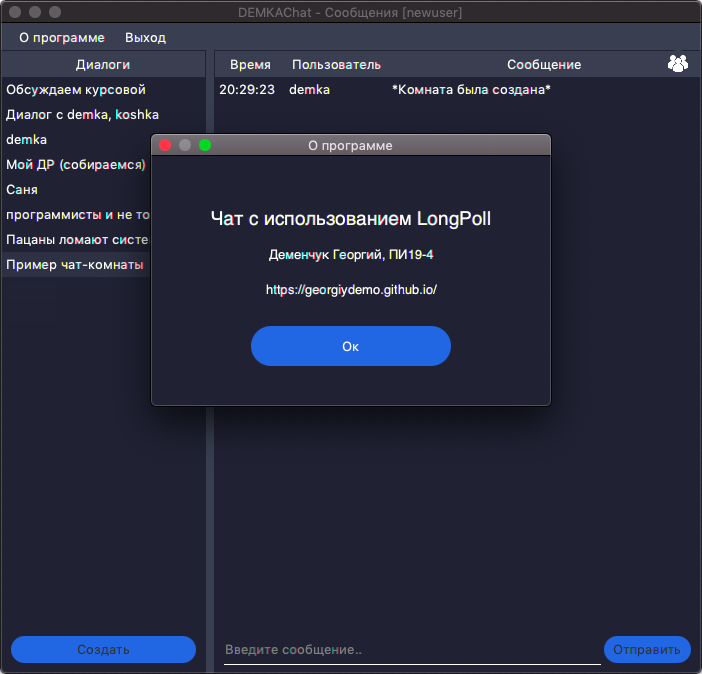
При активной чат-комнате возможно нажатие на значок статистики и информации о текущей комнате. В окне отображается информация об участниках беседы: их никнеймах и количестве. При нажатии на кнопку «Ок» осуществляется закрытие окна.



#### Рисунок – Окно с информацией о текущей чат-комнате

## Окно с информацией об авторе и программе

Форма показывает основную информацию о разработанной программе, а также авторе: ФИО и ссылку на web-сайт разработчика. При нажатии на ссылку происходит её открытие в браузере пользователя. При нажатии на кнопку «Ок» осуществляется закрытие окна.



#### Рисунок – Окно с информацией об авторе и программе

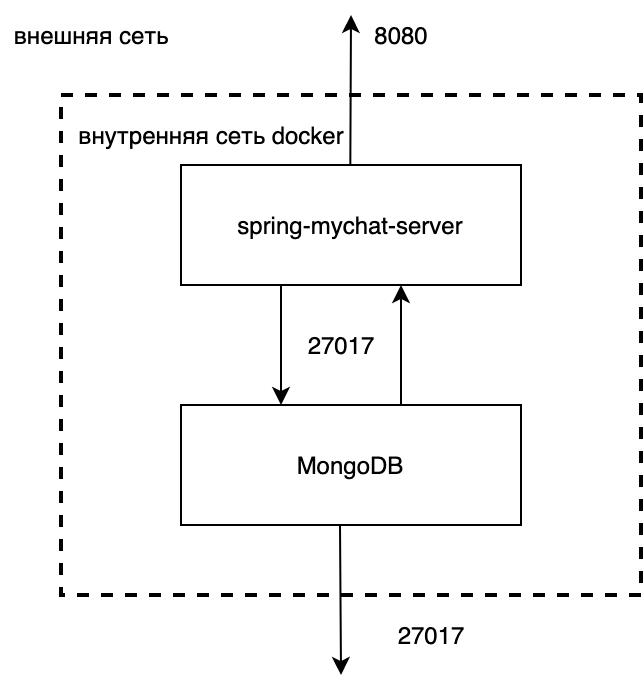
# СОСТАВ ПРИЛОЖЕНИЯ

## Сервер

В состав сервера входят следующие компоненты:

* Spring Web для создания web-приложений, в том числе RESTful, с использованием Spring MVC (Model-View-Controller). Использует сервер Apache Tomcat по умолчанию;
* Spring Data MongoDB – используется для представления сущностей в виде документов MongoDB. Содержит интерфейс MongoRepository: аналог JpaRepository, но для NoSQL СУБД;
* Lombok – удобный набор аннотаций для логирования и автопостроения setter’ов и getter’ов;
* Apache maven – основной сборщик проекта;
* Spring-boot-maven-plugin – плагин для сборки решения.

Все решение запаковано в образ docker-контейнера и доступно в репозиториях docker hub и github packages.



#### Рисунок – Конфигурация развертки решения с помощью docker-compose

Для оперативной развертки решения была написана конфигурация для скрипта docker-compose, которая включает в себя деплой 2 контейнеров (MongoDB и Spring Boot) с конфигурацией, указанной на рисунке 13 выше. Во внутренней сети docker передача данных происходит между контейнерами по порту 27017. Во внешнюю сеть пробрасывается 2 порта для получения соединений: 8080 (Apache Tomcat) и 27017 (MongoDB).

## База данных

В качестве СУБД используется документоориентированная NoSQL СУБД MongoDB и состоит из следующих коллекций и полей:

1. LongPolls – буферная коллекция соединений лонгпулов. Происходит очистка каждый раз, когда запускается сервер. Содержит следующие атрибуты:
   * \_id – уникальный идентификатор лонгпула;
   * ts – время создания лонгпула;
   * key – ключ доступа к лонгпулу;
   * user\_id – идентификатор пользователя, с которым связан лонгпул.
2. Messages – коллекция сообщений пользователей. Содержит следующие атрибуты:
   * \_id – уникальный идентификатор сообщения;
   * user\_id – идентификатор пользователя, который отправил сообщение;
   * text – текст сообщения, отправленного пользователем;
   * room\_id – идентификатор чат-комнаты, к которой относится сообщение;
   * time\_created – время создания сообщения.
3. Rooms – коллекция чат-комнат пользователей. Чат-комната может быть как диалогом (в комнате всего два участника), так и конференцией (более двух участников). Содержит следующие атрибуты:
   * \_id – уникальный идентификатор комнаты;
   * creator\_id – идентификатор создателя комнаты или того, кто инициировал диалог;
   * users – вложенная коллекция с идентификаторами пользователей-участников комнаты;
   * name – имя диалога/комнаты для отображения пользователю;
   * time\_created - время создания комнаты.
4. Users – коллекция с информацией о пользователях системы. Содержит следующие атрибуты:
   * \_id – уникальный идентификатор пользователя;
   * name – имя (ник) пользователя;
   * login – (e-mail) логин пользователя;
   * password – пароль пользователя;
   * key – ключ API пользователя;
   * master\_key – мастер-ключ пользователя системы для смены обычного пароля на авторизацию пользователя;
   * time\_created – время регистрации пользователя системы.

## Клиент

В состав клиента входят следующие компоненты/внешние библиотеки:

* JavaFX (javafx-controls, javafx-fxml, javafx-graphics) – библиотека для отображения GUI и её компоненты;
* JFoenix – библиотека для использования кастомных элементов интерфейса с material-дизайном;
* Google GSON – модуль для работы с запаковкой/распаковкой данных в формате JSON;
* SLF4J – решение для логирования действий пользователя в программе;
* Apache maven – основной сборщик решения (автозагрузка и менеджмент необходимых библиотек, работа с плагинами);
* Javafx-maven-plugin – плагин для автоподстановки флагов виртуальной машины Java (VM Options) библиотеки JavaFX при компиляции и запуске проекта. Содержит команду jlink, которая позволяет собирать разработанное решение в готовый модуль.

Скомпилированный образ java runtime image включает в себя следующие папки:

* bin – директория с исполняемыми файлами программы;
* conf – папка с файлами конфигурации программ;
* legal – файлы с лицензиями;
* lib – директория с файлами библиотек для корректной работы разработанного решения.

Содержимое каждой директории зависит от операционной системы, для которой распространяется собранный образ, но в директории bin всегда будут содержаться следующие файлы:

* java – дистрибутив java;
* jrun – программа для запуска машины java;
* RUN\_ME – основной файл для запуска программы. Может иметь разрешение .bat (Windows) или .sh (UNIX-подобные ОС).

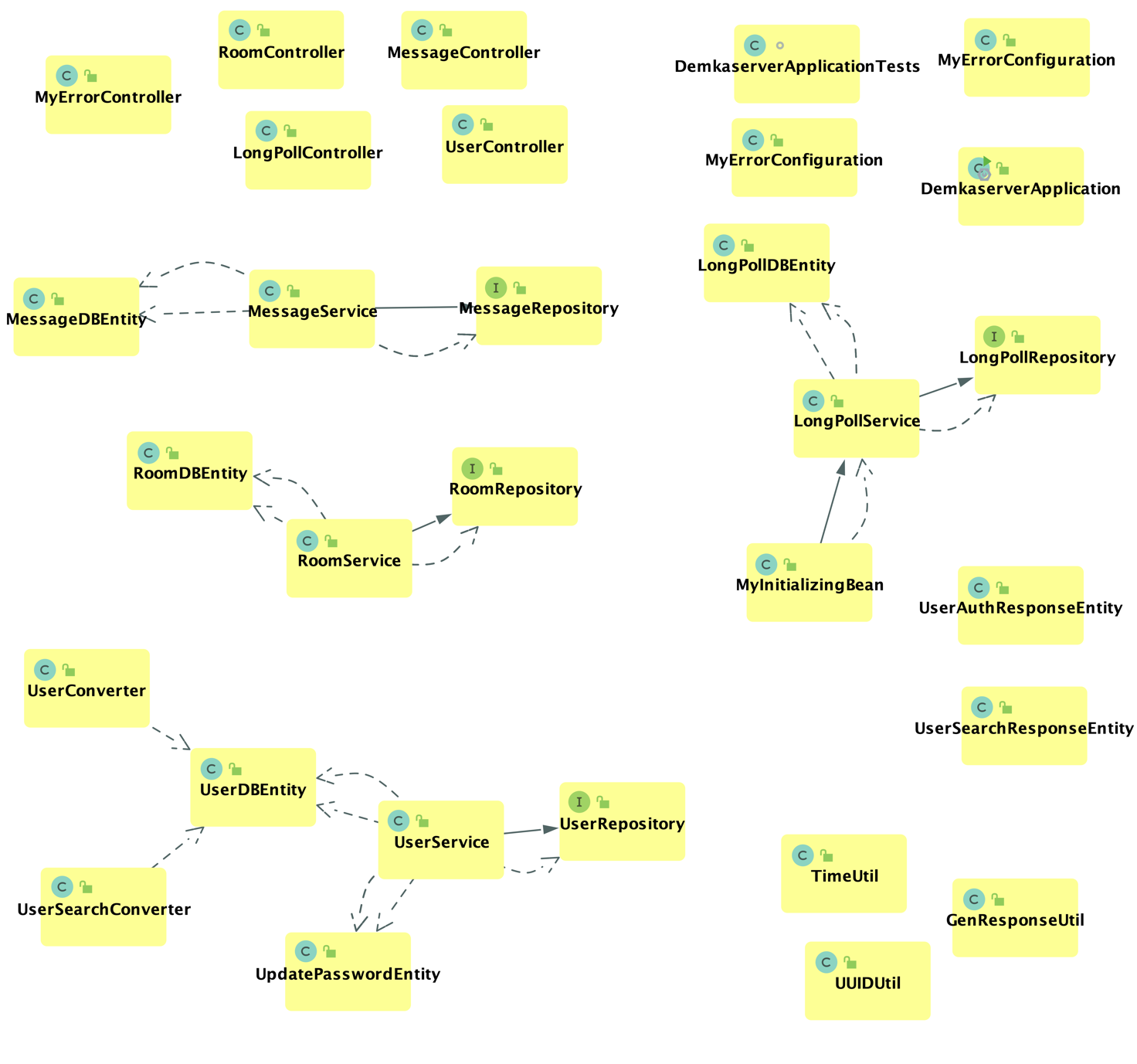
# НАЗНАЧЕНИЕ И СОСТАВ КЛАССОВ ПРОГРАММЫ

## Сервер

На сервере задействовано 30 классов для корректной работы решения. Среди них можно выделить следующие группы:

* Контроллеры (5 шт.) – классы для обработки входящих запросов от клиентов;
* Классы конфигураций и тестов (5 шт.) – необходимы для переопределения стандартной конфигурации Spring Boot в целях расширения существующего функционала;
* Классы-утилиты (3 шт.) – необходимы для обособленных решений простых задач, которые повторяются часто;
* Сущности запросов (1 шт.) – входные сущности, передаваемые клиентом для последующей обработки на стороне сервера;
* Сущности ответов (2 шт.) – выходные сущности, отдаваемые сервером для последующей обработки на стороне клиента. Отличаются от моделей СУБД тем, что не включают в себя критические данные (пароли, ключи авторизации и т.д.);
* Конвертеры сущностей (2 шт.) – необходимы для преобразования одних типов сущностей в другие;
* Сущности-документы СУБД (4 шт.) – являются представлением конкретной коллекции в базе данных для последующей работы с её элементами;
* Сервисы (4 шт.) – необходимы для обработки полученных данных с контроллеров. Являются посредниками между контроллерами и репозиториями;
* Репозитории (4 шт.) – представления коллекций (таблиц) СУБД.

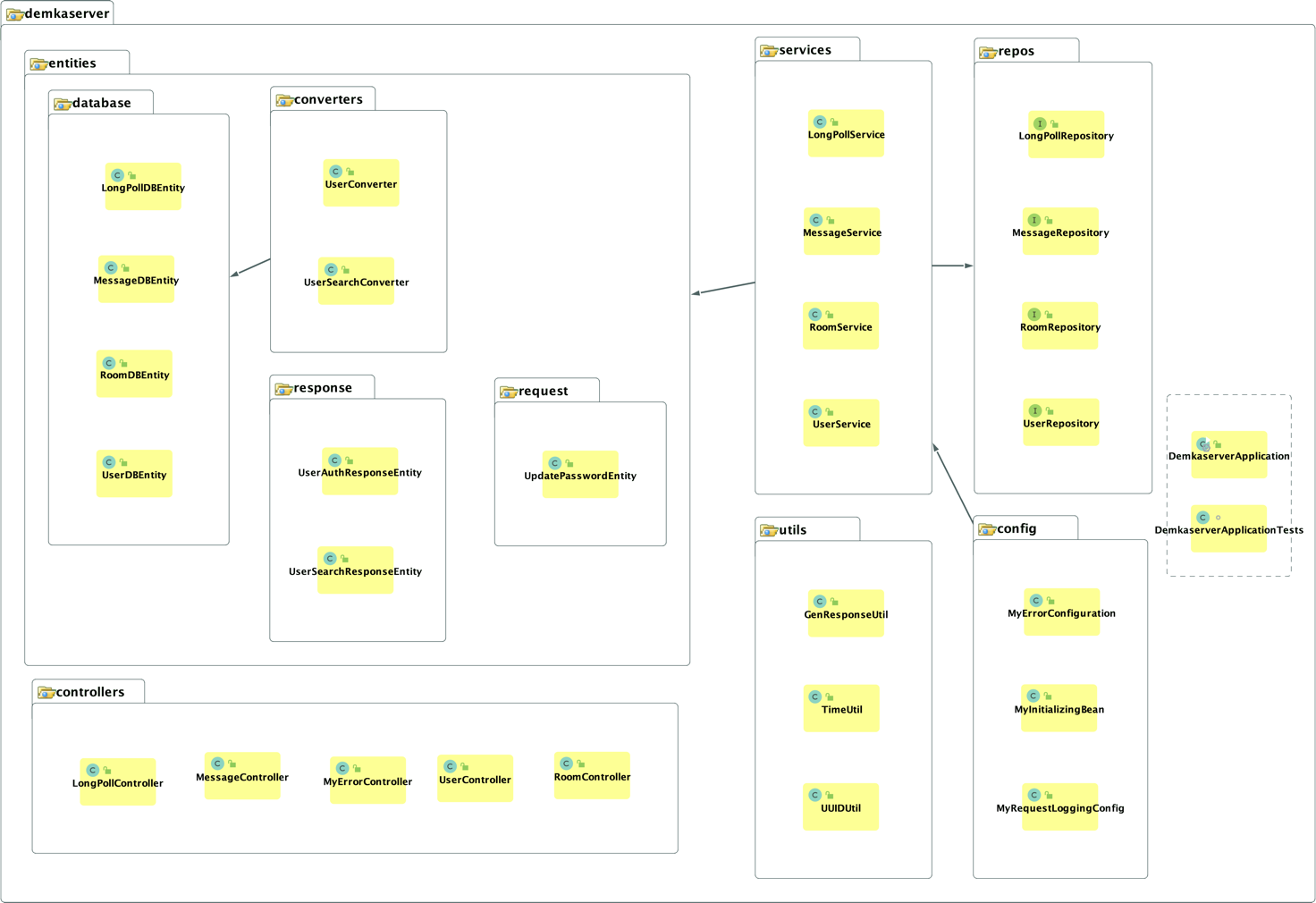
Диаграмма классов представлена на рисунке 14 ниже:



#### Рисунок – Диаграмма классов разработанного приложения-сервера

В неё входят следующие классы:

* MyErrorConfiguration – переопределение обработчика ошибок Spring Boot для использования класса ErrorController, который вводит дополнительный атрибут «result» при ошибке;
* MyInitializingBean – переопределение логики инициализации для того, чтоб удалять лонгпулы из одноименной коллекции MongoDB;
* MyRequestLoggingConfig – переопределение стандартного логгера Spring в целях более подробного логирования всех запросов, обрабатываемых сервером;
* LongPollController – контроллер для обработки http запросов, связанных с лонгпулом;
* MessageController – контроллер для обработки http запросов, связанных с сообщениями пользователя;
* MyErrorController – переопределенный контроллер ошибок, который вводит дополнительный атрибут «result» при ошибке;
* RoomController – контроллер для обработки http запросов, связанных с чат-комнатами и диалогами;
* UserController – контроллер для обработки http запросов, связанных с пользователями, в том числе для проверки авторизации и регистрации;
* UserConverter – содержит логику конвертации с UserDBEntity в UserAuthResponseEntity, который отдается при авторизации пользователя. Это необходимо для того, чтоб избавиться от полей, которые не нужны в ответе (пароль и мастер-ключ);
* UserSearchConverter – содержит логику, аналогичную UserConverter, но ещё и с исключением ключа API пользователя. Используется при поиске пользователя в системе;
* LongPollDBEntity – сущность для представления лонгпула в СУБД;
* MessageDBEntity – сущность для представления сообщения пользователя в СУБД;
* RoomDBEntity – сущность для представления чат-комнаты (диалога) в СУБД;
* UserDBEntity – сущность для представления пользователя в СУБД;
* UpdatePasswordEntity – сущность для обновления пароля пользователя. Используется при приёме данных от клиента;
* UserAuthResponseEntity – сущность для отдачи данных при авторизации пользователя. Является частным случаем UserDBEntity без полей-паролей, но с ключом API. Используется при отдаче данных клиенту;
* UserSearchResponseEntity – сущность для отдачи данных при поиске пользователя. Является частным случаем UserDBEntity без полей-паролей и без ключа API. Используется при отдаче данных клиенту;
* LongPollRepository – интерфейс MongoRepository для работы c коллекцией лонгпулов;
* MessageRepository – интерфейс MongoRepository для работы c коллекцией сообщений;
* RoomRepository – интерфейс MongoRepository для работы c коллекцией чат-комнат;
* UserRepository – интерфейс MongoRepository для работы c коллекцией пользователей;
* LongPollService – сервис для работы с лонгпулами;
* MessageService – сервис для работы с сообщениями;
* RoomService – сервис для работы с комнатами;
* UserService – сервис для работы с пользователями;
* GenResponseUtil – генерирует HashMap для успешно обработанного или некорректного запроса клиента;
* TimeUtil – возвращает текущее UNIX-время системы;
* UUIDUtil – класс для генерации идентификаторов пользователей, ключей доступа и URL для лонгпула;
* DemkaserverApplicationTests – класс для тестов;
* DemkaserverApplication – точка входа в программу.



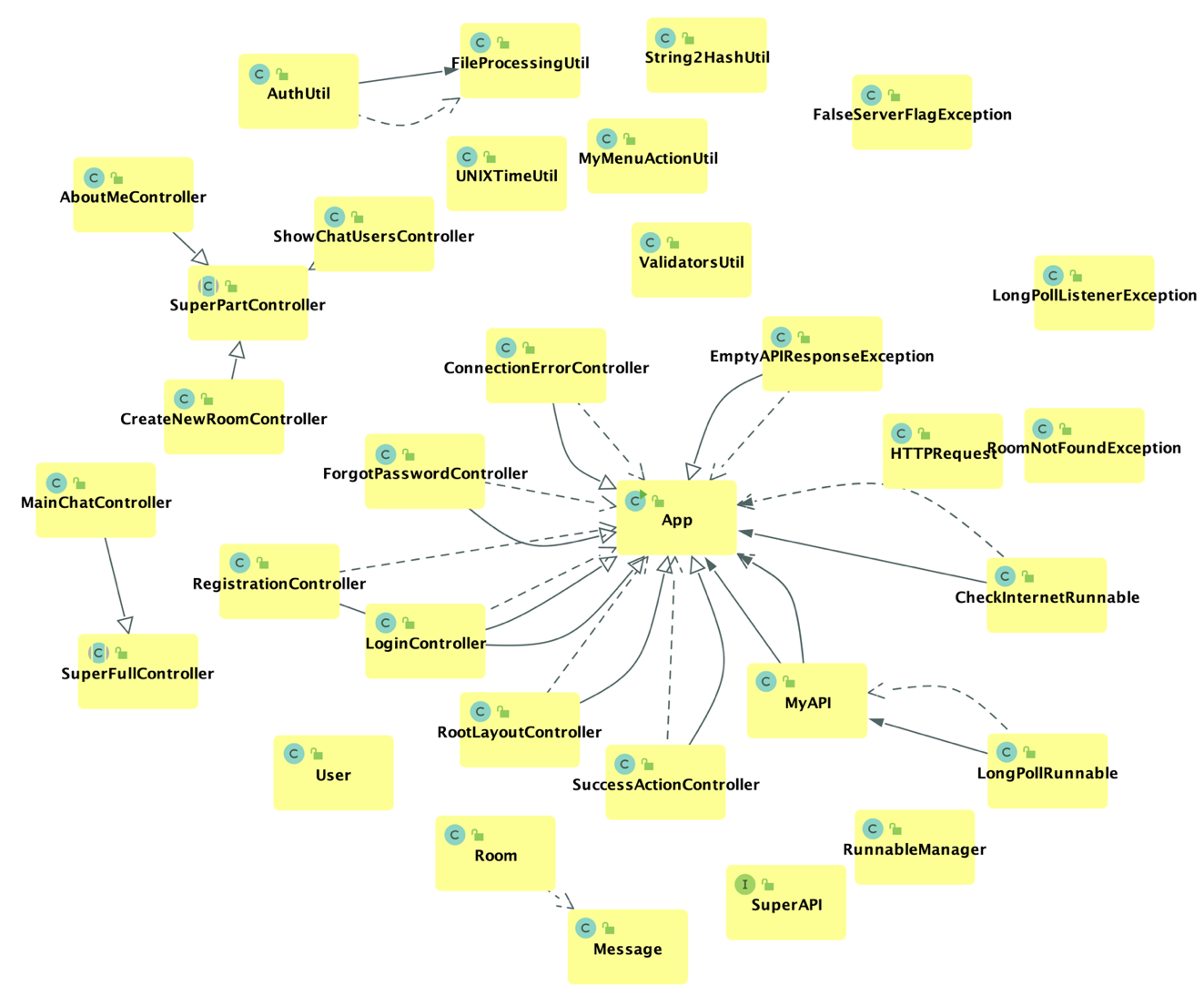
#### Рисунок – Диаграмма классов разработанного приложения-сервера с учётом иерархии модулей

## Клиент

Со стороны клиента было разработано 32 класса, которые также можно объединить в группы:

* Классы работы с API (3 шт.) – содержат логику для общения с сервером;
* Классы контроллеры (12 шт.) – отвечают за обработку действий пользователя на формах приложения;
* Классы ошибок (4 шт.) – специальные классы, наследованные от Exception. Вызываются при особых исключениях, которые возникли в программе;
* Классы сущностей (3 шт.) – модели для обработки сущностей-документов, которые определены со стороны сервера;
* Классы Runnable (3 шт.) – особые классы, наследованные от интерфейса Runnable. Запускаются в отдельном потоке, периодически возможна синхронизация с главным потоком JavaFX;
* Классы-утилиты (6 шт.) – необходимы для обособленных решений простых задач, которые повторяются часто;
* Точка входа в программу (1 шт.) – базовый класс, от которого запускается программа.

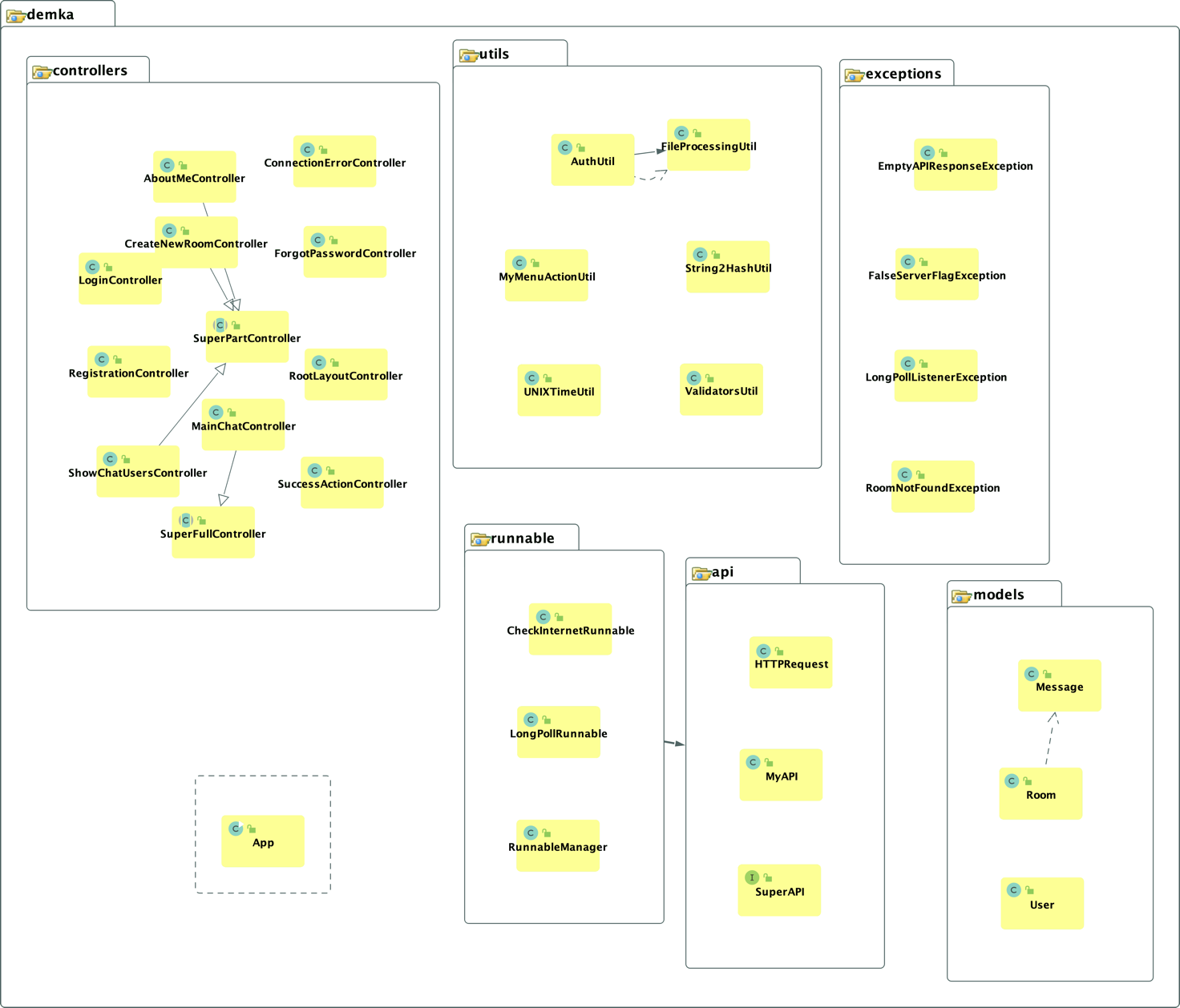
Диаграмма классов представлена на рисунке 16 ниже:



#### Рисунок – Диаграмма классов разработанного приложения-клиента

В ней содержатся следующие классы:

* HTTPRequest – класс со статическими методами для http-запросов (GET, POST, DELETE, PUT);
* SuperAPI – интерфейс с методами API, через которые клиент получает данные от сервера;
* MyAPI – класс с реализацией методов интерфейса SuperAPI;
* AboutMeController – контроллер-обработчик окна «О программе/об авторе»;
* ConnectionErrorController – контроллер, возникающий при ошибке подключения к серверу или получения от него пустого ответа;
* CreateNewRoomController – контроллер-обработчик создания новой чат-комнаты;
* ForgotPasswordController – контроллер-обработчик формы восстановления пароля;
* LoginController – контроллер-обработчик формы авторизации;
* MainChatController – контроллер-обработчик главного окна чата программы;
* RegistrationController – контроллер-обработчик регистрации пользователя;
* RootLayoutController – базовый контроллер фонового окна приложения;
* ShowChatUsersController – контроллер-обработчик окна с информацией о выбранном диалоге;
* SuccessActionController – контроллер-обработчик успешной операции пользователя;
* SuperFullController – абстрактный класс, от которого наследуются все контроллеры, layout которых открывается на всё окно;
* SuperPartController – абстрактный класс, от которого наследуются все контроллеры, layout которых открывается поверх основного окна;
* EmptyAPIResponseException – ошибка, которая возникает при пустом ответе со стороны сервера или если на устройстве отсутствует подключение к сети интернет;
* FalseServerFlagException – ошибка, возникающая при отрицательном ответе от сервера (когда флаг result равен false);
* LongPollListenerException – ошибка прослушки текущего лонгпула. При ее получении происходит получение нового адреса лонгпула;
* RoomNotFoundException – ошибка, если комната не найдена;
* Message – модель-сущность сообщения;
* Room – модель-сущность комнаты;
* User – модель-сущность пользователя;
* CheckInternetRunnable – отдельный поток для проверки интернет-соединения;
* LongPollRunnable – отдельный поток для получения сообщений через логику лонгпулинга;
* RunnableManager – класс для менеджмента потоков LongPollRunnable и CheckInternetRunnable;
* AuthUtil – класс для работы с автоавторизацией;
* FileProcessingUtil – класс для работы с файлами;
* MyMenuActionUtil – класс для обработки нажатия на Menu без Menuitem'ов;
* String2HashUtil – класс для хеширования данных в SHA-256;
* UNIXTimeUtil – класс для работы с UNIX-временем;
* ValidatorsUtil – класс валидации данных, вводимых пользователем;
* App – точка входа в программу, содержит логику перехода между окнами.



#### Рисунок – Диаграмма классов разработанного приложения-клиента с учётом иерархии модулей

# ЗАКЛЮЧЕНИЕ

В ходе написания данной работы было разработано клиент-серверное приложение для возможности обмена текстовыми сообщениями между несколькими пользователями, включающее в себя сервер с использованием библиотеки Spring Boot и клиент с применением GUI библиотеки JavaFX.

Особенностями данной работы является её своеобразный стек разработки: со стороны сервера был использован интерфейс MongoRepository для доступа к данным NoSQL СУБД MongoDB через Spring ORM, а со стороны клиента работают несколько потоков для получения обновлений сообщений пользователя с помощью механизма лонгпулинга с последующей синхронизацией через главный поток JavaFX.

Приложение полностью готово к дистрибуции: чат-сервер Spring запакован в образ docker-контейнера с помощью плагина spring-boot в Maven и загружен в репозитории Docker Hub и GitHub Packages, а образы клиентов JavaFX скомпилированы в Java Runtime Image с использованием инструмента Jlink. Все методы решения могут генерировать автоматически составляемую документацию при помощи инструмента JavaDoc.

Созданное решение удовлетворяет всем требованиям и задачам: реализует CRUD-методы со стороны бекенда, а фронт осуществляет получение и отправку данных через RESTful API с помощью протокола http с последующим вводом/выводом данных на элементы управления графического интерфейса JavaFX.

Решение может модернизироваться и обновляться. Например, можно добавить возможность отправки любых типов файлов в чат-комнаты, в том числе фото и видео. Также можно внедрить голосовые сообщения, как это сделано в большинстве популярных мессенджеров. В дальнейшем можно написать клиенты под мобильные устройства и подключить к существующему серверу.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Законодательные и нормативные акты

1. ГОСТ 2.316-2008. Правила нанесения надписей, технических требований и таблиц на графических документах.
2. ГОСТ 7.1-2003. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2004. – 169 с.
3. ГОСТ 7.32-2001. Система стандартов по информацию, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: ИПК Издательство стандартов, 2001. – 21 с.

Учебная и научная литература

1. Володченкова Л.А., Козырев Д.В. Разработка серверной части программного приложения для удаленного хранения данных // МСиМ. 2020. №1 (53).
2. Байдыбеков А.А., Гильванов Р.Г., Молодкин И.А. СОВРЕМЕННЫЕ ФРЕЙМВОРКИ ДЛЯ РАЗРАБОТКИ WEB-ПРИЛОЖЕНИЙ // Интеллектуальные технологии на транспорте. 2020. №4 (24).
3. Гасанов Заурбек Зубаирович Анализ производительности многопоточных программ, написанных на языках Java и Go // Наука и образование сегодня. 2018. №6 (29).
4. Барабанов В.Ф., Донских А.К., Гребенникова Н.И., Кенин С.Л. ПОЛУЧЕНИЕ МЕТРИК JAVA-ПРИЛОЖЕНИЯ В КОНТЕЙНЕРАХ DOCKER // Вестник ВГТУ. 2020. №2.
5. О.А. Ляшенко, О.О. Конашков, Н.А. Солодкая СРАВНИТЕЛЬНЫЙ АНАЛИЗ ВЫПОЛНЕНИЯ ЗАПРОСОВ К СЕРВЕРАМ БАЗ ДАННЫХ MYSQL И MONGODB // Вестник Херсонского национального технического университета. 2019. №4 (71).

# ПРИЛОЖЕНИЯ

## Приложение 1. Исходный код для развертки решения

docker-compose.yml

version: '3.1'

services:

mongo:

image: mongo

restart: on-failure

ports:

- 27017:27017

environment:

MONGO\_INITDB\_ROOT\_USERNAME: ${MONGO\_LOGIN}

MONGO\_INITDB\_ROOT\_PASSWORD: ${MONGO\_PASSWD}

volumes:

- ./database/mongodbdata:/data/db

spring:

image: georgiydemo/spring-mychat-server

restart: on-failure

ports:

- 8080:8080

environment:

MONGO\_DB: ${MONGO\_DB}

MONGO\_URL: ${MONGO\_URL}

depends\_on:

- mongo

## Приложение 2. Исходный код сервера

pom.xml

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.4.4</version>  
 <relativePath/> *<!-- lookup parent from repository -->* </parent>  
 <groupId>com.demka</groupId>  
 <artifactId>demkaserver</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>demkaserver</name>  
 <description>Spring Boot Chat server</description>  
 <properties>  
 <java.version>16</java.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-mongodb</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 <configuration>  
 <excludes>  
 <exclude>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 </exclude>  
 </excludes>  
 </configuration>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <configuration>  
 <source>15</source>  
 <target>15</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

demkaserver/src/main/resources/application.properties

logging.level.org.springframework.web.filter.CommonsRequestLoggingFilter=DEBUG  
spring.data.mongodb.database=${MONGO\_DB}  
spring.data.mongodb.uri=${MONGO\_URL}

com/demka/demkaserver/DemkaserverApplication.java

package com.demka.demkaserver;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
*/\*\*  
 \* The type Demkaserver application.  
 \*/*@SpringBootApplication  
public class DemkaserverApplication {  
  
 */\*\*  
 \* The entry point of application.  
 \*  
 \* @param args the input arguments  
 \*/* public static void main(String[] args) {  
 SpringApplication.*run*(DemkaserverApplication.class, args);  
 }  
  
}

com/demka/demkaserver/config/MyErrorConfiguration.java

package com.demka.demkaserver.config;  
  
import com.demka.demkaserver.controllers.MyErrorController;  
import org.springframework.beans.factory.ObjectProvider;  
import org.springframework.boot.autoconfigure.web.ServerProperties;  
import org.springframework.boot.autoconfigure.web.servlet.error.ErrorViewResolver;  
import org.springframework.boot.web.servlet.error.ErrorAttributes;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
  
import java.util.List;  
  
*/\*\*  
 \* The type My error configuration.  
 \*/*@Configuration  
public class MyErrorConfiguration {  
  
 */\*\*  
 \* Нужен, чтоб кастомный ErrorController использовать  
 \* А он в свою очередь нужен, чтоб доп атрибут result при ошибке выводить  
 \*  
 \* @param errorAttributes the error attributes  
 \* @param serverProperties the server properties  
 \* @param errorViewResolversProvider the error view resolvers provider  
 \* @return the my error controller  
 \*/* @Bean  
 public MyErrorController ErrorController(ErrorAttributes errorAttributes, ServerProperties serverProperties, ObjectProvider<List<ErrorViewResolver>> errorViewResolversProvider) {  
 return new MyErrorController(errorAttributes, serverProperties.getError(), errorViewResolversProvider.getIfAvailable());  
 }  
}

com/demka/demkaserver/config/MyInitializingBean.java

package com.demka.demkaserver.config;  
  
import com.demka.demkaserver.services.LongPollService;  
import org.springframework.beans.factory.InitializingBean;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
  
*/\*\*  
 \* The type My initializing bean.  
 \*/*@Component  
public class MyInitializingBean implements InitializingBean {  
  
 private final LongPollService longPollService;  
  
 */\*\*  
 \* Instantiates a new My initializing bean.  
 \*  
 \* @param longPollService the long poll service  
 \*/* @Autowired  
 public MyInitializingBean(LongPollService longPollService) {  
 this.longPollService = longPollService;  
 }  
  
 */\*\*  
 \* Переопределение нужно для того, чтоб удалять данные старых лонгпулов с СУБД  
 \* После реинициализации спринга  
 \*/* @Override  
 public void afterPropertiesSet() {  
 longPollService.deleteAll();  
 }  
}

com/demka/demkaserver/config/MyRequestLoggingConfig.java

package com.demka.demkaserver.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.web.filter.CommonsRequestLoggingFilter;  
  
*/\*\*  
 \* The type My request logging config.  
 \*/*@Configuration  
public class MyRequestLoggingConfig {  
  
 */\*\*  
 \* Нужно для того, чтоб было доп логирование всех запросов,  
 \* которые приходят на сервер  
 \*  
 \* @return the commons request logging filter  
 \*/* @Bean  
 public CommonsRequestLoggingFilter logFilter() {  
 CommonsRequestLoggingFilter filter = new CommonsRequestLoggingFilter();  
 filter.setIncludeQueryString(true);  
 filter.setIncludePayload(true);  
 filter.setMaxPayloadLength(10000);  
 filter.setIncludeHeaders(false);  
 filter.setAfterMessagePrefix("REQUEST DATA: ");  
 return filter;  
 }  
}

com/demka/demkaserver/controllers/LongPollController.java

package com.demka.demkaserver.controllers;  
  
import com.demka.demkaserver.entities.database.LongPollDBEntity;  
import com.demka.demkaserver.entities.database.MessageDBEntity;  
import com.demka.demkaserver.entities.database.RoomDBEntity;  
import com.demka.demkaserver.entities.database.UserDBEntity;  
import com.demka.demkaserver.services.LongPollService;  
import com.demka.demkaserver.services.MessageService;  
import com.demka.demkaserver.services.RoomService;  
import com.demka.demkaserver.services.UserService;  
import com.demka.demkaserver.utils.GenResponseUtil;  
import com.demka.demkaserver.utils.TimeUtil;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.time.Duration;  
import java.time.Instant;  
import java.util.\*;  
  
*/\*\*  
 \* The type Long poll controller.  
 \*/*@RestController  
@RequestMapping("/longpoll")  
public class LongPollController {  
  
 private final MessageService messageService;  
 private final UserService userService;  
 private final LongPollService longPollService;  
 private final RoomService roomService;  
  
  
 */\*\*  
 \* Instantiates a new Long poll controller.  
 \*  
 \* @param messageService the message service  
 \* @param userService the user service  
 \* @param longPollService the long poll service  
 \* @param roomService the room service  
 \*/* @Autowired  
 public LongPollController(MessageService messageService, UserService userService, LongPollService longPollService, RoomService roomService) {  
 this.messageService = messageService;  
 this.userService = userService;  
 this.longPollService = longPollService;  
 this.roomService = roomService;  
 }  
  
 */\*\*  
 \* Получение данных лонгпула для указанного ключа  
 \*  
 \* @param key - ключ пользователя  
 \* @return long poll server  
 \*/* @GetMapping(value = "/getServer")  
 public ResponseEntity<Map<String, Object>> getLongPollServer(@RequestParam String key) {  
  
 *//Получаем объект пользователя через key + проверка ключа* Optional<UserDBEntity> userOptional = userService.findByKey(key);  
 if (userOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
 }  
  
 UserDBEntity user = userOptional.get();  
 String userId = user.getId();  
  
 *//Находим комнаты пользователя* List<RoomDBEntity> roomsList = roomService.findUserRooms(userId);  
 List<MessageDBEntity> messagesList = messageService.getAllMessagesByRooms(roomsList);  
  
 *//Если нет сообщений пользователя, то возвращаем текущее время* if (messagesList.size() == 0) {  
 Long ts = TimeUtil.*unixTime*();  
 LongPollDBEntity newPoll = longPollService.create(userId, ts);  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(newPoll), HttpStatus.*OK*);  
 }  
  
 Comparator<MessageDBEntity> bufComparator = Comparator.*comparing*(MessageDBEntity::getTimeCreated);  
 Comparator<MessageDBEntity> MyComparator = bufComparator.reversed();  
 messagesList.sort(MyComparator);  
  
 Long ts = messagesList.get(0).getTimeCreated();  
 LongPollDBEntity newPoll = longPollService.create(userId, ts);  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(newPoll), HttpStatus.*OK*);  
 }  
  
  
 */\*\*  
 \* Получение данных для лонгпула по определенному ключу  
 \*  
 \* @param url - URL лонгпула  
 \* @param key - ключ ЛОНГПУЛА (не пользователя)  
 \* @param ts - значение ts, пполученное через /getServer  
 \* @return response entity  
 \* @throws InterruptedException the interrupted exception  
 \*/* @GetMapping(value = "/updates/{url}")  
 public ResponseEntity<Map<String, Object>> longPoll(@PathVariable String url, @RequestParam String key, @RequestParam Long ts) throws InterruptedException {  
  
 *//Проверка на существование url* if (longPollService.findByUrl(url).isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Указанного url не существует"), HttpStatus.*NOT\_FOUND*);  
 }  
  
 *//Проверка на существование ключа + url* Optional<LongPollDBEntity> currentPollOptional = longPollService.findByKeyAndUrl(key, url);  
 if (currentPollOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
 }  
  
 LongPollDBEntity currentPoll = currentPollOptional.get();  
  
 *//Время начала работы в цикле (чтоб ловить таймауты)* Instant timeStarted = Instant.*now*();  
 Long newTs;  
 List<MessageDBEntity> newMessages;  
  
 while (true) {  
  
 *//Находим комнаты пользователя* List<RoomDBEntity> roomsList = roomService.findUserRooms(currentPoll.getUserId());  
 *//Находим новые сообщения по каждой комнате* newMessages = messageService.getNewMessagesByRooms(roomsList, ts);  
 *//Сортируем по новизне* Comparator<MessageDBEntity> bufComparator = Comparator.*comparing*(MessageDBEntity::getTimeCreated);  
 Comparator<MessageDBEntity> MyComparator = bufComparator.reversed();  
  
 *//Если есть какие-то обновления, то возвращаем эти обновления* if (newMessages.size() != 0) {  
 newMessages.sort(MyComparator);  
 newTs = newMessages.get(0).getTimeCreated();  
 break;  
 }  
  
 *//Если время кручения в цикле более одного часа - надо бы сделать переавторизацию клиента* if (Duration.*between*(timeStarted, Instant.*now*()).toMinutes() > 60) {  
 *//Удаляем лонгпул с СУБД* longPollService.delete(currentPoll.getId());  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Необходимо обновить лонгпул через /getServer"), HttpStatus.*OK*);  
 }  
  
 *//Иначе просто крутимся в этом вечном цикле* Thread.*sleep*(500);  
 }  
  
 HashMap<String, Object> bodyMap = new HashMap<>();  
 bodyMap.put("updates", newMessages);  
 bodyMap.put("ts", newTs);  
  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(bodyMap), HttpStatus.*OK*);  
  
 }  
  
}

com/demka/demkaserver/controllers/MessageController.java

package com.demka.demkaserver.controllers;  
  
import com.demka.demkaserver.entities.database.MessageDBEntity;  
import com.demka.demkaserver.entities.database.RoomDBEntity;  
import com.demka.demkaserver.entities.database.UserDBEntity;  
import com.demka.demkaserver.services.MessageService;  
import com.demka.demkaserver.services.RoomService;  
import com.demka.demkaserver.services.UserService;  
import com.demka.demkaserver.utils.GenResponseUtil;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.Map;  
import java.util.Optional;  
  
*/\*\*  
 \* The type Message controller.  
 \*/*@RestController  
@RequestMapping("/messages")  
public class MessageController {  
  
 private final RoomService roomService;  
 private final UserService userService;  
 private final MessageService messageService;  
  
 */\*\*  
 \* Instantiates a new Message controller.  
 \*  
 \* @param roomService the room service  
 \* @param userService the user service  
 \* @param messageService the message service  
 \*/* @Autowired  
 public MessageController(RoomService roomService, UserService userService, MessageService messageService) {  
 this.roomService = roomService;  
 this.userService = userService;  
 this.messageService = messageService;  
 }  
  
 */\*\*  
 \* Отправка сообщения в опеределенную комнату roomId  
 \*  
 \* @param data - данные в JSON. Поля: key - ключ пользователя text - текст сообщения roomId - id комнаты, куда отправляется сообщение  
 \* @return response entity  
 \*/* @PostMapping(value = "/send")  
 public ResponseEntity<Map<String, Object>> sendMessage(@RequestBody Map<String, String> data) {  
  
 String key = data.get("key");  
 String messageText = data.get("text");  
 String roomId = data.get("roomId");  
  
 *//Проверка переданных полей* if ((key == null) || (messageText == null) || (roomId == null)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не все значения были переданы"), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 *//Получаем объект пользователя через key + проверка ключа* Optional<UserDBEntity> roomUserOptional = userService.findByKey(key);  
 if (roomUserOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
 }  
  
 *//Проверка на комнату* Optional<RoomDBEntity> currentRoomOptional = roomService.find(roomId);  
 if (currentRoomOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Комнаты с id " + roomId + " не существует"), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 RoomDBEntity currentRoom = currentRoomOptional.get();  
 UserDBEntity roomUser = roomUserOptional.get();  
  
 *//Проверка на то, может ли пользователь вообще писать в эту комнату* if (!currentRoom.getUsers().contains(roomUser.getId())) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("У вас нет прав на отправку сообщения в эту комнату"), HttpStatus.*FORBIDDEN*);  
 }  
  
 MessageDBEntity newMessage = messageService.create(roomUser.getId(), roomUser.getName(), messageText, roomId);  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(newMessage), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Получение истории сообщений определенной комнаты  
 \*  
 \* @param key - ключ пользователя  
 \* @param roomId - id комнаты  
 \* @return messages by room  
 \*/* @GetMapping(value = "/get")  
 public ResponseEntity<Map<String, Object>> getMessagesByRoom(@RequestParam String key, @RequestParam String roomId) {  
  
 *//Проверка ключа* Optional<UserDBEntity> currentUserOptional = userService.findByKey(key);  
 if (currentUserOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
 }  
 UserDBEntity currentUser = currentUserOptional.get();  
  
 *//Проверка на существование комнаты* Optional<RoomDBEntity> currentRoomOptional = roomService.find(roomId);  
 if (currentRoomOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Комнаты с id " + roomId + " не существует"), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 RoomDBEntity currentRoom = currentRoomOptional.get();  
 *//Проверка на то, имеет ли пользователь право на чтение сообщений из этой комнаты* if (!currentRoom.getUsers().contains(currentUser.getId())) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("У вас нет прав на чтение сообщений из этой комнаты"), HttpStatus.*FORBIDDEN*);  
 }  
  
 *//Если комната существует и все фильтрации пройдены* List<MessageDBEntity> messagesList = messageService.findByRoom(roomId);  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(messagesList), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Удаление сообщения.  
 \* Сообщение может удалять либо сам пользователь, либо создатель комнаты  
 \*  
 \* @param data - данные в JSON. Поля: messageId - идентификатор сообщения для удаления key - ключ API  
 \* @return response entity  
 \*/* @DeleteMapping(value = "/remove")  
 public ResponseEntity<Map<String, Object>> removeMessage(@RequestBody Map<String, String> data) {  
  
 String key = data.get("key");  
 String messageId = data.get("messageId");  
  
 if ((key == null) || (messageId == null)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не все значения были переданы"), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 *//Проверка на авторизацию API* Optional<UserDBEntity> currentUserOptional = userService.findByKey(key);  
 if (currentUserOptional.isEmpty())  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
 UserDBEntity currentUser = currentUserOptional.get();  
  
 *//Проверка на существование сообщения* Optional<MessageDBEntity> currentMessageOptional = messageService.find(messageId);  
 if (currentMessageOptional.isEmpty())  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Сообщения с id " + messageId + " не существует"), HttpStatus.*BAD\_REQUEST*);  
  
 MessageDBEntity currentMessage = currentMessageOptional.get();  
 *//Проверка на возможность удаления сообщения* Optional<RoomDBEntity> currentMessageRoomOptional = roomService.find(currentMessage.getRoomId());  
  
 *//Удаление от имени создателя беседы* if (currentMessageRoomOptional.isPresent() && (currentUser.getId().equals(currentMessageRoomOptional.get().getCreatorId()))) {  
 messageService.delete(currentMessage);  
 System.*out*.println("Произошло удаление от имени создателя беседы");  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*("Успешное удаление сообщения " + messageId), HttpStatus.*OK*);  
 *//Удаление от имени владельца сообщения* } else if (currentMessage.getUserId().equals(currentUser.getId())) {  
 messageService.delete(currentMessage);  
 System.*out*.println("Произошло удаление от имени владельца сообщения");  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*("Успешное удаление сообщения " + messageId), HttpStatus.*OK*);  
 *//Нет прав на удаление* } else {  
 System.*out*.println("Нет прав на удаление сообщения");  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("У вас нет права на удаление сообщения " + messageId), HttpStatus.*FORBIDDEN*);  
 }  
 }  
}

com/demka/demkaserver/controllers/MyErrorController.java

package com.demka.demkaserver.controllers;  
  
import org.springframework.boot.autoconfigure.web.ErrorProperties;  
import org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController;  
import org.springframework.boot.autoconfigure.web.servlet.error.ErrorViewResolver;  
import org.springframework.boot.web.error.ErrorAttributeOptions;  
import org.springframework.boot.web.servlet.error.ErrorAttributes;  
  
import javax.servlet.http.HttpServletRequest;  
import java.util.List;  
import java.util.Map;  
  
*/\*\*  
 \* The type My error controller.  
 \*/*public class MyErrorController extends BasicErrorController {  
  
 */\*\*  
 \* Instantiates a new My error controller.  
 \*  
 \* @param errorAttributes the error attributes  
 \* @param errorProperties the error properties  
 \* @param errorViewResolvers the error view resolvers  
 \*/* public MyErrorController(ErrorAttributes errorAttributes, ErrorProperties errorProperties, List<ErrorViewResolver> errorViewResolvers) {  
 super(errorAttributes, errorProperties, errorViewResolvers);  
 }  
  
  
 */\*\*  
 \* Переопределение полей, которые отдаются при ошибке обработки запроса  
 \*  
 \* @param request  
 \* @param options  
 \* @return  
 \*/* @Override  
 protected Map<String, Object> getErrorAttributes(HttpServletRequest request, ErrorAttributeOptions options) {  
 Map<String, Object> errorAttributes = super.getErrorAttributes(request, options);  
 errorAttributes.put("result", false);  
 return errorAttributes;  
 }  
  
}

com/demka/demkaserver/controllers/RoomController.java

package com.demka.demkaserver.controllers;  
  
import com.demka.demkaserver.entities.converters.UserSearchConverter;  
import com.demka.demkaserver.entities.database.RoomDBEntity;  
import com.demka.demkaserver.entities.database.UserDBEntity;  
import com.demka.demkaserver.entities.response.UserSearchResponseEntity;  
import com.demka.demkaserver.services.MessageService;  
import com.demka.demkaserver.services.RoomService;  
import com.demka.demkaserver.services.UserService;  
import com.demka.demkaserver.utils.GenResponseUtil;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.\*;  
  
*/\*\*  
 \* The type Room controller.  
 \*/*@RestController  
@RequestMapping("/room")  
public class RoomController {  
  
 private final RoomService roomService;  
 private final UserService userService;  
 private final MessageService messageService;  
  
 */\*\*  
 \* Instantiates a new Room controller.  
 \*  
 \* @param roomService the room service  
 \* @param userService the user service  
 \* @param messageService the message service  
 \*/* @Autowired  
 public RoomController(RoomService roomService, UserService userService, MessageService messageService) {  
 this.roomService = roomService;  
 this.userService = userService;  
 this.messageService = messageService;  
 }  
  
 */\*\*  
 \* Создание комнаты для общения пользователей  
 \*  
 \* @param data - данные в JSON. Поля: key - ключ пользователя roomName - название комнаты users - строка с id пользователей, состоящих в беседе  
 \* @return response entity  
 \*/* @PostMapping("/create")  
 public ResponseEntity<Map<String, Object>> createRoom(@RequestBody Map<String, String> data) {  
  
 String key = data.get("key");  
 String roomName = data.get("roomName");  
 String usersString = data.get("users");  
  
 *//Проверка переданных полей* if ((key == null) || (roomName == null) || (usersString == null)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не все значения были переданы"), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 List<String> users = Arrays.*asList*(usersString.split(","));  
  
 *//Получаем id создателя через key + проверка ключа* Optional<UserDBEntity> creatorUserOptional = userService.findByKey(key);  
 *//Если вдруг пользователь с key не найден* if (creatorUserOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
 }  
  
 *//Проверка на существование пользователей* for (String userId : users) {  
 if (userService.find(userId).isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Пользователя с id " + userId + " не существует"), HttpStatus.*BAD\_REQUEST*);  
 }  
 }  
  
 UserDBEntity creatorUser = creatorUserOptional.get();  
 String creatorId = creatorUser.getId();  
 RoomDBEntity newRoom = roomService.create(creatorId, roomName, users);  
  
 *//Формируем сообщение, что комната создана* String messageText = "\*Комната была создана\*";  
 messageService.create(creatorId, creatorUser.getName(), messageText, newRoom.getId());  
  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(newRoom), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Получение инфорации о комнате  
 \*  
 \* @param key - ключ пользователя  
 \* @param roomId - id комнаты  
 \* @return room info  
 \*/* @GetMapping(value = "/get")  
 public ResponseEntity<Map<String, Object>> getRoomInfo(@RequestParam String key, @RequestParam String roomId) {  
  
  
 *//Проверка ключа* if (!userService.checkUserKey(key)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
 }  
  
 Optional<RoomDBEntity> roomResult = roomService.find(roomId);  
 *//Если комнаты с таким id не существует* if (roomResult.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Комнаты с id " + roomId + " не существует"), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 *//Если существует* return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(roomResult.get()), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Удаление пользователя из комнаты  
 \*  
 \* @param data - данные в JSON. Поля: roomId - идентификатор комнаты, с которой работаем userId - идентификатор пользователя, которого удаляем key - ключ API создателя беседы  
 \* @return response entity  
 \*/* @PostMapping(value = "/removeUser")  
 public ResponseEntity<Map<String, Object>> removeUser(@RequestBody Map<String, String> data) {  
 String roomId = data.get("roomId");  
 String userIdToRemove = data.get("userId");  
 String key = data.get("key");  
  
 *//Если клиент передал не все значения* if ((roomId == null) || (userIdToRemove == null) || (key == null)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не все значения были переданы"), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 *//Проверка на авторизацию* Optional<UserDBEntity> creatorUserOptional = userService.findByKey(key);  
 if (creatorUserOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
 }  
 UserDBEntity currentUser = creatorUserOptional.get();  
  
 Optional<RoomDBEntity> currentRoomOptional = roomService.find(roomId);  
 *//Проверка на существование комнаты* if (currentRoomOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Комнаты с id " + roomId + " не существует"), HttpStatus.*BAD\_REQUEST*);  
 }  
 RoomDBEntity currentRoom = currentRoomOptional.get();  
  
 *//Проверка, что пользователь - создатель комнаты* if (!currentRoom.getCreatorId().equals(currentUser.getId())) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("У вас нет права на удаление пользователей из комнаты " + roomId), HttpStatus.*FORBIDDEN*);  
 }  
  
 *//Проверка, что удаляем не создателя* if (currentRoom.getCreatorId().equals(userIdToRemove)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Вы не можете удалить самого себя (создателя) из комнаты"), HttpStatus.*FORBIDDEN*);  
 }  
  
 *//Проверка на сущестование этого человека в комнате для его последующего удаления* if (!currentRoom.getUsers().contains(userIdToRemove)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Пользователя " + userIdToRemove + " не существует в комнате " + roomId), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 roomService.removeUser(currentRoom, userIdToRemove);  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(currentRoom), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Добавление пользователя в комнату  
 \*  
 \* @param data - данные в JSON. Поля: roomId - идентификатор комнаты, с которой работаем userId - идентификатор пользователя, которого добавляем key - ключ API создателя беседы  
 \* @return response entity  
 \*/* @PostMapping(value = "/addUser")  
 public ResponseEntity<Map<String, Object>> addUser(@RequestBody Map<String, String> data) {  
 String roomId = data.get("roomId");  
 String userIdToAdd = data.get("userId");  
 String key = data.get("key");  
  
 *//Если клиент передал не все значения* if ((roomId == null) || (userIdToAdd == null) || (key == null)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не все значения были переданы"), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 *//Проверка на авторизацию* Optional<UserDBEntity> creatorUserOptional = userService.findByKey(key);  
 if (creatorUserOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
 }  
 UserDBEntity currentUser = creatorUserOptional.get();  
  
 *//Проверка на существование комнаты* Optional<RoomDBEntity> currentRoomOptional = roomService.find(roomId);  
 if (currentRoomOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Комнаты с id " + roomId + " не существует"), HttpStatus.*BAD\_REQUEST*);  
 }  
 RoomDBEntity currentRoom = currentRoomOptional.get();  
  
 *//Проверка, что пользователь - создатель комнаты* if (!currentRoom.getCreatorId().equals(currentUser.getId())) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("У вас нет права на добавление пользователей в комнату " + roomId), HttpStatus.*FORBIDDEN*);  
 }  
  
 *//Проверка на существование этого человека уже в комнате* if (currentRoom.getUsers().contains(userIdToAdd)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Пользователь " + userIdToAdd + " уже есть в комнате " + roomId), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 *//Проверка на существование идентификатора пользователя для добавления в комнату* Optional<UserDBEntity> userToAddOptional = userService.find(userIdToAdd);  
 if (userToAddOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Пользователя с id " + userIdToAdd + " не существует"), HttpStatus.*BAD\_REQUEST*);  
 }  
 UserDBEntity userToAdd = userToAddOptional.get();  
 roomService.addUser(currentRoom, userToAdd);  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(currentRoom), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Получение всех комнат, в которых состоит пользователь  
 \*  
 \* @param key - ключ пользователя  
 \* @return by user  
 \*/* @GetMapping(value = "/getByUser")  
 public ResponseEntity<Map<String, Object>> getByUser(@RequestParam String key) {  
  
 *//Проверка ключа* Optional<UserDBEntity> currentUserOptional = userService.findByKey(key);  
 if (currentUserOptional.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
 }  
  
 UserDBEntity currentUser = currentUserOptional.get();  
 List<RoomDBEntity> userRooms = roomService.findUserRooms(currentUser.getId());  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(userRooms), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Получение пользователей, которые состоят в комнате  
 \*  
 \* @param key - ключ пользователя  
 \* @param roomId - идентификатор чат-комнаты  
 \* @return room users  
 \*/* @GetMapping(value = "/getUsers")  
 public ResponseEntity<Map<String, Object>> getRoomUsers(@RequestParam String key, @RequestParam String roomId) {  
  
 *//Проверка ключа* if (!userService.checkUserKey(key)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
 }  
  
 Optional<RoomDBEntity> roomResult = roomService.find(roomId);  
 *//Если комнаты с таким id не существует* if (roomResult.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Комнаты с id " + roomId + " не существует"), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 RoomDBEntity currentRoom = roomResult.get();  
 List<UserSearchResponseEntity> roomUsersList = new ArrayList<>();  
 for (String userId : currentRoom.getUsers()) {  
 Optional<UserDBEntity> currentUserOptional = userService.find(userId);  
  
 *//Если пользователь найден - добавляем в список результатов* if (currentUserOptional.isPresent()) {  
 UserDBEntity currentUser = currentUserOptional.get();  
 roomUsersList.add(UserSearchConverter.*convert*(currentUser));  
 }  
 }  
  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(roomUsersList), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Удаление комнаты  
 \*  
 \* @param data - данные в JSON. Поля: roomId - идентификатор комнаты для удаления key - ключ API создателя беседы  
 \* @return response entity  
 \*/* @DeleteMapping(value = "/remove")  
 public ResponseEntity<Map<String, Object>> removeRoom(@RequestBody Map<String, String> data) {  
 String roomId = data.get("roomId");  
 String key = data.get("key");  
  
 if ((roomId == null) || (key == null)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не все значения были переданы"), HttpStatus.*BAD\_REQUEST*);  
 }  
  
 *//Проверка на авторизацию* Optional<UserDBEntity> currentUserOptional = userService.findByKey(key);  
 if (currentUserOptional.isEmpty())  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
  
 *//Проврека на существование комнаты* Optional<RoomDBEntity> currentRoomOptional = roomService.find(roomId);  
 if (currentRoomOptional.isEmpty())  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Комнаты с id " + roomId + " не существует"), HttpStatus.*BAD\_REQUEST*);  
  
 *//Проверка на возможность удаления комнаты* RoomDBEntity currentRoom = currentRoomOptional.get();  
 UserDBEntity currentUser = currentUserOptional.get();  
 if (!currentUser.getId().equals(currentRoom.getCreatorId()))  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("У вас нет права на удаление комнаты " + roomId), HttpStatus.*FORBIDDEN*);  
  
 *//Если все ок* roomService.delete(currentRoom);  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*("Успешное удаление комнаты " + roomId), HttpStatus.*OK*);  
 }  
}

com/demka/demkaserver/controllers/UserController.java

package com.demka.demkaserver.controllers;  
  
import com.demka.demkaserver.entities.converters.UserConverter;  
import com.demka.demkaserver.entities.converters.UserSearchConverter;  
import com.demka.demkaserver.entities.database.UserDBEntity;  
import com.demka.demkaserver.entities.request.UpdatePasswordEntity;  
import com.demka.demkaserver.entities.response.UserAuthResponseEntity;  
import com.demka.demkaserver.entities.response.UserSearchResponseEntity;  
import com.demka.demkaserver.services.UserService;  
import com.demka.demkaserver.utils.GenResponseUtil;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.\*;  
  
  
*/\*\*  
 \* The type User controller.  
 \*/*@RestController  
@RequestMapping("/user")  
public class UserController {  
  
 private final UserService userService;  
  
 */\*\*  
 \* Instantiates a new User controller.  
 \*  
 \* @param userService the user service  
 \*/* @Autowired  
 public UserController(UserService userService) {  
 this.userService = userService;  
 }  
  
 */\*\*  
 \* Авторизация пользователя через пару логина/пароля  
 \*  
 \* @param login - логин пользователя  
 \* @param password - пароль пользователя  
 \* @return response entity  
 \*/* @GetMapping(value = "/auth", params = {"login", "password"})  
 public ResponseEntity<Map<String, Object>> auth(@RequestParam String login, @RequestParam String password) {  
  
 UserDBEntity result = userService.checkAuth(login, password);  
  
 if (result != null) {  
 UserAuthResponseEntity user = UserConverter.*convert*(result);  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(user), HttpStatus.*OK*);  
 }  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось войти с указанным логином/паролем"), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Авторизация пользователя через ключ API  
 \*  
 \* @param key - ключ API  
 \* @return response entity  
 \*/* @GetMapping(value = "/auth", params = {"key"})  
 public ResponseEntity<Map<String, Object>> auth(@RequestParam String key) {  
  
 UserDBEntity result = userService.checkAuth(key);  
 if (result != null) {  
 UserAuthResponseEntity user = UserConverter.*convert*(result);  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(user), HttpStatus.*OK*);  
 }  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось войти с указанным ключем API"), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Регистрация пользователя  
 \*  
 \* @param data - данные в JSON. Поля: login - логин пользователя password - пароль пользователя username - ник пользователя  
 \* @return response entity  
 \*/* @PostMapping("/register")  
 public ResponseEntity<Map<String, Object>> regUser(@RequestBody Map<String, String> data) {  
  
 String login = data.get("login");  
 String password = data.get("password");  
 String username = data.get("username");  
 String masterKey = data.get("masterKey");  
  
 *//Если клиент передал не все значения* if ((login == null) || (password == null) || (username == null) || (masterKey == null)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не все значения были переданы"), HttpStatus.*BAD\_REQUEST*);  
 }  
 login = login.toLowerCase(Locale.*ROOT*);  
  
 UserDBEntity result = userService.create(login, password, username, masterKey);  
 *//Если все хорошо* if (result != null) {  
 UserAuthResponseEntity user = UserConverter.*convert*(result);  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(user), HttpStatus.*OK*);  
 }  
 *//Если такой ник или e-mail занят* return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Пользователь с таким ником или e-mail уже существует"), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Поиск пользователя в системе  
 \*  
 \* @param key - ключ пользователя, выполняющего поиск  
 \* @param searchName - паттерн имени ника пользователя, которого ищем (необязательно)  
 \* @param limit - кол-во пользователей, которое надо вернуть, макс 200 (необязательно)  
 \* @return response entity  
 \*/* @GetMapping(value = "/search")  
 public ResponseEntity<Map<String, Object>> searchUser(@RequestParam String key, @RequestParam(required = false) String searchName, @RequestParam(required = false) Integer limit) {  
  
 *//Проверка ключа* if (!userService.checkUserKey(key)) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось авторизоваться по указанному ключу"), HttpStatus.*FORBIDDEN*);  
 }  
  
 if ((limit == null) || (limit > 200)) {  
 limit = 200;  
 }  
  
 List<UserDBEntity> bufList = userService.searchUsers(searchName, limit, key);  
 List<UserSearchResponseEntity> resultList = new ArrayList<>();  
 for (UserDBEntity item : bufList) {  
 resultList.add(UserSearchConverter.*convert*(item));  
 }  
  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(resultList), HttpStatus.*OK*);  
 }  
  
 */\*\*  
 \* Восстановление (обновление) пароля пользователя по мастер-ключу  
 \*  
 \* @param updateItem - объект UpdatePasswordEntity  
 \* @return response entity  
 \*/* @PutMapping("/reset")  
 public ResponseEntity<?> updatePassword(@RequestBody UpdatePasswordEntity updateItem) {  
  
 Optional<UserDBEntity> searchResult = userService.findByMasterKeyAndEmail(updateItem.getMasterKey(), updateItem.getEmail());  
 *//Если не нашли пользователей, которые связаны с переданным e-mail и мастер-ключем* if (searchResult.isEmpty()) {  
 return new ResponseEntity<>(GenResponseUtil.*ResponseError*("Не удалось сменить пароль."), HttpStatus.*OK*);  
 }  
 UserDBEntity currentUser = searchResult.get();  
 userService.update(currentUser, updateItem);  
 return new ResponseEntity<>(GenResponseUtil.*ResponseOK*(UserSearchConverter.*convert*(currentUser)), HttpStatus.*OK*);  
 }  
}

com/demka/demkaserver/entities/converters/UserConverter.java

package com.demka.demkaserver.entities.converters;  
  
import com.demka.demkaserver.entities.database.UserDBEntity;  
import com.demka.demkaserver.entities.response.UserAuthResponseEntity;  
  
*/\*\*  
 \* The type User converter.  
 \*/*public class UserConverter {  
  
 */\*\*  
 \* Логика конвертации с UserDBEntity в UserAuthResponseEntity, который отдается при авторизации пользователя  
 \* Это нужно для того, чтоб избавиться от полей, которые не нужны в ответе (т.е. пароль, мастер-ключ и т д)  
 \*  
 \* @param dbEntity - сущность UserDBEntity  
 \* @return user auth response entity  
 \*/* public static UserAuthResponseEntity convert(UserDBEntity dbEntity) {  
 UserAuthResponseEntity responseEntity = new UserAuthResponseEntity();  
 responseEntity.setId(dbEntity.getId());  
 responseEntity.setName(dbEntity.getName());  
 responseEntity.setKey(dbEntity.getKey());  
 responseEntity.setTimeCreated(dbEntity.getTimeCreated());  
 return responseEntity;  
 }  
}

com/demka/demkaserver/entities/converters/UserSearchConverter.java

package com.demka.demkaserver.entities.converters;  
  
import com.demka.demkaserver.entities.database.UserDBEntity;  
import com.demka.demkaserver.entities.response.UserSearchResponseEntity;  
  
*/\*\*  
 \* The type User search converter.  
 \*/*public class UserSearchConverter {  
  
 */\*\*  
 \* Логика конвертации с UserDBEntity в UserSearchResponseEntity, который отдается при поиске пользователя в системе  
 \* Это нужно для того, чтоб избавиться от всех полей, которые не нужны в ответе (т.е. пароль, мастер-ключ и т д)  
 \*  
 \* @param dbEntity - сущность UserDBEntity  
 \* @return user search response entity  
 \*/* public static UserSearchResponseEntity convert(UserDBEntity dbEntity) {  
 UserSearchResponseEntity responseEntity = new UserSearchResponseEntity();  
  
 responseEntity.setId(dbEntity.getId());  
 responseEntity.setName(dbEntity.getName());  
 responseEntity.setTimeCreated(dbEntity.getTimeCreated());  
  
 return responseEntity;  
 }  
}

com/demka/demkaserver/entities/database/LongPollDBEntity.java

package com.demka.demkaserver.entities.database;  
  
import com.fasterxml.jackson.annotation.JsonProperty;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import org.springframework.data.annotation.Id;  
import org.springframework.data.mongodb.core.mapping.Document;  
import org.springframework.data.mongodb.core.mapping.Field;  
  
*/\*\*  
 \* Сущность Лонгпула в MongoDB  
 \*/*@Data  
@NoArgsConstructor  
@Document(collection = "LongPolls")  
public class LongPollDBEntity {  
  
 @Id  
 @JsonProperty  
 private String id;  
 @JsonProperty  
 private Long ts;  
 @JsonProperty  
 private String key;  
 @JsonProperty  
 private String url;  
 @JsonProperty  
 @Field(name = "user\_id")  
 private String userId;  
  
 @Override  
 public String toString() {  
 return "LongPollDBEntity{" +  
 "id='" + id + '\'' +  
 ", ts=" + ts +  
 ", key='" + key + '\'' +  
 ", url='" + url + '\'' +  
 ", userId='" + userId + '\'' +  
 '}';  
 }  
}

com/demka/demkaserver/entities/database/MessageDBEntity.java

package com.demka.demkaserver.entities.database;  
  
import com.fasterxml.jackson.annotation.JsonProperty;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import org.springframework.data.annotation.Id;  
import org.springframework.data.mongodb.core.mapping.Document;  
import org.springframework.data.mongodb.core.mapping.Field;  
  
*/\*\*  
 \* Сущность сообщения пользователя в MongoDB  
 \*/*@Data  
@NoArgsConstructor  
@Document(collection = "Messages")  
public class MessageDBEntity {  
  
 @Id  
 @JsonProperty  
 private String id;  
 @JsonProperty  
 @Field(name = "user\_name")  
 private String userName;  
 @JsonProperty  
 @Field(name = "user\_id")  
 private String userId;  
 @JsonProperty  
 private String text;  
 @JsonProperty  
 @Field(name = "room\_id")  
 private String roomId;  
 @JsonProperty  
 @Field(name = "time\_created")  
 private Long timeCreated;  
  
 @Override  
 public String toString() {  
 return "MessageDBEntity{" +  
 "id='" + id + '\'' +  
 ", userName='" + userName + '\'' +  
 ", userId='" + userId + '\'' +  
 ", text='" + text + '\'' +  
 ", roomId='" + roomId + '\'' +  
 ", timeCreated=" + timeCreated +  
 '}';  
 }  
}

com/demka/demkaserver/entities/database/RoomDBEntity.java

package com.demka.demkaserver.entities.database;  
  
import com.fasterxml.jackson.annotation.JsonProperty;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import org.springframework.data.annotation.Id;  
import org.springframework.data.mongodb.core.mapping.Document;  
import org.springframework.data.mongodb.core.mapping.Field;  
  
import java.util.List;  
  
*/\*\*  
 \* Сущность чат-комнаты в MongoDB  
 \*/*@Data  
@NoArgsConstructor  
@Document(collection = "Rooms")  
public class RoomDBEntity {  
  
 @Id  
 @JsonProperty  
 private String id;  
 @JsonProperty  
 @Field(name = "creator\_id")  
 private String creatorId;  
 @JsonProperty  
 private List<String> users;  
 @JsonProperty  
 private String name;  
 @JsonProperty  
 @Field(name = "time\_created")  
 private Long timeCreated;  
  
 @Override  
 public String toString() {  
 return "RoomDBEntity{" +  
 "id='" + id + '\'' +  
 ", creatorId='" + creatorId + '\'' +  
 ", users=" + users +  
 ", name='" + name + '\'' +  
 ", timeCreated=" + timeCreated +  
 '}';  
 }  
}

com/demka/demkaserver/entities/database/RoomDBEntity.java

package com.demka.demkaserver.entities.database;  
  
  
import com.fasterxml.jackson.annotation.JsonProperty;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import org.springframework.data.annotation.Id;  
import org.springframework.data.mongodb.core.index.Indexed;  
import org.springframework.data.mongodb.core.mapping.Document;  
import org.springframework.data.mongodb.core.mapping.Field;  
  
*/\*\*  
 \* Сущность пользователя в MongoDB  
 \*/*@Data  
@NoArgsConstructor  
@Document(collection = "Users")  
public class UserDBEntity {  
  
 @Id  
 @JsonProperty  
 private String id;  
 @JsonProperty  
 private String name;  
 @JsonProperty  
 @Indexed(unique = true)  
 private String login;  
 @JsonProperty  
 private String password;  
 @JsonProperty  
 private String key;  
 @JsonProperty  
 @Field(name = "master\_key")  
 private String masterKey;  
 @JsonProperty  
 @Field(name = "time\_created")  
 private Long timeCreated;  
  
 @Override  
 public String toString() {  
 return "UserDBEntity{" +  
 "id='" + id + '\'' +  
 ", name='" + name + '\'' +  
 ", login='" + login + '\'' +  
 ", password='" + password + '\'' +  
 ", key='" + key + '\'' +  
 ", masterKey='" + masterKey + '\'' +  
 ", timeCreated=" + timeCreated +  
 '}';  
 }  
}

com/demka/demkaserver/entities/database/UserDBEntity.java

package com.demka.demkaserver.entities.database;  
  
  
import com.fasterxml.jackson.annotation.JsonProperty;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
import org.springframework.data.annotation.Id;  
import org.springframework.data.mongodb.core.index.Indexed;  
import org.springframework.data.mongodb.core.mapping.Document;  
import org.springframework.data.mongodb.core.mapping.Field;  
  
*/\*\*  
 \* Сущность пользователя в MongoDB  
 \*/*@Data  
@NoArgsConstructor  
@Document(collection = "Users")  
public class UserDBEntity {  
  
 @Id  
 @JsonProperty  
 private String id;  
 @JsonProperty  
 private String name;  
 @JsonProperty  
 @Indexed(unique = true)  
 private String login;  
 @JsonProperty  
 private String password;  
 @JsonProperty  
 private String key;  
 @JsonProperty  
 @Field(name = "master\_key")  
 private String masterKey;  
 @JsonProperty  
 @Field(name = "time\_created")  
 private Long timeCreated;  
  
 @Override  
 public String toString() {  
 return "UserDBEntity{" +  
 "id='" + id + '\'' +  
 ", name='" + name + '\'' +  
 ", login='" + login + '\'' +  
 ", password='" + password + '\'' +  
 ", key='" + key + '\'' +  
 ", masterKey='" + masterKey + '\'' +  
 ", timeCreated=" + timeCreated +  
 '}';  
 }  
}

com/demka/demkaserver/entities/request/UpdatePasswordEntity.java

package com.demka.demkaserver.entities.request;  
  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
*/\*\*  
 \* Сущность для обновления пароля пользователя  
 \* Используется при приёме данных  
 \*/*@Data  
@NoArgsConstructor  
public class UpdatePasswordEntity {  
 private String email;  
 private String masterKey;  
 private String newPassword;  
  
 @Override  
 public String toString() {  
 return "UpdatePasswordEntity{" +  
 "email='" + email + '\'' +  
 ", masterKey='" + masterKey + '\'' +  
 ", newPassword='" + newPassword + '\'' +  
 '}';  
 }  
}

com/demka/demkaserver/entities/response/UserAuthResponseEntity.java

package com.demka.demkaserver.entities.response;  
  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
*/\*\*  
 \* Сущность для отдачи данных при авторизации пользователя  
 \* Является частным случаем UserDBEntity без полей-паролей, но с ключем API  
 \* Используется при отдаче данных  
 \*/*@Data  
@NoArgsConstructor  
public class UserAuthResponseEntity {  
 private String id;  
 private String name;  
 private String key;  
 private Long timeCreated;  
  
 @Override  
 public String toString() {  
 return "UserAuthResponseEntity{" +  
 "id='" + id + '\'' +  
 ", name='" + name + '\'' +  
 ", key='" + key + '\'' +  
 ", timeCreated=" + timeCreated +  
 '}';  
 }  
}

com/demka/demkaserver/entities/response/UserSearchResponseEntity.java

package com.demka.demkaserver.entities.response;  
  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
*/\*\*  
 \* Сущность для отдачи данных при поиске пользователя  
 \* Является частным случаем UserDBEntity без полей-паролей и без ключа API  
 \* Используется при отдаче данных  
 \*/*@Data  
@NoArgsConstructor  
public class UserSearchResponseEntity {  
 private String id;  
 private String name;  
 private Long timeCreated;  
  
 @Override  
 public String toString() {  
 return "UserSearchResponseEntity{" +  
 "id='" + id + '\'' +  
 ", name='" + name + '\'' +  
 ", timeCreated=" + timeCreated +  
 '}';  
 }  
}

com/demka/demkaserver/repos/LongPollRepository.java

package com.demka.demkaserver.repos;  
  
import com.demka.demkaserver.entities.database.LongPollDBEntity;  
import org.springframework.data.mongodb.repository.MongoRepository;  
import org.springframework.data.mongodb.repository.Query;  
  
import java.util.Optional;  
  
*/\*\*  
 \* Интерфейс MongoRepository для работы c коллекцией лонгпулов  
 \*/*public interface LongPollRepository extends MongoRepository<LongPollDBEntity, String> {  
  
 */\*\*  
 \* Find by url optional.  
 \*  
 \* @param url the url  
 \* @return the optional  
 \*/* @Query("{ 'url' : ?0}")  
 Optional<LongPollDBEntity> findByUrl(String url);  
  
 */\*\*  
 \* Find by key and url optional.  
 \*  
 \* @param key the key  
 \* @param url the url  
 \* @return the optional  
 \*/* @Query("{ 'key' : ?0, 'url' : ?1}")  
 Optional<LongPollDBEntity> findByKeyAndUrl(String key, String url);  
}

com/demka/demkaserver/repos/MessageRepository.java

package com.demka.demkaserver.repos;  
  
import com.demka.demkaserver.entities.database.MessageDBEntity;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.mongodb.repository.MongoRepository;  
import org.springframework.data.mongodb.repository.Query;  
  
import java.util.List;  
  
*/\*\*  
 \* Интерфейс MongoRepository для работы c коллекцией сообщений  
 \*/*public interface MessageRepository extends MongoRepository<MessageDBEntity, String> {  
  
 */\*\*  
 \* Find all by room id list.  
 \*  
 \* @param roomId the room id  
 \* @param pageable the pageable  
 \* @return the list  
 \*/* @Query("{ 'room\_id' : ?0}")  
 List<MessageDBEntity> findAllByRoomId(String roomId, Pageable pageable);  
  
 */\*\*  
 \* Find all by user list.  
 \*  
 \* @param userId the user id  
 \* @param pageable the pageable  
 \* @return the list  
 \*/* @Query("{ 'user\_id' : ?0}")  
 List<MessageDBEntity> findAllByUser(String userId, Pageable pageable);  
  
 */\*\*  
 \* Gets new messages by room.  
 \*  
 \* @param roomId the room id  
 \* @param ts the ts  
 \* @return the new messages by room  
 \*/* @Query("{ 'room\_id': ?0, 'time\_created': {'$gt': ?1}}")  
 List<MessageDBEntity> getNewMessagesByRoom(String roomId, Long ts);  
  
}

com/demka/demkaserver/repos/RoomRepository.java

package com.demka.demkaserver.repos;  
  
import com.demka.demkaserver.entities.database.RoomDBEntity;  
import org.springframework.data.mongodb.repository.MongoRepository;  
import org.springframework.data.mongodb.repository.Query;  
  
import java.util.List;  
  
*/\*\*  
 \* Интерфейс MongoRepository для работы c коллекцией чат-комнат  
 \*/*public interface RoomRepository extends MongoRepository<RoomDBEntity, String> {  
  
 */\*\*  
 \* Find all by user list.  
 \*  
 \* @param userId the user id  
 \* @return the list  
 \*/* @Query("{ 'users' : ?0}")  
 List<RoomDBEntity> findAllByUser(String userId);  
}

com/demka/demkaserver/repos/UserRepository.java

package com.demka.demkaserver.repos;  
  
import com.demka.demkaserver.entities.database.UserDBEntity;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.mongodb.repository.MongoRepository;  
import org.springframework.data.mongodb.repository.Query;  
  
import java.util.List;  
import java.util.Optional;  
  
*/\*\*  
 \* Интерфейс MongoRepository для работы c коллекцией пользователей  
 \*/*public interface UserRepository extends MongoRepository<UserDBEntity, String> {  
  
  
 */\*\*  
 \* Check user key optional.  
 \*  
 \* @param key the key  
 \* @return the optional  
 \*/* @Query("{ 'key' : ?0}")  
 Optional<UserDBEntity> checkUserKey(String key);  
  
 */\*\*  
 \* Check user auth optional.  
 \*  
 \* @param login the login  
 \* @param password the password  
 \* @return the optional  
 \*/* @Query("{ 'login' : ?0, 'password' : ?1}")  
 Optional<UserDBEntity> checkUserAuth(String login, String password);  
  
 */\*\*  
 \* Find by login optional.  
 \*  
 \* @param login the login  
 \* @return the optional  
 \*/* @Query("{ 'login' : ?0}")  
 Optional<UserDBEntity> findByLogin(String login);  
  
 */\*\*  
 \* Find by name optional.  
 \*  
 \* @param name the name  
 \* @return the optional  
 \*/* @Query("{ 'name' : ?0}")  
 Optional<UserDBEntity> findByName(String name);  
  
 */\*\*  
 \* Find all by name limit list.  
 \*  
 \* @param regexp the regexp  
 \* @param pageable the pageable  
 \* @return the list  
 \*/* @Query("{ 'name' : { $regex: ?0 } }")  
 List<UserDBEntity> findAllByNameLimit(String regexp, Pageable pageable);  
  
 */\*\*  
 \* Find all limit list.  
 \*  
 \* @param pageable the pageable  
 \* @return the list  
 \*/* @Query("{}")  
 List<UserDBEntity> findAllLimit(Pageable pageable);  
  
 */\*\*  
 \* Find by master key and email optional.  
 \*  
 \* @param masterKey the master key  
 \* @param email the email  
 \* @return the optional  
 \*/* @Query("{ 'master\_key' : ?0, 'login' : ?1}")  
 Optional<UserDBEntity> findByMasterKeyAndEmail(String masterKey, String email);  
}

com/demka/demkaserver/services/LongPollService.java

package com.demka.demkaserver.services;  
  
import com.demka.demkaserver.entities.database.LongPollDBEntity;  
import com.demka.demkaserver.repos.LongPollRepository;  
import com.demka.demkaserver.utils.UUIDUtil;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
import java.util.Optional;  
  
*/\*\*  
 \* Сервис для работы с лонгпулами  
 \*/*@Service  
public class LongPollService {  
  
 private final LongPollRepository longPollRepo;  
  
 */\*\*  
 \* Instantiates a new Long poll service.  
 \*  
 \* @param longPollRepo the long poll repo  
 \*/* @Autowired  
 public LongPollService(LongPollRepository longPollRepo) {  
 this.longPollRepo = longPollRepo;  
 }  
  
 */\*\*  
 \* Создание нового лонгпула  
 \*  
 \* @param userId - иднетификатор пользователя  
 \* @param ts - UNIX-время последнего принятого сообщения  
 \* @return long poll db entity  
 \*/* public LongPollDBEntity create(String userId, Long ts) {  
  
 LongPollDBEntity newPoll = new LongPollDBEntity();  
 newPoll.setId(UUIDUtil.*newId*());  
 newPoll.setTs(ts);  
 newPoll.setKey(UUIDUtil.*newKey*());  
 newPoll.setUserId(userId);  
 newPoll.setUrl(UUIDUtil.*newURL*());  
  
 longPollRepo.save(newPoll);  
 return newPoll;  
 }  
  
 */\*\*  
 \* Поиск лонгпула по URL  
 \*  
 \* @param url - URL лонгпула  
 \* @return optional  
 \*/* public Optional<LongPollDBEntity> findByUrl(String url) {  
 return longPollRepo.findByUrl(url);  
 }  
  
 */\*\*  
 \* Поиск лонгпула по ключу и url  
 \*  
 \* @param key - ключ лонгпула  
 \* @param url - URL лонгпула  
 \* @return optional  
 \*/* public Optional<LongPollDBEntity> findByKeyAndUrl(String key, String url) {  
 return longPollRepo.findByKeyAndUrl(key, url);  
 }  
  
 */\*\*  
 \* Обновление данных лонгпула  
 \*  
 \* @param oldObj - старый объект лонгпула  
 \* @param newObj - новый объект лонгпула  
 \*/* public void update(LongPollDBEntity oldObj, LongPollDBEntity newObj) {  
 longPollRepo.delete(oldObj);  
 longPollRepo.save(newObj);  
 }  
  
 */\*\*  
 \* Удаление лонгпула по его id  
 \*  
 \* @param pollId - иднетификатор лонгпула  
 \*/* public void delete(String pollId) {  
 longPollRepo.deleteById(pollId);  
 }  
  
 */\*\*  
 \* Удаление всех лонгпулов в коллекции.  
 \* Необходим для отчистки всей коллекции при перезапуске сервера  
 \* Используется в переопределенном InitializingBean  
 \*/* public void deleteAll() {  
 longPollRepo.deleteAll();  
 }  
  
 */\*\*  
 \* Поиск всех лонгпулов  
 \*  
 \* @return list  
 \*/* public List<LongPollDBEntity> findAll() {  
 return longPollRepo.findAll();  
 }  
  
 */\*\*  
 \* Поиск лонгпула по его id  
 \*  
 \* @param id - идентификатор лонгпула  
 \* @return optional  
 \*/* public Optional<LongPollDBEntity> find(String id) {  
 return longPollRepo.findById(id);  
 }  
}

com/demka/demkaserver/services/MessageService.java

package com.demka.demkaserver.services;  
  
import com.demka.demkaserver.entities.database.MessageDBEntity;  
import com.demka.demkaserver.entities.database.RoomDBEntity;  
import com.demka.demkaserver.repos.MessageRepository;  
import com.demka.demkaserver.utils.TimeUtil;  
import com.demka.demkaserver.utils.UUIDUtil;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.domain.PageRequest;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
*/\*\*  
 \* Сервис для работы с сообщениями  
 \*/*@Service  
public class MessageService {  
  
 private final MessageRepository messageRepo;  
  
 */\*\*  
 \* Instantiates a new Message service.  
 \*  
 \* @param messageRepo the message repo  
 \*/* @Autowired  
 public MessageService(MessageRepository messageRepo) {  
 this.messageRepo = messageRepo;  
 }  
  
 */\*\*  
 \* Отдельный метод для получения последнего сообщения пользователя по времени  
 \*  
 \* @param userId - идентификатор пользователя  
 \* @return message db entity  
 \*/* public MessageDBEntity GetLastMessageByUser(String userId) {  
  
 Pageable pageLimit = PageRequest.*of*(0, 1, Sort.*by*(Sort.Direction.*DESC*, "time\_created"));  
 List<MessageDBEntity> userMessageList = messageRepo.findAllByUser(userId, pageLimit);  
 if (userMessageList.size() != 1) {  
 return null;  
 }  
 return userMessageList.get(0);  
 }  
  
 */\*\*  
 \* Создание нового сообщения  
 \*  
 \* @param userId - идентификатор пользователя, который отправил сообщение  
 \* @param userName - имя пользователя, который отправляет сообщение  
 \* @param text - текст сообщения  
 \* @param roomId - идентификатор комнаты, куда отправляется сообщение  
 \* @return message db entity  
 \*/* public MessageDBEntity create(String userId, String userName, String text, String roomId) {  
  
 MessageDBEntity newMessage = new MessageDBEntity();  
 newMessage.setUserId(userId);  
 newMessage.setUserName(userName);  
 newMessage.setText(text);  
 newMessage.setRoomId(roomId);  
  
 newMessage.setTimeCreated(TimeUtil.*unixTime*());  
 newMessage.setId(UUIDUtil.*newId*());  
  
 messageRepo.save(newMessage);  
 return newMessage;  
 }  
  
 */\*\*  
 \* Обновление сообщения  
 \*  
 \* @param oldObj - старый объект сообщения  
 \* @param newObj - новый оьъект сообщения  
 \*/* public void update(MessageDBEntity oldObj, MessageDBEntity newObj) {  
 messageRepo.delete(oldObj);  
 messageRepo.save(newObj);  
 }  
  
  
 */\*\*  
 \* Получение новых сообщний по каждой комнате, время которых больше, чем указанное в ts  
 \*  
 \* @param roomsList - список комнат пользователя (предварительно вызывается в findUserRooms)  
 \* @param ts - UNIX-время  
 \* @return new messages by rooms  
 \*/* public List<MessageDBEntity> getNewMessagesByRooms(List<RoomDBEntity> roomsList, Long ts) {  
 List<MessageDBEntity> messagesList = new ArrayList<>();  
  
 for (RoomDBEntity room : roomsList) {  
 messagesList.addAll(messageRepo.getNewMessagesByRoom(room.getId(), ts));  
 }  
 return messagesList;  
  
 }  
  
 */\*\*  
 \* Получение всех сообщний по каждой комнате  
 \*  
 \* @param roomsList - список комнат пользователя (предварительно вызывается в findUserRooms)  
 \* @return all messages by rooms  
 \*/* public List<MessageDBEntity> getAllMessagesByRooms(List<RoomDBEntity> roomsList) {  
 List<MessageDBEntity> messagesList = new ArrayList<>();  
 Pageable pageLimit = PageRequest.*of*(0, Integer.*MAX\_VALUE*, Sort.*by*(Sort.Direction.*ASC*, "time\_created"));  
 for (RoomDBEntity room : roomsList) {  
 messagesList.addAll(messageRepo.findAllByRoomId(room.getId(), pageLimit));  
 }  
 return messagesList;  
 }  
  
 */\*\*  
 \* Получение сообщений по конкретной комнате  
 \*  
 \* @param roomId - идентификатор комнаты  
 \* @return list  
 \*/* public List<MessageDBEntity> findByRoom(String roomId) {  
 Pageable pageLimit = PageRequest.*of*(0, Integer.*MAX\_VALUE*, Sort.*by*(Sort.Direction.*ASC*, "time\_created"));  
 return messageRepo.findAllByRoomId(roomId, pageLimit);  
 }  
  
 */\*\*  
 \* Поиск сообщения по его id  
 \*  
 \* @param id - идентификатор сообщения  
 \* @return optional  
 \*/* public Optional<MessageDBEntity> find(String id) {  
 return messageRepo.findById(id);  
 }  
  
 */\*\*  
 \* Удаление сообщения по его объекту  
 \*  
 \* @param item - объект сообщения  
 \*/* public void delete(MessageDBEntity item) {  
 messageRepo.delete(item);  
 }  
  
 */\*\*  
 \* Поиск всех сообщений  
 \*  
 \* @return list  
 \*/* public List<MessageDBEntity> findAll() {  
 return messageRepo.findAll();  
 }  
  
}

com/demka/demkaserver/services/RoomService.java

package com.demka.demkaserver.services;  
  
import com.demka.demkaserver.entities.database.RoomDBEntity;  
import com.demka.demkaserver.entities.database.UserDBEntity;  
import com.demka.demkaserver.repos.RoomRepository;  
import com.demka.demkaserver.utils.TimeUtil;  
import com.demka.demkaserver.utils.UUIDUtil;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
*/\*\*  
 \* Сервис для работы с комнатами  
 \*/*@Service  
public class RoomService {  
  
 private final RoomRepository roomRepo;  
  
 */\*\*  
 \* Instantiates a new Room service.  
 \*  
 \* @param roomRepo the room repo  
 \*/* @Autowired  
 public RoomService(RoomRepository roomRepo) {  
 this.roomRepo = roomRepo;  
 }  
  
 */\*\*  
 \* Отдаёт список комнат, в которых состоит пользователь  
 \*  
 \* @param userId - идентификатор пользователя  
 \* @return list  
 \*/* public List<RoomDBEntity> findUserRooms(String userId) {  
 return roomRepo.findAllByUser(userId);  
 }  
  
 */\*\*  
 \* Поиск комнаты по её id  
 \*  
 \* @param id - идентификатор комнаты  
 \* @return optional  
 \*/* public Optional<RoomDBEntity> find(String id) {  
 return roomRepo.findById(id);  
 }  
  
 */\*\*  
 \* Коздание комнаты  
 \*  
 \* @param creatorId - идентификатор пользователя-создателя комнаты  
 \* @param roomName - имя комнаты  
 \* @param usersBuffer - спосок идентификаторов пользователей-участников комнаты  
 \* @return room db entity  
 \*/* public RoomDBEntity create(String creatorId, String roomName, List<String> usersBuffer) {  
  
 *//Проверка, чтоб создатель также был в списке пользователей комнаты* List<String> allUsers = new ArrayList<>(usersBuffer);  
  
 if (!allUsers.contains(creatorId)) {  
 allUsers.add(creatorId);  
 }  
  
 System.*out*.println();  
  
 RoomDBEntity newRoom = new RoomDBEntity();  
 newRoom.setCreatorId(creatorId);  
 newRoom.setName(roomName);  
 newRoom.setUsers(allUsers);  
 newRoom.setId(UUIDUtil.*newId*());  
 newRoom.setTimeCreated(TimeUtil.*unixTime*());  
  
 roomRepo.save(newRoom);  
 return newRoom;  
 }  
  
 */\*\*  
 \* Обновление комнаты  
 \*  
 \* @param oldObj - старый объект комнаты  
 \* @param newObj - новый объект комнаты  
 \*/* public void update(RoomDBEntity oldObj, RoomDBEntity newObj) {  
 roomRepo.delete(oldObj);  
 roomRepo.save(newObj);  
 }  
  
 */\*\*  
 \* Удаление комнаты по её объекту  
 \*  
 \* @param room - объект комнаты  
 \*/* public void delete(RoomDBEntity room) {  
 roomRepo.delete(room);  
 }  
  
 */\*\*  
 \* Поиск всех комнат  
 \*  
 \* @return list  
 \*/* public List<RoomDBEntity> findAll() {  
 return roomRepo.findAll();  
 }  
  
 */\*\*  
 \* Удаление пользователя из комнаты  
 \*  
 \* @param room - объект комнаты  
 \* @param userId - объект пользователя для удаления  
 \*/* public void removeUser(RoomDBEntity room, String userId) {  
 room.getUsers().remove(userId);  
 roomRepo.save(room);  
 }  
  
 */\*\*  
 \* Удаление пользователя из комнаты  
 \*  
 \* @param room - объект комнаты  
 \* @param user - объект пользователя для удаления  
 \*/* public void removeUser(RoomDBEntity room, UserDBEntity user) {  
 String userId = user.getId();  
 room.getUsers().remove(userId);  
 roomRepo.save(room);  
 }  
  
 */\*\*  
 \* Добавление пользователя в комнату  
 \*  
 \* @param room - объект комнаты  
 \* @param userId - идентификатор пользователя для добавления  
 \*/* public void addUser(RoomDBEntity room, String userId) {  
 room.getUsers().add(userId);  
 roomRepo.save(room);  
 }  
  
 */\*\*  
 \* Добавление пользователя в комнату  
 \*  
 \* @param room - объект комнаты  
 \* @param user - объект пользователя для добавления  
 \*/* public void addUser(RoomDBEntity room, UserDBEntity user) {  
 String userId = user.getId();  
 room.getUsers().add(userId);  
 roomRepo.save(room);  
 }  
}

com/demka/demkaserver/services/UserService.java

package com.demka.demkaserver.services;  
  
import com.demka.demkaserver.entities.database.UserDBEntity;  
import com.demka.demkaserver.entities.request.UpdatePasswordEntity;  
import com.demka.demkaserver.repos.UserRepository;  
import com.demka.demkaserver.utils.TimeUtil;  
import com.demka.demkaserver.utils.UUIDUtil;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.data.domain.PageRequest;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.domain.Sort;  
import org.springframework.stereotype.Service;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
  
*/\*\*  
 \* Сервис для работы с пользователями  
 \*/*@Service  
public class UserService {  
  
 private final UserRepository userRepo;  
  
 */\*\*  
 \* Instantiates a new User service.  
 \*  
 \* @param userRepo the user repo  
 \*/* @Autowired  
 public UserService(UserRepository userRepo) {  
 this.userRepo = userRepo;  
 }  
  
 */\*\*  
 \* Обновление пароля пользователя  
 \*  
 \* @param user - объект пользователя  
 \* @param newData - объект обновления пароля (да, объект)  
 \*/* public void update(UserDBEntity user, UpdatePasswordEntity newData) {  
 user.setPassword(newData.getNewPassword());  
 userRepo.save(user);  
 }  
  
 */\*\*  
 \* Поиск пользователя по его id  
 \*  
 \* @param id - идентификатор пользователя  
 \* @return optional  
 \*/* public Optional<UserDBEntity> find(String id) {  
 return userRepo.findById(id);  
 }  
  
 */\*\*  
 \* Поиск пользователя по мастер-ключу и e-mail  
 \* Используется при восстановлении пароля  
 \*  
 \* @param masterKey - мастер-ключ пользователя  
 \* @param login - логин пользоваетля  
 \* @return optional  
 \*/* public Optional<UserDBEntity> findByMasterKeyAndEmail(String masterKey, String login) {  
 return userRepo.findByMasterKeyAndEmail(masterKey, login);  
 }  
  
 */\*\*  
 \* Проверка на авторизацию пользователя по паре логин/пароль  
 \*  
 \* @param login - логин пользователя  
 \* @param password - пароль пользователя  
 \* @return user db entity  
 \*/* public UserDBEntity checkAuth(String login, String password) {  
 Optional<UserDBEntity> result = userRepo.checkUserAuth(login, password);  
 return result.orElse(null);  
 }  
  
 */\*\*  
 \* Проверка на авторизацию пользователя по ключу API  
 \*  
 \* @param key - ключ API пользователя  
 \* @return user db entity  
 \*/* public UserDBEntity checkAuth(String key) {  
 Optional<UserDBEntity> result = userRepo.checkUserKey(key);  
 return result.orElse(null);  
 }  
  
 */\*\*  
 \* Создание/регистрация пользователя в системе  
 \*  
 \* @param login - логин пользователя (e-mail)  
 \* @param password - пароль пользоваетля  
 \* @param username - ник пользователя (не путать с логином)  
 \* @param masterPassword - мастер-ключ пользователя. Нужен для восстановления пароля.  
 \* @return user db entity  
 \*/* public UserDBEntity create(String login, String password, String username, String masterPassword) {  
  
 *//Проверяем на то, не зарегался ли уже пользователь с таким email или именем* if ((userRepo.findByLogin(login).isPresent()) || (userRepo.findByName(username).isPresent())) {  
 return null;  
 }  
  
 UserDBEntity newUser = new UserDBEntity();  
  
 newUser.setLogin(login);  
 newUser.setPassword(password);  
 newUser.setMasterKey(masterPassword);  
 newUser.setName(username);  
  
 newUser.setKey(UUIDUtil.*newKey*());  
 newUser.setTimeCreated(TimeUtil.*unixTime*());  
 newUser.setId(UUIDUtil.*newId*());  
  
 userRepo.save(newUser);  
 return newUser;  
 }  
  
 */\*\*  
 \* Поиск пользователя по ключу API  
 \*  
 \* @param key - ключ API пользователя  
 \* @return optional  
 \*/* public Optional<UserDBEntity> findByKey(String key) {  
 return userRepo.checkUserKey(key);  
 }  
  
 */\*\*  
 \* Поиск пользователей в системе. Используется при создании комнаты и выборе её участников  
 \*  
 \* @param name - паттерн имени пользователя  
 \* @param limit - лимит найденный пользователей  
 \* @param currentUserKey - ключ API пользователя, осуществляющего поиск  
 \* @return list  
 \*/* public List<UserDBEntity> searchUsers(String name, Integer limit, String currentUserKey) {  
  
 List<UserDBEntity> bufList = new ArrayList<>();  
 List<UserDBEntity> resultList = new ArrayList<>();  
 Pageable pageLimit = PageRequest.*of*(0, limit, Sort.*by*(Sort.Direction.*DESC*, "time\_created"));  
  
 if (name != null) {  
 bufList.addAll(userRepo.findAllByNameLimit("" + name, pageLimit));  
 } else {  
 bufList.addAll(userRepo.findAllLimit(pageLimit));  
 }  
  
 *//Выкидываем самого пользователя, который запросил данные (чтоб не отображать ему самого себя)* for (UserDBEntity item : bufList) {  
 if (!item.getKey().equals(currentUserKey)) {  
 resultList.add(item);  
 }  
 }  
 return resultList;  
 }  
  
 */\*\*  
 \* Проверка на существование ключа API пользователя  
 \*  
 \* @param key - ключ API пользователя  
 \* @return boolean  
 \*/* public boolean checkUserKey(String key) {  
 return userRepo.checkUserKey(key).isPresent();  
 }  
  
 */\*\*  
 \* Удаление пользователя по его объекту  
 \*  
 \* @param item - объект пользователя  
 \*/* public void delete(UserDBEntity item) {  
 userRepo.delete(item);  
 }  
  
 */\*\*  
 \* Поиск всех пользователей в коллекции  
 \*  
 \* @return list  
 \*/* public List<UserDBEntity> findAll() {  
 return userRepo.findAll();  
 }  
}

com/demka/demkaserver/utils/GenResponseUtil.java

package com.demka.demkaserver.utils;  
  
import java.util.HashMap;  
  
*/\*\*  
 \* The type Gen response util.  
 \*/*public class GenResponseUtil {  
 */\*\*  
 \* Генерирует HashMap для успешно обработанного запроса  
 \*  
 \* @param bodyObject - любой объект, который надо передать в body ответа  
 \* @return hash map  
 \*/* public static HashMap<String, Object> ResponseOK(Object bodyObject) {  
 HashMap<String, Object> map = new HashMap<>();  
 map.put("result", true);  
 map.put("body", bodyObject);  
 return map;  
 }  
  
 */\*\*  
 \* Генерирует HashMap для некорректного запроса пользователя  
 \*  
 \* @param description - описание того, почему это некорректный запрос  
 \* @return hash map  
 \*/* public static HashMap<String, Object> ResponseError(String description) {  
 HashMap<String, Object> map = new HashMap<>();  
 map.put("result", false);  
 map.put("description", description);  
 return map;  
 }  
}

com/demka/demkaserver/utils/TimeUtil.java

package com.demka.demkaserver.utils;  
  
import java.time.Instant;  
  
*/\*\*  
 \* The type Time util.  
 \*/*public class TimeUtil {  
 */\*\*  
 \* Возвращает текущее UNIX-время системы  
 \*  
 \* @return Текущее UNIX-время  
 \*/* public static long unixTime() {  
 return Instant.*now*().getEpochSecond();  
 }  
}

com/demka/demkaserver/utils/UUIDUtil.java

package com.demka.demkaserver.utils;  
  
import java.util.UUID;  
  
*/\*\*  
 \* The type Uuid util.  
 \*/*public class UUIDUtil {  
  
 */\*\*  
 \* Генерация id для записи в СУБД  
 \*  
 \* @return - новый сгенерированный id  
 \*/* public static String newId() {  
 return UUID.*randomUUID*().toString().replace("-", "");  
 }  
  
 */\*\*  
 \* Генерация ключа для использования в качестве ключа пользователя или лонгпула  
 \*  
 \* @return - новый сгенерированный ключ  
 \*/* public static String newKey() {  
 return UUID.*randomUUID*().toString().replace("-", "");  
 }  
  
 */\*\*  
 \* Генерация URL для обращения к лонгпулу  
 \*  
 \* @return - новый сгенерированный URL  
 \*/* public static String newURL() {  
 return UUID.*randomUUID*().toString().replace("-", "");  
 }  
}

## Приложение 3. Исходный код клиента

pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <groupId>org.demka</groupId>  
 <artifactId>newclient</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
 <maven.compiler.source>15</maven.compiler.source>  
 <maven.compiler.target>15</maven.compiler.target>  
 </properties>  
 <dependencies>  
  
 *<!--JavaFX -->* <dependency>  
 <groupId>org.openjfx</groupId>  
 <artifactId>javafx-controls</artifactId>  
 <version>15.0.1</version>  
 </dependency>  
 <dependency>  
 <groupId>org.openjfx</groupId>  
 <artifactId>javafx-fxml</artifactId>  
 <version>15.0.1</version>  
 </dependency>  
 <dependency>  
 <groupId>org.openjfx</groupId>  
 <artifactId>javafx-graphics</artifactId>  
 <version>15.0.1</version>  
 </dependency>  
  
 *<!--Красивый интерфейс-->* <dependency>  
 <groupId>com.jfoenix</groupId>  
 <artifactId>jfoenix</artifactId>  
 <version>9.0.10</version>  
 </dependency>  
  
 *<!--Работа с JSON-->* <dependency>  
 <groupId>com.google.code.gson</groupId>  
 <artifactId>gson</artifactId>  
 <version>2.8.6</version>  
 </dependency>  
  
 *<!--Логирование-->* <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>2.0.0-alpha1</version>  
 </dependency>  
 <dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
 <version>1.3.0-alpha5</version>  
 </dependency>  
  
 </dependencies>  
 <build>  
 <plugins>  
  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.8.0</version>  
 <configuration>  
 <release>11</release>  
 </configuration>  
 </plugin>  
  
 <plugin>  
 <groupId>org.openjfx</groupId>  
 <artifactId>javafx-maven-plugin</artifactId>  
 <version>0.0.5</version>  
 <configuration>  
 <mainClass>org.demka.App</mainClass>  
 <options>  
 <option>--add-opens</option>  
 <option>java.base/java.lang.reflect=com.jfoenix</option>  
 <option>-Dfile.encoding=UTF8</option>  
 </options>  
 <launcher>RUN\_ME</launcher>  
 <compress>2</compress>  
 <jlinkZipName>DEMKAChat</jlinkZipName>  
 <jlinkImageName>DEMKAChat</jlinkImageName>  
 <noManPages>true</noManPages>  
 <noHeaderFiles>true</noHeaderFiles>  
 </configuration>  
 </plugin>  
  
 </plugins>  
 </build>  
</project>

src/main/java/org/demka/api/HTTPRequest.java

package org.demka.api;  
  
import com.google.gson.Gson;  
  
import java.io.BufferedReader;  
import java.io.InputStreamReader;  
import java.io.OutputStream;  
import java.net.HttpURLConnection;  
import java.net.URL;  
import java.nio.charset.StandardCharsets;  
import java.util.Map;  
  
*/\*\*  
 \* The type Http request.  
 \*/*public class HTTPRequest {  
  
 */\*\*  
 \* Отправка GET-запроса  
 \*  
 \* @param urlString - URL, куда отправляется запрос  
 \* @return string  
 \*/* public static String sendGET(String urlString) {  
  
 try {  
 URL url = new URL(urlString);  
 HttpURLConnection con = (HttpURLConnection) url.openConnection();  
 con.setRequestMethod("GET");  
 int responseCode = con.getResponseCode();  
  
 *//Если все хорошо* if (responseCode == HttpURLConnection.*HTTP\_OK*) {  
 BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));  
 String inputLine;  
 StringBuilder response = new StringBuilder();  
 while ((inputLine = in.readLine()) != null) {  
 response.append(inputLine);  
 }  
 in.close();  
 return response.toString();  
  
 } else {  
 System.*out*.println("GET получил код " + con.getResponseCode());  
 return null;  
 }  
 } catch (Exception e) {  
 return null;  
 }  
 }  
  
 */\*\*  
 \* Отправка POST-запроса  
 \*  
 \* @param urlString - URL, куда отправляется запрос  
 \* @param paramsMap - параметры запроса  
 \* @return string  
 \*/* public static String sendPOST(String urlString, Map<String, String> paramsMap) {  
 try {  
 URL obj = new URL(urlString);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestProperty("Content-Type", "application/json; utf-8");  
 con.setRequestProperty("Accept", "application/json");  
 con.setRequestMethod("POST");  
  
 Gson gsonObj = new Gson();  
  
 String jsonStr = gsonObj.toJson(paramsMap);  
  
 con.setDoOutput(true);  
 OutputStream stream = con.getOutputStream();  
  
 stream.write(jsonStr.getBytes(StandardCharsets.UTF\_8));  
 stream.flush();  
 stream.close();  
  
 int responseCode = con.getResponseCode();  
  
 *//Успешно отработал* if (responseCode == HttpURLConnection.HTTP\_OK) {  
 BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));  
 String inputLine;  
 StringBuilder response = new StringBuilder();  
  
 while ((inputLine = in.readLine()) != null) {  
 response.append(inputLine);  
 }  
 in.close();  
 return response.toString();  
  
 } else {  
 System.out.println("POST получил код " + con.getResponseCode());  
 return null;  
 }  
 } catch (Exception e) {  
 return null;  
 }  
 }  
  
 */\*\*  
 \* Отпрвка PUT-запроса  
 \*  
 \* @param urlString - URL, куда отправляется запрос  
 \* @param jsonString - строка с JSON для обновления данных  
 \* @return string  
 \*/* public static String sendPUT(String urlString, String jsonString) {  
 try {  
 URL obj = new URL(urlString);  
 HttpURLConnection con = (HttpURLConnection) obj.openConnection();  
 con.setRequestProperty("Content-Type", "application/json; utf-8");  
 con.setRequestProperty("Accept", "application/json");  
 con.setRequestMethod("PUT");  
 con.setDoOutput(true);  
 OutputStream stream = con.getOutputStream();  
  
 stream.write(jsonString.getBytes(StandardCharsets.UTF\_8));  
 stream.flush();  
 stream.close();  
  
 int responseCode = con.getResponseCode();  
  
 *//Успешно отработал* if (responseCode == HttpURLConnection.HTTP\_OK) {  
 BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));  
 String inputLine;  
 StringBuilder response = new StringBuilder();  
  
 while ((inputLine = in.readLine()) != null) {  
 response.append(inputLine);  
 }  
 in.close();  
 return response.toString();  
  
 } else {  
 System.out.println("PUT получил код " + con.getResponseCode());  
 return null;  
 }  
 } catch (Exception e) {  
 return null;  
 }  
 }  
  
}

src/main/java/org/demka/api/MyAPI.java

package org.demka.api;  
  
  
import com.google.gson.JsonArray;  
import com.google.gson.JsonObject;  
import com.google.gson.JsonParser;  
import org.demka.App;  
import org.demka.exceptions.EmptyAPIResponseException;  
import org.demka.exceptions.FalseServerFlagException;  
import org.demka.exceptions.LongPollListenerException;  
import org.demka.exceptions.RoomNotFoundException;  
import org.demka.models.Message;  
import org.demka.models.Room;  
import org.demka.models.User;  
import org.demka.utils.String2HashUtil;  
  
import java.net.URLEncoder;  
import java.nio.charset.StandardCharsets;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
  
*/\*\*  
 \* The type My api.  
 \*/*public class MyAPI implements SuperAPI {  
  
 private final App app;  
  
 private String userName;  
 private String userKey;  
 private String userId;  
  
 private String longPollTs;  
 private String longPollSubUrl;  
 private String longPollKey;  
 private String currentRoomId;  
  
 private boolean isAuthenticated;  
  
 */\*\*  
 \* Авторизация через пару логин/пароль  
 \*  
 \* @param login - логин пользователя  
 \* @param password - пароль пользователя  
 \* @param app - экземпляр главного приложения  
 \*/* public MyAPI(String login, String password, App app) {  
 this.app = app;  
 try {  
 isAuthenticated = this.auth(login, password);  
 } catch (EmptyAPIResponseException e) {  
 e.printStackTrace();  
 }  
 *logger*.info("Инициализировали MyAPI");  
 }  
  
 */\*\*  
 \* Авторизация через ключ  
 \*  
 \* @param key - ключ API приложения  
 \* @param app - экземпляр главного приложения  
 \*/* public MyAPI(String key, App app) {  
 this.app = app;  
 try {  
 isAuthenticated = this.auth(key);  
 } catch (EmptyAPIResponseException e) {  
 e.printStackTrace();  
 }  
 logger.info("Инициализировали MyAPI");  
 }  
  
 */\*\*  
 \* Авторизация пользователя в системе  
 \* с помощью пары логин-пароль  
 \*  
 \* @param login - логин пользователя  
 \* @param password - пароль пользоваетля  
 \* @return  
 \* @throws EmptyAPIResponseException  
 \*/* @Override  
 public boolean auth(String login, String password) throws EmptyAPIResponseException {  
 login = URLEncoder.encode(login, StandardCharsets.UTF\_8);  
 password = String2HashUtil.convert(password);  
  
 String URL = String.format("%s/user/auth?login=%s&password=%s", serverURL, login, password);  
 String response = HTTPRequest.sendGET(URL);  
 if (response != null) {  
 JsonObject jsonResult = JsonParser.parseString(response).getAsJsonObject();  
 boolean authResult = jsonResult.get("result").getAsBoolean();  
 if (authResult) {  
 JsonObject userData = jsonResult.get("body").getAsJsonObject();  
 this.userId = userData.get("id").getAsString();  
 this.userKey = userData.get("key").getAsString();  
 this.userName = userData.get("name").getAsString();  
 logger.info("Auth - Авторизация с помощью пары логин/пароль прошла успешно");  
 return true;  
 }  
 logger.info("Auth - Авторизация с помощью пары логин/пароль не удалась");  
 return false;  
  
 }  
 throw new EmptyAPIResponseException(app, "Auth - получили пустой ответ от сервера");  
 }  
  
 */\*\*  
 \* Авторизация пользователя в системе  
 \* с помощью ключа API  
 \*  
 \* @param key - ключ API пользователя  
 \* @return  
 \* @throws EmptyAPIResponseException  
 \*/* @Override  
 public boolean auth(String key) throws EmptyAPIResponseException {  
 key = URLEncoder.encode(key, StandardCharsets.UTF\_8);  
  
 String URL = String.format("%s/user/auth?key=%s", serverURL, key);  
 String response = HTTPRequest.sendGET(URL);  
 if (response != null) {  
 JsonObject jsonResult = JsonParser.parseString(response).getAsJsonObject();  
 boolean authResult = jsonResult.get("result").getAsBoolean();  
 if (authResult) {  
 JsonObject userData = jsonResult.get("body").getAsJsonObject();  
 this.userId = userData.get("id").getAsString();  
 this.userKey = userData.get("key").getAsString();  
 this.userName = userData.get("name").getAsString();  
 logger.info("Auth - Авторизация с помощью ключа API прошла успешно");  
 return true;  
 }  
 logger.info("Auth - Авторизация с помощью ключа API не удалась");  
 return false;  
 }  
 throw new EmptyAPIResponseException(app, "Auth - получили пустой ответ от сервера");  
 }  
  
 */\*\*  
 \* Получение всех чат-комнат пользователя  
 \*  
 \* @return  
 \* @throws FalseServerFlagException  
 \* @throws EmptyAPIResponseException  
 \*/* @Override  
 public List<Room> getUserRooms() throws FalseServerFlagException, EmptyAPIResponseException {  
  
  
 *//Список комнат, который метод отдаёт* List<Room> resultList = new ArrayList<>();  
  
 *//Запрашиваем данные по URL* String URL = String.format("%s/room/getByUser?key=%s", serverURL, userKey);  
 String response = HTTPRequest.sendGET(URL);  
 *//Если ответ есть* if (response != null) {  
 JsonObject jsonResult = JsonParser.parseString(response).getAsJsonObject();  
 if (jsonResult.get("result").getAsBoolean()) {  
  
 JsonArray bufList = jsonResult.get("body").getAsJsonArray();  
  
 for (int i = 0; i < bufList.size(); i++) {  
 JsonObject currentRoom = bufList.get(i).getAsJsonObject();  
  
 String roomId = currentRoom.get("id").getAsString();  
 String creatorId = currentRoom.get("creatorId").getAsString();  
 String roomName = currentRoom.get("name").getAsString();  
  
 List<String> usersList = new ArrayList<>();  
 JsonArray usersArray = currentRoom.get("users").getAsJsonArray();  
 for (int j = 0; j < usersArray.size(); j++) {  
 usersList.add(usersArray.get(j).getAsString());  
 }  
  
 int timeCreated = currentRoom.get("timeCreated").getAsInt();  
 Room room = new Room(creatorId, roomName, timeCreated, usersList, roomId);  
 resultList.add(room);  
 }  
  
 logger.info("getUserRooms - Получили список комнат для пользователя " + userId);  
 return resultList;  
 } else {  
 throw new FalseServerFlagException(URL, response, "getUserRooms - Не удалось получить список комнат для пользователя " + userId);  
 }  
 } else {  
 throw new EmptyAPIResponseException(app, "getUserRooms - получили пустой ответ от сервера");  
 }  
 }  
  
 */\*\*  
 \* Получение объекта комнаты, в которой состоит пользователь, по её id  
 \*  
 \* @param roomId - идентификатор комнаты  
 \* @return  
 \* @throws RoomNotFoundException  
 \* @throws EmptyAPIResponseException  
 \*/* @Override  
 public Room getRoomInfo(String roomId) throws RoomNotFoundException, EmptyAPIResponseException {  
  
 *//Запрашиваем данные по URL* String URL = String.format("%s/room/get?roomId=%s&key=%s", serverURL, roomId, userKey);  
 String response = HTTPRequest.sendGET(URL);  
 *//Если ответ есть* if (response != null) {  
 JsonObject jsonResult = JsonParser.parseString(response).getAsJsonObject();  
 if (jsonResult.get("result").getAsBoolean()) {  
  
 JsonObject currentRoom = jsonResult.get("body").getAsJsonObject();  
 String creatorId = currentRoom.get("creatorId").getAsString();  
 String roomName = currentRoom.get("name").getAsString();  
 List<String> usersList = new ArrayList<>();  
 JsonArray usersArray = currentRoom.get("users").getAsJsonArray();  
 for (int i = 0; i < usersArray.size(); i++) {  
 usersList.add(usersArray.get(i).getAsString());  
 }  
 int timeCreated = currentRoom.get("timeCreated").getAsInt();  
 logger.info("getRoomInfo - отдали данные для комнаты с id " + roomId);  
 return new Room(creatorId, roomName, timeCreated, usersList, roomId);  
 } else {  
 throw new RoomNotFoundException("getRoomInfo - Запрашиваемой комнаты с id " + roomId + " не существует");  
 }  
 } else {  
 throw new EmptyAPIResponseException(app, "getRoomInfo - получили пустой ответ от сервера");  
 }  
  
 }  
  
 */\*\*  
 \* Получение истории сообщений по конкретной комнате  
 \*  
 \* @param roomId - идентификатор комнаты  
 \* @return  
 \* @throws FalseServerFlagException  
 \* @throws EmptyAPIResponseException  
 \*/* @Override  
 public List<Message> getRoomMessagesHistory(String roomId) throws FalseServerFlagException, EmptyAPIResponseException {  
 *//Список сообщений, который метод отдаёт* List<Message> resultList = new ArrayList<>();  
  
 *//Запрашиваем данные по URL* String URL = String.format("%s/messages/get?roomId=%s&key=%s", serverURL, roomId, userKey);  
 String response = HTTPRequest.sendGET(URL);  
 *//Если ответ есть* if (response != null) {  
 JsonObject jsonResult = JsonParser.parseString(response).getAsJsonObject();  
 if (jsonResult.get("result").getAsBoolean()) {  
  
 JsonArray bufList = jsonResult.get("body").getAsJsonArray();  
  
 for (int i = 0; i < bufList.size(); i++) {  
 JsonObject currentMessage = bufList.get(i).getAsJsonObject();  
  
 String messageId = currentMessage.get("id").getAsString();  
 String messageRoom = currentMessage.get("roomId").getAsString();  
 String messageText = currentMessage.get("text").getAsString();  
 int messageTimeCreated = currentMessage.get("timeCreated").getAsInt();  
 String messageUserId = currentMessage.get("userId").getAsString();  
 String messageUserName = currentMessage.get("userName").getAsString();  
  
 Message bufMessage = new Message(messageUserId, messageUserName, messageText, messageRoom, messageTimeCreated, messageId);  
 resultList.add(bufMessage);  
 }  
 logger.info("getRoomMessagesHistory - Получили список сообщений для комнаты " + roomId);  
 return resultList;  
 } else {  
 throw new FalseServerFlagException(URL, response, "getRoomMessagesHistory - Не удалось получить список сообщений для комнаты " + roomId);  
 }  
 } else {  
 throw new EmptyAPIResponseException(app, "getRoomMessagesHistory - Получили пустой ответ от сервера");  
 }  
 }  
  
 */\*\*  
 \* Создание комнаты  
 \*  
 \* @param roomName - название комнаты  
 \* @param usersString - строка с идентификаторами пользователей-участинков комнаты  
 \* @return  
 \* @throws FalseServerFlagException  
 \* @throws EmptyAPIResponseException  
 \*/* @Override  
 public boolean createRoom(String roomName, String usersString) throws FalseServerFlagException, EmptyAPIResponseException {  
 *//Запрашиваем данные по URL* String URL = String.format("%s/room/create", serverURL);  
  
 Map<String, String> params = new HashMap<>();  
 params.put("users", usersString);  
 params.put("roomName", roomName);  
 params.put("key", userKey);  
  
 String response = HTTPRequest.sendPOST(URL, params);  
  
 *//Если ответ есть* if (response != null) {  
 JsonObject jsonResult = JsonParser.parseString(response).getAsJsonObject();  
 if (jsonResult.get("result").getAsBoolean()) {  
 logger.info("createRoom - Успешно создали новую комнату");  
 return true;  
 } else {  
 throw new FalseServerFlagException(URL, response, "createRoom - Не удалось создать новую комнату");  
 }  
 } else {  
 throw new EmptyAPIResponseException(app, "createRoom - Получили пустой ответ от сервера");  
 }  
 }  
  
 */\*\*  
 \* Поиск пользователей в системе по имени  
 \*  
 \* @param searchExp - паттерн имени пользователя  
 \* @return  
 \* @throws FalseServerFlagException  
 \* @throws EmptyAPIResponseException  
 \*/* @Override  
 public List<User> getUsers(String searchExp) throws FalseServerFlagException, EmptyAPIResponseException {  
  
 List<User> resultList = new ArrayList<>();  
 String URL;  
 if (searchExp == null) {  
 URL = String.format("%s/user/search?key=%s", serverURL, userKey);  
 } else {  
 searchExp = URLEncoder.encode(searchExp, StandardCharsets.UTF\_8);  
 URL = String.format("%s/user/search?key=%s&searchName=%s", serverURL, userKey, searchExp);  
 }  
 String response = HTTPRequest.sendGET(URL);  
 *//Если ответ есть* if (response != null) {  
 JsonObject jsonResult = JsonParser.parseString(response).getAsJsonObject();  
 if (jsonResult.get("result").getAsBoolean()) {  
  
 JsonArray bufList = jsonResult.get("body").getAsJsonArray();  
 for (int i = 0; i < bufList.size(); i++) {  
 JsonObject currentUser = bufList.get(i).getAsJsonObject();  
  
 String userId = currentUser.get("id").getAsString();  
 String userName = currentUser.get("name").getAsString();  
 Integer userTimeCreated = currentUser.get("timeCreated").getAsInt();  
  
 User bufUser = new User(userId, userName, userTimeCreated);  
 resultList.add(bufUser);  
 }  
 logger.info("getUsers - получили " + resultList.size() + " пользователей");  
 return resultList;  
 } else {  
 throw new FalseServerFlagException(URL, response, "getUsers - не удалось получить пользователей");  
 }  
 } else {  
 throw new EmptyAPIResponseException(app, "getUsers - получили пустой ответ от сервера");  
 }  
 }  
  
 */\*\*  
 \* Получение объектов пользователей по id комнаты, в которой они состоят  
 \*  
 \* @param roomId - идентификатор комнаты  
 \* @return  
 \* @throws FalseServerFlagException  
 \* @throws EmptyAPIResponseException  
 \*/* @Override  
 public List<User> getUsersByRoom(String roomId) throws FalseServerFlagException, EmptyAPIResponseException {  
  
 List<User> resultList = new ArrayList<>();  
 String URL = String.format("%s/room/getUsers?key=%s&roomId=%s", serverURL, userKey, roomId);  
 String response = HTTPRequest.sendGET(URL);  
 *//Если ответ есть* if (response != null) {  
 JsonObject jsonResult = JsonParser.parseString(response).getAsJsonObject();  
 if (jsonResult.get("result").getAsBoolean()) {  
  
 JsonArray bufList = jsonResult.get("body").getAsJsonArray();  
 for (int i = 0; i < bufList.size(); i++) {  
 JsonObject currentUser = bufList.get(i).getAsJsonObject();  
 String userId = currentUser.get("id").getAsString();  
 String userName = currentUser.get("name").getAsString();  
 Integer userTimeCreated = currentUser.get("timeCreated").getAsInt();  
 User bufUser = new User(userId, userName, userTimeCreated);  
 resultList.add(bufUser);  
 }  
 logger.info("getUsersByRoom - получили " + resultList.size() + " пользователей");  
 return resultList;  
 } else {  
 throw new FalseServerFlagException(URL, response, "getUsersByRoom - Не удалось получить список пользоватлеей для комнаты");  
 }  
 } else {  
 throw new EmptyAPIResponseException(app, "getUsersByRoom - получили пустой ответ от сервера");  
 }  
 }  
  
 */\*\*  
 \* Отправка сообщения в текущую комнату  
 \*  
 \* @param text - текст сообщения  
 \* @return  
 \* @throws FalseServerFlagException  
 \* @throws EmptyAPIResponseException  
 \*/* @Override  
 public Message writeMessage(String text) throws FalseServerFlagException, EmptyAPIResponseException {  
  
 *//Запрашиваем данные по URL* String URL = String.format("%s/messages/send", serverURL);  
  
 Map<String, String> params = new HashMap<>();  
 params.put("key", userKey);  
 params.put("text", text);  
 params.put("roomId", currentRoomId);  
  
 String response = HTTPRequest.sendPOST(URL, params);  
 *//Если ответ есть* if (response != null) {  
 JsonObject jsonResult = JsonParser.parseString(response).getAsJsonObject();  
 if (jsonResult.get("result").getAsBoolean()) {  
  
 JsonObject newMessageJsonObject = jsonResult.get("body").getAsJsonObject();  
  
 String messageId = newMessageJsonObject.get("id").getAsString();  
 String roomId = newMessageJsonObject.get("roomId").getAsString();  
 String messageText = newMessageJsonObject.get("text").getAsString();  
 int timeCreated = newMessageJsonObject.get("timeCreated").getAsInt();  
 String messageUserId = newMessageJsonObject.get("userId").getAsString();  
 String messageUserName = newMessageJsonObject.get("userName").getAsString();  
  
 logger.info("writeMessage - Отправили новое сообщение " + messageText + " " + messageId);  
 return new Message(messageUserId, messageUserName, messageText, roomId, timeCreated, messageId);  
 } else {  
 throw new FalseServerFlagException(URL, response, "writeMessage - Не удалось отправить новое сообщение");  
 }  
  
 } else {  
 throw new EmptyAPIResponseException(app, "writeMessage - получили пустой ответ от сервера");  
 }  
  
 }  
  
 */\*\*  
 \* Получение сервера лонгпула  
 \* Выставляет longPollTs, longPollSubUrl, longPollKey  
 \*  
 \* @throws EmptyAPIResponseException  
 \* @throws FalseServerFlagException  
 \*/* @Override  
 public void getLongPollServer() throws EmptyAPIResponseException, FalseServerFlagException {  
 *//Запрашиваем данные по URL* String URL = String.format("%s/longpoll/getServer?key=%s", serverURL, userKey);  
 String response = HTTPRequest.sendGET(URL);  
 *//Если ответ есть* if (response != null) {  
 JsonObject jsonResult = JsonParser.parseString(response).getAsJsonObject();  
 if (jsonResult.get("result").getAsBoolean()) {  
  
 JsonObject credentials = jsonResult.get("body").getAsJsonObject();  
  
 this.longPollTs = credentials.get("ts").getAsString();  
 this.longPollSubUrl = credentials.get("url").getAsString();  
 this.longPollKey = credentials.get("key").getAsString();  
 logger.info("getLongPollServer - получили конфиг, инициализировались");  
 } else {  
 throw new FalseServerFlagException(URL, response, "getLongPollServer - Не удалось получить конфиг, не инициализировались");  
 }  
 } else {  
 throw new EmptyAPIResponseException(app, "getLongPollServer - Получили пустой ответ от сервера");  
 }  
 }  
  
 */\*\*  
 \* Слушатель лонгпула  
 \*  
 \* @return  
 \* @throws LongPollListenerException  
 \*/* @Override  
 public List<Message> longPollListener() throws LongPollListenerException {  
  
 *//Если лонгпул не инициализирован* if (longPollKey == null || longPollTs == null || longPollSubUrl == null) {  
 logger.info("longPollListener - LongPool не инициалзирован");  
 throw new LongPollListenerException("[Не пугаться] Лонгпул не был иницилизирован. Использую getLongPollServer для иницализации..");  
 }  
  
 *//Если же инициализация прошла успешно* else {  
  
 *//Список результатов сообщений* List<Message> resultList = new ArrayList<>();  
  
 *//Запрашиваем данные по URL* String URL = String.format("%s/longpoll/updates/%s?key=%s&ts=%s", serverURL, longPollSubUrl, longPollKey, longPollTs);  
 logger.info("longPollListener - отправили запрос, ожидаем новые сообщения..");  
 String response = HTTPRequest.sendGET(URL);  
  
 *//Если ответ есть* if (response != null) {  
 JsonObject jsonResult = JsonParser.parseString(response).getAsJsonObject();  
 if (jsonResult.get("result").getAsBoolean()) {  
  
 *//Данные, которые получили* JsonObject newData = jsonResult.get("body").getAsJsonObject();  
 *//Обновляем ts* this.longPollTs = newData.get("ts").getAsString();  
  
 *//Работем с сообщениями* JsonArray newMessages = newData.get("updates").getAsJsonArray();  
 for (int i = 0; i < newMessages.size(); i++) {  
  
 JsonObject currentMessage = newMessages.get(i).getAsJsonObject();  
  
 String messageId = currentMessage.get("id").getAsString();  
 String messageRoom = currentMessage.get("roomId").getAsString();  
 String messageText = currentMessage.get("text").getAsString();  
 int messageTimeCreated = currentMessage.get("timeCreated").getAsInt();  
 String messageUserId = currentMessage.get("userId").getAsString();  
 String messageUserName = currentMessage.get("userName").getAsString();  
  
 Message bufMessage = new Message(messageUserId, messageUserName, messageText, messageRoom, messageTimeCreated, messageId);  
 resultList.add(bufMessage);  
 }  
 logger.info("longPollListener - получили новые сообщения");  
 } else {  
 throw new LongPollListenerException("Ответ result = false. Нужна повторная авторизация (или же мы передали что-то некорректно");  
 }  
 } else {  
 throw new LongPollListenerException("Лонгпул ничего не получил от сервера. Что-то сломалось");  
 }  
  
 return resultList;  
 }  
 }  
  
 */\*\*  
 \* Gets is authenticated.  
 \*  
 \* @return the is authenticated  
 \*/* public boolean getIsAuthenticated() {  
 return isAuthenticated;  
 }  
  
 */\*\*  
 \* Gets current room id.  
 \*  
 \* @return the current room id  
 \*/* public String getCurrentRoomId() {  
 return currentRoomId;  
 }  
  
 */\*\*  
 \* Sets current room id.  
 \*  
 \* @param currentRoomId the current room id  
 \*/* public void setCurrentRoomId(String currentRoomId) {  
 this.currentRoomId = currentRoomId;  
 }  
  
 */\*\*  
 \* Gets user name.  
 \*  
 \* @return the user name  
 \*/* public String getUserName() {  
 return userName;  
 }  
  
 */\*\*  
 \* Gets user key.  
 \*  
 \* @return the user key  
 \*/* public String getUserKey() {  
 return userKey;  
 }  
}

src/main/java/org/demka/api/SuperAPI.java

package org.demka.api;  
  
import com.google.gson.Gson;  
import com.google.gson.JsonObject;  
import com.google.gson.JsonParser;  
import org.demka.exceptions.EmptyAPIResponseException;  
import org.demka.exceptions.FalseServerFlagException;  
import org.demka.exceptions.LongPollListenerException;  
import org.demka.exceptions.RoomNotFoundException;  
import org.demka.models.Message;  
import org.demka.models.Room;  
import org.demka.models.User;  
import org.demka.utils.String2HashUtil;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
*/\*\*  
 \* The interface Super api.  
 \*/*public interface SuperAPI {  
  
 */\*\*  
 \* The constant logger.  
 \*/* Logger *logger* = LoggerFactory.*getLogger*(MyAPI.class);  
 */\*\*  
 \* The constant serverURL.  
 \*/* String *serverURL* = "http://149.248.54.195:8080";  
  
 */\*\*  
 \* Восстановление пароля  
 \*  
 \* @param login - логин пользователя  
 \* @param newPassword - новый пароль пользователя  
 \* @param masterKey - мастер-ключ пользователя  
 \* @return map  
 \*/* static Map<String, Object> resetPassword(String login, String newPassword, String masterKey) {  
 String URL = String.*format*("%s/user/reset", *serverURL*);  
 Map<String, String> params = new HashMap<>();  
 params.put("email", login);  
 params.put("newPassword", String2HashUtil.*convert*(newPassword));  
 params.put("masterKey", String2HashUtil.*convert*(masterKey));  
  
 Gson gson = new Gson();  
 String jsonString = gson.toJson(params);  
 String response = HTTPRequest.*sendPUT*(URL, jsonString);  
 Map<String, Object> resultMap = new HashMap<>();  
  
 if (response == null) {  
 resultMap.put("result", false);  
 resultMap.put("error", "Нет ответа от сервера");  
 *logger*.error("Не получили ответ от сервера во время смены пароля пользователя с логином " + login);  
 return resultMap;  
 }  
  
 *//Парсим результат* JsonObject jsonResult = JsonParser.*parseString*(response).getAsJsonObject();  
 Boolean authResult = jsonResult.get("result").getAsBoolean();  
 resultMap.put("result", authResult);  
  
 if (!authResult) {  
 String errDescription = jsonResult.get("description").getAsString();  
 resultMap.put("error", errDescription);  
 *logger*.info("Сервер возвратил ошибку при смене пароля: " + errDescription);  
 return resultMap;  
 }  
 *logger*.info("Пользователь " + login + " успешно сменил пароль");  
 return resultMap;  
 }  
  
 */\*\*  
 \* Регистрация пользователя в системе  
 \*  
 \* @param name - ник  
 \* @param login - логин (e-mail) пользователя  
 \* @param password - пароль пользователя  
 \* @param masterKey - мастер-пароль пользователя  
 \* @return map  
 \*/* static Map<String, Object> registration(String name, String login, String password, String masterKey) {  
  
 String URL = String.*format*("%s/user/register", *serverURL*);  
 Map<String, String> params = new HashMap<>();  
 params.put("login", login);  
 params.put("username", name);  
 params.put("password", String2HashUtil.*convert*(password));  
 params.put("masterKey", String2HashUtil.*convert*(masterKey));  
 String response = HTTPRequest.*sendPOST*(URL, params);  
 Map<String, Object> resultMap = new HashMap<>();  
  
 if (response == null) {  
 resultMap.put("result", false);  
 resultMap.put("error", "Нет ответа от сервера");  
 *logger*.error("Не получили ответ от сервера во время регистрации пользователя " + name);  
 return resultMap;  
 }  
  
 *//Парсим результат* JsonObject jsonResult = JsonParser.*parseString*(response).getAsJsonObject();  
 Boolean authResult = jsonResult.get("result").getAsBoolean();  
 resultMap.put("result", authResult);  
  
 if (!authResult) {  
 String errDescription = jsonResult.get("description").getAsString();  
 resultMap.put("error", errDescription);  
 *logger*.info("Сервер возвратил ошибку регистрации: " + errDescription);  
 return resultMap;  
 }  
  
 *logger*.info("Пользователь " + name + " успешно зарегистрировался");  
 return resultMap;  
 }  
  
 */\*\*  
 \* Авторизация пользователя в системе  
 \* с помощью пары логин-пароль  
 \*  
 \* @param login - логин пользователя  
 \* @param password - пароль пользоваетля  
 \* @return boolean  
 \* @throws EmptyAPIResponseException the empty api response exception  
 \*/* boolean auth(String login, String password) throws EmptyAPIResponseException;  
  
 */\*\*  
 \* Авторизация пользователя в системе  
 \* с помощью ключа API  
 \*  
 \* @param key - ключ API пользователя  
 \* @return boolean  
 \* @throws EmptyAPIResponseException the empty api response exception  
 \*/* boolean auth(String key) throws EmptyAPIResponseException;  
  
 */\*\*  
 \* Получение всех чат-комнат пользователя  
 \*  
 \* @return user rooms  
 \* @throws FalseServerFlagException the false server flag exception  
 \* @throws EmptyAPIResponseException the empty api response exception  
 \*/* List<Room> getUserRooms() throws FalseServerFlagException, EmptyAPIResponseException;  
  
  
 */\*\*  
 \* Получение объекта комнаты, в которой состоит пользователь, по её id  
 \*  
 \* @param roomId - идентификатор комнаты  
 \* @return room info  
 \* @throws RoomNotFoundException the room not found exception  
 \* @throws EmptyAPIResponseException the empty api response exception  
 \*/* Room getRoomInfo(String roomId) throws RoomNotFoundException, EmptyAPIResponseException;  
  
 */\*\*  
 \* Получение истории сообщений по конкретной комнате  
 \*  
 \* @param roomId - идентификатор комнаты  
 \* @return room messages history  
 \* @throws FalseServerFlagException the false server flag exception  
 \* @throws EmptyAPIResponseException the empty api response exception  
 \*/* List<Message> getRoomMessagesHistory(String roomId) throws FalseServerFlagException, EmptyAPIResponseException;  
  
  
 */\*\*  
 \* Создание комнаты  
 \*  
 \* @param roomName - название комнаты  
 \* @param usersString - строка с идентификаторами пользователей-участинков комнаты  
 \* @return boolean  
 \* @throws FalseServerFlagException the false server flag exception  
 \* @throws EmptyAPIResponseException the empty api response exception  
 \*/* boolean createRoom(String roomName, String usersString) throws FalseServerFlagException, EmptyAPIResponseException;  
  
 */\*\*  
 \* Поиск пользователей в системе по имени  
 \*  
 \* @param searchExp - паттерн имени пользователя  
 \* @return users  
 \* @throws FalseServerFlagException the false server flag exception  
 \* @throws EmptyAPIResponseException the empty api response exception  
 \*/* List<User> getUsers(String searchExp) throws FalseServerFlagException, EmptyAPIResponseException;  
  
  
 */\*\*  
 \* Получение объектов пользователей по id комнаты, в которой они состоят  
 \*  
 \* @param roomId - идентификатор комнаты  
 \* @return users by room  
 \* @throws FalseServerFlagException the false server flag exception  
 \* @throws EmptyAPIResponseException the empty api response exception  
 \*/* List<User> getUsersByRoom(String roomId) throws FalseServerFlagException, EmptyAPIResponseException;  
  
 */\*\*  
 \* Отправка сообщения в текущую комнату  
 \*  
 \* @param text - текст сообщения  
 \* @return message  
 \* @throws FalseServerFlagException the false server flag exception  
 \* @throws EmptyAPIResponseException the empty api response exception  
 \*/* Message writeMessage(String text) throws FalseServerFlagException, EmptyAPIResponseException;  
  
 */\*\*  
 \* Получение сервера лонгпула  
 \* Выставляет longpollTs, longpollSubUrl, longpollKey  
 \*  
 \* @throws EmptyAPIResponseException the empty api response exception  
 \* @throws FalseServerFlagException the false server flag exception  
 \*/* void getLongPollServer() throws EmptyAPIResponseException, FalseServerFlagException;  
  
 */\*\*  
 \* Слушатель лонгпула  
 \*  
 \* @return list  
 \* @throws LongPollListenerException the long poll listener exception  
 \*/* List<Message> longPollListener() throws LongPollListenerException;  
}

src/main/java/org/demka/controllers/AboutMeController.java

package org.demka.controllers;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.Hyperlink;  
import javafx.stage.Stage;  
import org.demka.App;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.awt.\*;  
import java.io.IOException;  
import java.net.URI;  
import java.net.URISyntaxException;  
  
*/\*\*  
 \* Контроллер-обработчик окна "О программе/об авторе"  
 \*/*public class AboutMeController extends SuperPartController {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(AboutMeController.class);  
  
 @FXML  
 private Hyperlink siteLink;  
  
 */\*\*  
 \* Метод инициализации (вызывается с Main)  
 \*  
 \* @param app  
 \* @param dialogStage  
 \*/* @Override  
 public void initialize(App app, Stage dialogStage) {  
 this.app = app;  
 this.dialogStage = dialogStage;  
 }  
  
 */\*\*  
 \* Нажатие на button выхода из текущего окна  
 \*/* @FXML  
 private void okButtonClicked() {  
 *logger*.info("Выход из подменю с информацией об авторе и программе");  
 dialogStage.close();  
 }  
  
 */\*\*  
 \* Нажатие на ссылку "Об авторе"  
 \*/* @FXML  
 private void siteLinkClicked() {  
 String linkText = siteLink.getText();  
 *logger*.info("Нажатие на ссылку " + linkText);  
 if (Desktop.*isDesktopSupported*()) {  
 try {  
 Desktop.*getDesktop*().browse(new URI(linkText));  
 *logger*.info("Открыли ссылку " + linkText + " в браузере");  
 } catch (IOException | URISyntaxException e1) {  
 e1.printStackTrace();  
 }  
 }  
 }  
  
}

src/main/java/org/demka/controllers/ConnectionErrorController.java

package org.demka.controllers;  
  
import org.demka.App;  
  
*/\*\*  
 \* Контроллер, возникающий при ошибке подключения к серверу (или получения от него пустого ответа)  
 \*/*public class ConnectionErrorController extends SuperFullController {  
  
 */\*\*  
 \* The constant isActive.  
 \*/* public static boolean *isActive* = false;  
  
 */\*\*  
 \* Метод инициализации (вызывается с Main)  
 \*  
 \* @param app  
 \*/* @Override  
 public void initialize(App app) {  
 this.app = app;  
 *isActive* = true;  
 }  
}

src/main/java/org/demka/controllers/CreateNewRoomController.java

package org.demka.controllers;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import javafx.stage.Stage;  
import org.demka.App;  
import org.demka.api.MyAPI;  
import org.demka.exceptions.EmptyAPIResponseException;  
import org.demka.exceptions.FalseServerFlagException;  
import org.demka.models.User;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Контроллер-обработчик создания новой чат-комнаты  
 \*/*public class CreateNewRoomController extends SuperPartController {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(CreateNewRoomController.class);  
 private final ObservableList<User> allUsersData = FXCollections.*observableArrayList*();  
 private final ObservableList<User> chatUsersData = FXCollections.*observableArrayList*();  
 private MyAPI myAPI;  
 @FXML  
 private TableView<User> allUsersTable;  
 @FXML  
 private TableColumn<User, String> allUsersColumn;  
 @FXML  
 private TableView<User> chatUsersTable;  
 @FXML  
 private TableColumn<User, String> chatUsersColumn;  
 @FXML  
 private Button createRoomButton;  
 @FXML  
 private TextField chatName;  
 @FXML  
 private TextField searchUser;  
  
 @Override  
 public void initialize(App app, Stage dialogStage) {  
 this.app = app;  
 this.dialogStage = dialogStage;  
  
 *//Инициализация таблиц* allUsersTable.setItems(allUsersData);  
 chatUsersTable.setItems(chatUsersData);  
  
 *//API, через которое взаимодействуем с миром* myAPI = app.getMyAPI();  
  
 *//Ресайз, зависящий от данных* allUsersTable.setColumnResizePolicy(TableView.*CONSTRAINED\_RESIZE\_POLICY*);  
 chatUsersTable.setColumnResizePolicy(TableView.*CONSTRAINED\_RESIZE\_POLICY*);  
  
 *//Отображение имен пользователей* allUsersColumn.setCellValueFactory(cellData -> cellData.getValue().getNameProperty());  
 chatUsersColumn.setCellValueFactory(cellData -> cellData.getValue().getNameProperty());  
  
 allUsersTable.getSelectionModel().setSelectionMode(SelectionMode.*MULTIPLE*);  
 chatUsersTable.getSelectionModel().setSelectionMode(SelectionMode.*MULTIPLE*);  
  
 *//Добавляем пользователей системы* try {  
 allUsersData.addAll(myAPI.getUsers(null));  
 } catch (FalseServerFlagException | EmptyAPIResponseException e) {  
 e.printStackTrace();  
 }  
  
 *//Слушатель изменения текста* searchUser.textProperty().addListener(  
 ((observableValue, oldValue, newValue) -> searchUserChanged(newValue))  
 );  
 }  
  
 */\*\*  
 \* Обработчик изменения текста в searchUser  
 \*  
 \* @param newValue - обновлённое значение поля  
 \*/* private void searchUserChanged(String newValue) {  
 try {  
 List<User> searchResult = myAPI.getUsers(newValue);  
 allUsersData.clear();  
  
 *//Фильтруем на то, чтоб не было пользователей, которых мы уже выбрали* for (User searchUser : searchResult) {  
  
 boolean contains = false;  
 for (User chatUser : chatUsersData) {  
 if (searchUser.getId().equals(chatUser.getId())) {  
 contains = true;  
 break;  
 }  
 }  
 if (!contains) {  
 allUsersData.add(searchUser);  
 }  
 }  
 } catch (FalseServerFlagException | EmptyAPIResponseException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Кнопка выхода из окна  
 \*/* @FXML  
 private void cancelButtonClicked() {  
 dialogStage.close();  
 }  
  
 */\*\*  
 \* Обработчик нажатия на кнопку создания комнаты  
 \*/* @FXML  
 public void createRoomButtonClicked() {  
 String localeChatName;  
 StringBuilder usersIds = new StringBuilder();  
 *//Если не было задано название конфы, то генерируем ее из имен* if (chatName.getText().equals("")) {  
  
 StringBuilder ChatNameBuilder = new StringBuilder();  
 ChatNameBuilder.append("Чат с ");  
 for (User user : chatUsersData) {  
 ChatNameBuilder.append(user.getName()).append(", ");  
 }  
 *//Удаляем последнюю запятую* ChatNameBuilder.setLength(ChatNameBuilder.length() - 2);  
 *//Если длина слишком большая - сокращаем* if (ChatNameBuilder.length() > 100) {  
 ChatNameBuilder.setLength(100);  
 ChatNameBuilder.append("..");  
 }  
 localeChatName = ChatNameBuilder.toString();  
 } else {  
 localeChatName = chatName.getText();  
 }  
  
 *//Добавляем id всех пользователей* for (User user : chatUsersData) {  
 usersIds.append(user.getId());  
 usersIds.append(",");  
 }  
 *//Удаляем последнюю запятую* usersIds.setLength(usersIds.length() - 1);  
  
 *//Создаем комнату* try {  
 myAPI.createRoom(localeChatName, usersIds.toString());  
 } catch (FalseServerFlagException | EmptyAPIResponseException e) {  
 e.printStackTrace();  
 }  
  
 dialogStage.close();  
 }  
  
 */\*\*  
 \* Добавление пользователя в чат  
 \*/* @FXML  
 private void addUserToChat() {  
  
 ObservableList<User> selectedItems = allUsersTable.getSelectionModel().getSelectedItems();  
 *//Буферный список, который не изменяется* List<User> selectedUsers = new ArrayList<>(selectedItems);  
  
 for (User user : selectedUsers) {  
 *logger*.info("addUserToChat - добавили пользователя " + user.getName());  
 allUsersData.remove(user);  
 chatUsersData.add(user);  
 }  
  
 createRoomButton.setDisable(chatUsersData.size() <= 0);  
  
 }  
  
 */\*\*  
 \* Удаление пользователя из чата  
 \*/* @FXML  
 private void removeUserFromChat() {  
  
 ObservableList<User> selectedItems = chatUsersTable.getSelectionModel().getSelectedItems();  
 *//Буферный список, который не изменяется* List<User> selectedUsers = new ArrayList<>(selectedItems);  
  
 for (User user : selectedUsers) {  
 *logger*.info("addUserToChat - удалили пользователя " + user.getName());  
 chatUsersData.remove(user);  
 allUsersData.add(user);  
 }  
  
 createRoomButton.setDisable(chatUsersData.size() <= 0);  
 }  
}

src/main/java/org/demka/controllers/ForgotPasswordController.java

package org.demka.controllers;  
  
import com.jfoenix.controls.JFXPasswordField;  
import com.jfoenix.controls.JFXTextField;  
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
import org.demka.App;  
import org.demka.api.SuperAPI;  
import org.demka.utils.ValidatorsUtil;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.util.Map;  
  
*/\*\*  
 \* Контроллер-обработчик формы восстановления пароля  
 \*/*public class ForgotPasswordController extends SuperFullController {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(ForgotPasswordController.class);  
 */\*\*  
 \* The New password text field.  
 \*/* @FXML  
 JFXPasswordField newPasswordTextField;  
 */\*\*  
 \* The Master key text field.  
 \*/* @FXML  
 JFXPasswordField masterKeyTextField;  
 */\*\*  
 \* The Login text field.  
 \*/* @FXML  
 JFXTextField loginTextField;  
 */\*\*  
 \* The Error description.  
 \*/* @FXML  
 Label errorDescription;  
 private String errorString;  
  
 */\*\*  
 \* Метод инициализации (вызывается с Main)  
 \*  
 \* @param app  
 \*/* @Override  
 public void initialize(App app) {  
 errorDescription.setOpacity(0.0);  
 this.app = app;  
 }  
  
 */\*\*  
 \* Кнопка выхода из смены пароля  
 \*/* @FXML  
 private void backButtonClicked() {  
 app.UserAuthorisation();  
 *logger*.info("Выход из формы смены пароля");  
 }  
  
 */\*\*  
 \* Кнопка сброса пароля  
 \*/* @FXML  
 private void resetButtonClicked() {  
 *logger*.info("Нажатие на button сброса пароля");  
  
 String login = loginTextField.getText();  
 String newPassword = newPasswordTextField.getText();  
 String masterKey = masterKeyTextField.getText();  
  
 if (dataValidator(login, newPassword, masterKey)) {  
 *logger*.info("Валидация данных со стороны клиента прошла успешно");  
 *//Получаем ответ от сервера* Map<String, Object> regResult = SuperAPI.*resetPassword*(login, newPassword, masterKey);  
 if ((boolean) regResult.get("result")) {  
 errorDescription.setOpacity(0.0);  
 errorDescription.setText("");  
 *logger*.info("Успешно сменили пароль пользователя");  
 app.SuccessUserAction("Успешная смена пароля", "Успешно сменили пароль!");  
 } else {  
 errorString = (String) regResult.get("error");  
 errorDescription.setText(errorString);  
 *logger*.info("Валидация со стороны сервера не прошла: " + errorString);  
 errorDescription.setOpacity(1.0);  
 }  
  
 } else {  
 *logger*.info("Валидация данных со стороны клиента не прошла: " + errorString);  
 errorDescription.setText(errorString);  
 errorDescription.setOpacity(1.0);  
 }  
  
  
 }  
  
 */\*\*  
 \* Фильтрация входных данных для смены пароля  
 \*  
 \* @param email - e-mail пользвателя  
 \* @param password - новый пароль пользователя  
 \* @param masterKey - мастер-ключ пользователя  
 \* @return  
 \*/* private Boolean dataValidator(String email, String password, String masterKey) {  
  
 if ((email == null) || (email.equals(""))) {  
 errorString = "E-mail не может быть пустым";  
 return false;  
 }  
  
 if ((password == null) || (password.equals(""))) {  
 errorString = "Новый пароль не может быть пустым";  
 return false;  
 }  
  
 if ((masterKey == null) || (masterKey.equals(""))) {  
 errorString = "Мастер-пароль не может быть пустым";  
 return false;  
 }  
  
 if (!ValidatorsUtil.*emailValidator*(email)) {  
 errorString = "Некорректный e-mail";  
 return false;  
 }  
  
 if (password.length() < 8) {  
 errorString = "Пароль должен быть не менее 8 символов";  
 return false;  
 }  
  
 if (masterKey.length() < 8) {  
 errorString = "Мастер-пароль должен быть не менее 8 символов";  
 return false;  
 }  
  
 if (masterKey.equals(password)) {  
 errorString = "Пароль и мастер-пароль не должны совпадать";  
 return false;  
 }  
  
 return true;  
 }  
}

src/main/java/org/demka/controllers/LoginController.java

package org.demka.controllers;  
  
import com.jfoenix.controls.JFXCheckBox;  
import com.jfoenix.controls.JFXPasswordField;  
import com.jfoenix.controls.JFXTextField;  
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
import org.demka.App;  
import org.demka.api.MyAPI;  
import org.demka.utils.AuthUtil;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
*/\*\*  
 \* Контроллер-обработчик формы авторизации  
 \*/*public class LoginController extends SuperFullController {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(LoginController.class);  
  
 @FXML  
 private Label wrongAuth;  
 @FXML  
 private JFXTextField loginTextField;  
 @FXML  
 private JFXPasswordField passwordTextField;  
 @FXML  
 private JFXCheckBox autoLoginCheckBox;  
  
 */\*\*  
 \* Нажатие на кнопку авторизации в программе  
 \*/* @FXML  
 public void mainButtonClick() {  
  
 *logger*.info("Нажатие на button авторизации в программе");  
 String login = loginTextField.getText();  
 String password = passwordTextField.getText();  
  
 if ((login.equals("")) || (password.equals(""))) {  
 *logger*.info("Попытка входа без ввода данных");  
 } else {  
 *//Пытаемся авторизоваться* MyAPI bufSession = new MyAPI(login, password, this.app);  
 *//Если удалось произвести авторизацию* if (bufSession.getIsAuthenticated()) {  
 app.setMyAPI(bufSession);  
 *//Если пользователь выбрал "Запомнить меня"* if (autoLoginCheckBox.isSelected()) {  
 AuthUtil authUtil = app.getAuthUtil();  
 authUtil.writeKey(bufSession.getUserKey());  
 }  
  
 app.getPrimaryStage().setTitle("DEMKAChat - Сообщения [" + bufSession.getUserName() + "]");  
 app.MainChat();  
 *logger*.info("Пользователь успешно авторизовался");  
 } else {  
 wrongAuth.setOpacity(1);  
 *logger*.info("Не удалось авторизоваться");  
 }  
  
  
 }  
  
 }  
  
 */\*\*  
 \* Нажатие на ссылку перехода на форму регистрации  
 \*/* @FXML  
 public void regLinkClicked() {  
 app.UserRegistration();  
 *logger*.info("Переход на форму регистрации");  
 }  
  
 */\*\*  
 \* Нажатие на ссылку восстановления пароля  
 \*/* @FXML  
 public void forgotPasswordLinkClicked() {  
 app.ForgotPassword();  
 *logger*.info("Переход на форму восстановления пароля");  
 }  
  
 */\*\*  
 \* Метод инициализации (вызывается с Main)  
 \*  
 \* @param app  
 \*/* @Override  
 public void initialize(App app) {  
 this.app = app;  
 }  
}

src/main/java/org/demka/controllers/MainChatController.java

package org.demka.controllers;  
  
import com.jfoenix.controls.JFXButton;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import org.demka.App;  
import org.demka.api.MyAPI;  
import org.demka.exceptions.EmptyAPIResponseException;  
import org.demka.exceptions.FalseServerFlagException;  
import org.demka.models.Message;  
import org.demka.models.Room;  
import org.demka.runnable.CheckInternetRunnable;  
import org.demka.runnable.LongPollRunnable;  
import org.demka.runnable.RunnableManager;  
import org.demka.utils.MyMenuActionUtil;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.util.Comparator;  
import java.util.List;  
  
*/\*\*  
 \* Контроллер-обработчик главного окна чата  
 \*/*public class MainChatController extends SuperFullController {  
 private static final Logger *logger* = LoggerFactory.*getLogger*(MainChatController.class);  
 private final ObservableList<Room> roomData = FXCollections.*observableArrayList*();  
 private final ObservableList<Message> messageData = FXCollections.*observableArrayList*();  
 private MyAPI myAPI;  
 @FXML  
 private TableView<Room> roomTable;  
 @FXML  
 private TableColumn<Room, String> roomColumn;  
 @FXML  
 private TableColumn<Room, String> newMessageFlagColumn;  
  
 @FXML  
 private TableView<Message> messageTable;  
 @FXML  
 private TableColumn<Message, String> messageTimeColumn;  
 @FXML  
 private TableColumn<Message, String> messageUserColumn;  
 @FXML  
 private TableColumn<Message, String> messageTextColumn;  
  
  
 @FXML  
 private Menu aboutMenuItem;  
 @FXML  
 private Menu exitMenuItem;  
 @FXML  
 private MenuBar mainMenuBar;  
  
 @FXML  
 private JFXButton chatUsersButton;  
 @FXML  
 private JFXButton sendMessageButton;  
 @FXML  
 private TextField newMessageText;  
  
 */\*\*\*  
 \* Обработчик нажатия на кнопку сообщения  
 \*/* @FXML  
 private void sendMessageButtonClicked() {  
 String messageText = newMessageText.getText();  
 if (messageText.equals("")) {  
 *logger*.info("Поле отправки сообщения пустое");  
 } else {  
 *//Отправляем сообщеньку* try {  
 myAPI.writeMessage(messageText);  
 } catch (FalseServerFlagException | EmptyAPIResponseException e) {  
 e.printStackTrace();  
 }  
  
 newMessageText.setText("");  
 messageTable.scrollTo(messageData.get(messageData.size() - 1));  
 }  
 }  
  
 */\*\*  
 \* Обработчик нажатия на кнопку создания новой комнаты  
 \*/* @FXML  
 private void createRoomButtonClicked() {  
 app.NewRoom();  
 }  
  
  
 */\*\*  
 \* Метод инициализации (вызывается с Main)  
 \*  
 \* @param app  
 \*/* @Override  
 public void initialize(App app) {  
 this.app = app;  
 roomTable.setItems(roomData);  
  
 sendMessageButton.setOpacity(0);  
 newMessageText.setDisable(true);  
 chatUsersButton.setDisable(true);  
 newMessageText.setOpacity(0);  
 chatUsersButton.setOpacity(0);  
  
 *//API, через которое взаимодействуем с миром* myAPI = app.getMyAPI();  
  
 *//Обработчик нажатия на комнату* roomTable.getSelectionModel().selectedItemProperty().addListener(  
 ((observableValue, oldValue, newValue) -> showChatRoomDetails(newValue))  
 );  
 roomTable.setColumnResizePolicy(TableView.*CONSTRAINED\_RESIZE\_POLICY*);  
  
 *//Отображение имени комнаты в таблице* roomColumn.setCellValueFactory(cellData -> cellData.getValue().getNameProperty());  
 newMessageFlagColumn.setCellValueFactory(cellData -> cellData.getValue().newMessagesFlagString());  
  
 *//Отображение сообщений в таблице* messageTimeColumn.setCellValueFactory(cellData -> cellData.getValue().getTimeCreatedProperty());  
 messageUserColumn.setCellValueFactory(cellData -> cellData.getValue().getUserNameProperty());  
 messageTextColumn.setCellValueFactory(cellData -> cellData.getValue().getTextProperty());  
  
 *//Текст для таблицы сообщений, когда комната не выбрана* Label placeholder = new Label();  
 placeholder.setText("Выберите диалог слева или создайте новый");  
 placeholder.setStyle("-fx-text-fill: white");  
 messageTable.setPlaceholder(placeholder);  
 messageTable.setColumnResizePolicy(TableView.*CONSTRAINED\_RESIZE\_POLICY*);  
  
 *//Слушатель изменения текста* newMessageText.textProperty().addListener((observable, oldValue, newValue) -> {  
 boolean flag = newValue.equals("");  
 sendMessageButton.setDisable(flag);  
 });  
  
 *//Получаем комнаты и запускаем цикл по каждой из них* List<Room> roomList = null;  
 try {  
 roomList = myAPI.getUserRooms();  
 } catch (FalseServerFlagException | EmptyAPIResponseException e) {  
 e.printStackTrace();  
 }  
 *//Ну мало ли* assert roomList != null;  
 for (Room currentRoom : roomList) {  
  
 String currentRoomId = currentRoom.getId();  
 *//Получаем сообщения для каждой из комнат* try {  
 for (Message message : myAPI.getRoomMessagesHistory(currentRoomId)) {  
 *//Добавляем сообщеньку для комнаты* currentRoom.addMessage(message);  
 }  
 } catch (FalseServerFlagException | EmptyAPIResponseException e) {  
 e.printStackTrace();  
 }  
 }  
  
 *//Не очень эффективно, но работает* Comparator<Room> byMessagesUpdate = Comparator.*comparing*((Room o) -> o.getMessages().get(o.getMessages().size() - 1).getTimeCreatedLong());  
 roomList.sort(byMessagesUpdate.reversed());  
 roomData.addAll(roomList);  
 *logger*.info("MainChatController - инициализировали все комнаты");  
  
 *//Запускаем отдельный поток, который будет:  
 // Добавлять новую комнату в RoomData, если пришло обновление по комнате, id которой нет в RoomData  
 // Добавлять в определенный элемент room.addMessage() новое сообщение, которое прилетело через лонгпул* LongPollRunnable runnable1 = new LongPollRunnable(roomData, messageData, myAPI, app, this);  
 Thread thread1 = new Thread(runnable1, "LongPoll thread");  
 RunnableManager.*threadsList*.add(thread1);  
 thread1.start();  
 *logger*.info("MainChatController - стартанули LongPollRunnable с id " + thread1.getId());  
  
 *//Поток, который проверяет доступонсть интернета* CheckInternetRunnable runnable2 = new CheckInternetRunnable(app);  
 Thread thread2 = new Thread(runnable2, "Check internet connection thread");  
 RunnableManager.*threadsList*.add(thread2);  
 thread2.start();  
 *logger*.info("MainChatController - стартанули CheckInternetRunnable с id " + thread2.getId());  
  
 *//Биндим действия к AboutMenuItem и ExitMenuItem* MyMenuActionUtil.*onAction*(aboutMenuItem);  
 MyMenuActionUtil.*onAction*(exitMenuItem);  
 }  
  
 */\*\*  
 \* Нажатие на пункт "О программе"  
 \*/* @FXML  
 private void aboutMenuItemClicked() {  
 *logger*.info("Нажатие на button 'О программе'");  
 app.AboutMe();  
 }  
  
 */\*\*  
 \* Нажатие на выход из аккаунта пользователя  
 \*/* @FXML  
 private void exitMenuItemClicked() {  
 *//Останавливаем все потоки (лонгпул и проверка интернета)* RunnableManager.*interruptAll*();  
 *//Удаляем ключ авторизации* app.getAuthUtil().writeKey("");  
 mainMenuBar.setDisable(true);  
 *logger*.info("Осуществлен выход из профиля");  
 app.myStart(app.getPrimaryStage());  
 }  
  
 */\*\*  
 \* Обработчик нажатия на комнату-чат  
 \*  
 \* @param room - объект комнаты  
 \*/* private void showChatRoomDetails(Room room) {  
  
 *//Отображаем отправку сообщений* newMessageText.setDisable(false);  
 chatUsersButton.setDisable(false);  
 chatUsersButton.setOpacity(1);  
 sendMessageButton.setOpacity(1);  
 newMessageText.setOpacity(1);  
  
 *//Выставляем id текущей комнаты* messageData.clear();  
 if (room != null) {  
 myAPI.setCurrentRoomId(room.getId());  
 messageData.addAll(room.getMessages());  
 room.setNewMessageFlag(false);  
 messageTable.setItems(messageData);  
 }  
  
 *//Отчистка поля отправки сообщения* newMessageText.clear();  
 *//Пролистываем до конца* if (messageData.size() > 0)  
 messageTable.scrollTo(messageData.get(messageData.size() - 1));  
 }  
  
 */\*\*  
 \* Показ пользователей комнаты  
 \*/* @FXML  
 private void chatUsersButtonClicked() {  
 app.showCurrentRoomUsers();  
 *logger*.info("Нажатие на button показа пользователей комнаты");  
 }  
  
 */\*\*  
 \* Метод для LongPollRunnable, чтоб при получении обновления комнат и их последующем  
 \* поднимании вверх SelectionModel() сохранялся;  
 \*/* public void selectFirstSelectionModel() {  
 roomTable.getSelectionModel().select(0);  
 }  
  
}

src/main/java/org/demka/controllers/RegistrationController.java

package org.demka.controllers;  
  
import com.jfoenix.controls.JFXPasswordField;  
import com.jfoenix.controls.JFXTextField;  
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
import org.demka.App;  
import org.demka.api.SuperAPI;  
import org.demka.utils.ValidatorsUtil;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.util.Map;  
  
*/\*\*  
 \* Контроллер-обработчик регистрации пользователя  
 \*/*public class RegistrationController extends SuperFullController {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(RegistrationController.class);  
  
 @FXML  
 private JFXTextField userNameTextField;  
 @FXML  
 private JFXTextField emailTextField;  
 @FXML  
 private JFXPasswordField passwordTextField;  
 @FXML  
 private JFXPasswordField masterKey;  
 @FXML  
 private Label errorDescription;  
 private String errorString;  
  
 */\*\*  
 \* Валидация данных регистрации пользователя  
 \*  
 \* @param userName - ник пользователя  
 \* @param email - e-mail пользователя  
 \* @param password - пароль пользователя  
 \* @param masterKey - мастер-ключ пользователя  
 \* @return  
 \*/* private Boolean dataValidator(String userName, String email, String password, String masterKey) {  
  
 *//Проверка на пустоту* if ((userName == null) || (userName.equals(""))) {  
 errorString = "Ник не может быть пустым";  
 return false;  
 }  
  
 if ((email == null) || (email.equals(""))) {  
 errorString = "E-mail не может быть пустым";  
 return false;  
 }  
  
 if ((password == null) || (password.equals(""))) {  
 errorString = "Пароль не может быть пустым";  
 return false;  
 }  
  
 if ((masterKey == null) || (masterKey.equals(""))) {  
 errorString = "Мастер-пароль не может быть пустым";  
 return false;  
 }  
  
 if (userName.length() > 14) {  
 errorString = "Ник слишком длинный";  
 return false;  
 }  
  
 if (!ValidatorsUtil.*emailValidator*(email)) {  
 errorString = "Некорректный e-mail";  
 return false;  
 }  
  
 if (password.length() < 8) {  
 errorString = "Пароль должен быть не менее 8 символов";  
 return false;  
 }  
  
 if (masterKey.length() < 8) {  
 errorString = "Мастер-пароль должен быть не менее 8 символов";  
 return false;  
 }  
  
 if (masterKey.equals(password)) {  
 errorString = "Пароль и мастер-пароль не должны совпадать";  
 return false;  
 }  
  
 return true;  
 }  
  
 */\*\*  
 \* Обработчик нажатия на кнопку регистрации  
 \*/* @FXML  
 private void regButtonClicked() {  
 *logger*.info("Нажатие на button регистрации");  
 String userName = userNameTextField.getText();  
 String userEmail = emailTextField.getText();  
 String userPassword = passwordTextField.getText();  
 String masterKey = this.masterKey.getText();  
 if (dataValidator(userName, userEmail, userPassword, masterKey)) {  
 *logger*.info("Валидация данных со стороны клиента прошла успешно");  
 *//Получаем ответ от сервера* Map<String, Object> regResult = SuperAPI.*registration*(userName, userEmail, userPassword, masterKey);  
  
 if ((boolean) regResult.get("result")) {  
 errorDescription.setOpacity(0.0);  
 errorDescription.setText("");  
 *logger*.info("Успешно зарегистрировали пользователя");  
 app.SuccessUserAction("Успешная регистрация", "Регистрация прошла успешно!");  
 } else {  
 errorString = (String) regResult.get("error");  
 errorDescription.setText(errorString);  
 *logger*.info("Валидация со стороны сервера не прошла: " + errorString);  
 errorDescription.setOpacity(1.0);  
 }  
  
 } else {  
 *logger*.info("Валидация данных со стороны клиента не прошла: " + errorString);  
 errorDescription.setText(errorString);  
 errorDescription.setOpacity(1.0);  
 }  
  
  
 }  
  
  
 */\*\*  
 \* Обработчик нажатия на кнопку возврата на форму авторизации  
 \*/* @FXML  
 private void backButtonClicked() {  
 *logger*.info("Возврат в меню авторизации");  
 app.UserAuthorisation();  
 }  
  
 */\*\*  
 \* Метод инициализации (вызывается с Main)  
 \*  
 \* @param app  
 \*/* @Override  
 public void initialize(App app) {  
 this.app = app;  
 errorDescription.setOpacity(0.0);  
 }  
}

src/main/java/org/demka/controllers/RootLayoutController.java

package org.demka.controllers;  
  
import org.demka.App;  
  
*/\*\*  
 \* Базовый контроллер  
 \*/*public class RootLayoutController extends SuperFullController {  
  
 */\*\*  
 \* Метод инициализации (вызывается с Main)  
 \*  
 \* @param app  
 \*/* @Override  
 public void initialize(App app) {  
 this.app = app;  
 }  
}

src/main/java/org/demka/controllers/ShowChatUsersController.java

package org.demka.controllers;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.fxml.FXML;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.stage.Stage;  
import org.demka.App;  
import org.demka.api.MyAPI;  
import org.demka.exceptions.EmptyAPIResponseException;  
import org.demka.exceptions.FalseServerFlagException;  
import org.demka.exceptions.RoomNotFoundException;  
import org.demka.models.Room;  
import org.demka.models.User;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.util.List;  
  
*/\*\*  
 \* Контроллер-обработчик окна с информацией о выбранном диалоге;  
 \*/*public class ShowChatUsersController extends SuperPartController {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(ShowChatUsersController.class);  
 private final ObservableList<User> chatUserData = FXCollections.*observableArrayList*();  
 @FXML  
 private TableView<User> chatUserTable;  
 @FXML  
 private TableColumn<User, String> chatUserNameColumn;  
  
 */\*\*  
 \* Метод инициализации (вызывается с Main)  
 \*  
 \* @param app  
 \* @param dialogStage  
 \*/* @Override  
 public void initialize(App app, Stage dialogStage) {  
 this.app = app;  
 this.dialogStage = dialogStage;  
 MyAPI myAPI = app.getMyAPI();  
  
 chatUserTable.setItems(chatUserData);  
 chatUserNameColumn.setCellValueFactory(cellData -> cellData.getValue().getNameProperty());  
 String currentRoomId = myAPI.getCurrentRoomId();  
 try {  
 Room currentRoom = myAPI.getRoomInfo(currentRoomId);  
 dialogStage.setTitle("Информация о диалоге " + currentRoom.getName());  
 List<User> roomUsersList = myAPI.getUsersByRoom(currentRoomId);  
 chatUserData.addAll(roomUsersList);  
 chatUserNameColumn.setText("Участники (" + roomUsersList.size() + " всего)");  
 } catch (FalseServerFlagException | EmptyAPIResponseException | RoomNotFoundException e) {  
 e.printStackTrace();  
 dialogStage.close();  
 }  
 }  
  
 @FXML  
 private void backButtonClicked() {  
 *logger*.info("Выход из из подменю информации о пользователях чата");  
 dialogStage.close();  
 }  
  
}

src/main/java/org/demka/controllers/SuccessActionController.java

package org.demka.controllers;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.Label;  
import org.demka.App;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
*/\*\*  
 \* Контроллер-обработчик успешной операции пользователя  
 \*/*public class SuccessActionController extends SuperFullController {  
 private static final Logger *logger* = LoggerFactory.*getLogger*(RegistrationController.class);  
  
  
 @FXML  
 private Label mainText;  
  
 */\*\*  
 \* Метод инициализации (вызывается с Main)  
 \*  
 \* @param app  
 \*/* @Override  
 public void initialize(App app) {  
 this.app = app;  
 }  
  
 */\*\*  
 \* Нажатие на кнопку формы авторизации  
 \*/* @FXML  
 private void okButtonClicked() {  
 *logger*.info("Нажатие на button возврата на формы авторизации");  
 app.UserAuthorisation();  
 }  
  
 */\*\*  
 \* Sets main text.  
 \*  
 \* @param text the text  
 \*/* public void setMainText(String text) {  
 mainText.setText(text);  
 }  
}

src/main/java/org/demka/controllers/SuperFullController.java

package org.demka.controllers;  
  
import org.demka.App;  
  
*/\*\*  
 \* Класс, от которого наследуются все контроллеры, layout которых открывается на всё окно  
 \*/*public abstract class SuperFullController {  
 */\*\*  
 \* The App.  
 \*/* protected App app;  
  
 */\*\*  
 \* Метод инициализации (вызывается с Main)  
 \*  
 \* @param app the app  
 \*/* public abstract void initialize(App app);  
}

src/main/java/org/demka/controllers/SuperPartController.java

package org.demka.controllers;  
  
import javafx.stage.Stage;  
import org.demka.App;  
  
*/\*\*  
 \* Класс, от которого наследуются все контроллеры, layout которых открывается поверх основного окна  
 \*/*public abstract class SuperPartController {  
 */\*\*  
 \* The App.  
 \*/* protected App app;  
 */\*\*  
 \* The Dialog stage.  
 \*/* protected Stage dialogStage;  
  
 */\*\*  
 \* Метод инициализации (вызывается с Main)  
 \*  
 \* @param app the app  
 \* @param dialogStage the dialog stage  
 \*/* public abstract void initialize(App app, Stage dialogStage);  
}

src/main/java/org/demka/exceptions/EmptyAPIResponseException.java

package org.demka.exceptions;  
  
  
import javafx.application.Platform;  
import org.demka.App;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
*/\*\*  
 \* Ошибка, возникающая при пустом ответе со стороны сервера (или, например, если нет интернета)  
 \*/*public class EmptyAPIResponseException extends Exception {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(EmptyAPIResponseException.class);  
  
 */\*\*  
 \* Конструктор ошибки пустого ответа от сервера  
 \*  
 \* @param mainApp - объект экземпляра app  
 \* @param errorMessage - описание ошибки  
 \*/* public EmptyAPIResponseException(App mainApp, String errorMessage) {  
 super(errorMessage);  
 *logger*.error(errorMessage);  
  
 *//Это синхронизация с main JavaFX потоком, чтоб там отрисовать эту сцену* Platform.*runLater*(mainApp::ConnectionError);  
 }  
  
  
}

src/main/java/org/demka/exceptions/FalseServerFlagException.java

package org.demka.exceptions;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
*/\*\*  
 \* Обработка отрицательного ответа от сервера (response.result == false)  
 \*/*public class FalseServerFlagException extends Exception {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(FalseServerFlagException.class);  
  
 */\*\*  
 \* Конструктор ошибки некорректного ответа от сервера  
 \*  
 \* @param url - url, по которому обращались  
 \* @param serverResponse - ответ сервера  
 \* @param errorMessage - описание ошибки  
 \*/* public FalseServerFlagException(String url, String serverResponse, String errorMessage) {  
 super(errorMessage);  
 *logger*.error(errorMessage + "\n" + "URL: " + url + "\n" + "Ответ: " + serverResponse);  
 }  
}

src/main/java/org/demka/exceptions/LongPollListenerException.java

package org.demka.exceptions;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
*/\*\*  
 \* Ошибка прослушки longPoll  
 \*/*public class LongPollListenerException extends Exception {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(LongPollListenerException.class);  
  
 */\*\*  
 \* Ошибка прослушки лонгпула  
 \*  
 \* @param errorMessage - описание ошибки  
 \*/* public LongPollListenerException(String errorMessage) {  
 super(errorMessage);  
 *logger*.info(errorMessage);  
 }  
}

src/main/java/org/demka/exceptions/RoomNotFoundException.java

package org.demka.exceptions;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
*/\*\*  
 \* Ошибка, если комната не найдена  
 \*/*public class RoomNotFoundException extends Exception {  
 private static final Logger *logger* = LoggerFactory.*getLogger*(RoomNotFoundException.class);  
  
 */\*\*  
 \* Ошибка, если комната не найдена  
 \*  
 \* @param errorMessage - описание ошибки  
 \*/* public RoomNotFoundException(String errorMessage) {  
 super(errorMessage);  
 *logger*.error(errorMessage);  
 }  
}

src/main/java/org/demka/models/Message.java

package org.demka.models;  
  
import javafx.beans.property.LongProperty;  
import javafx.beans.property.SimpleLongProperty;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.beans.property.StringProperty;  
import org.demka.utils.UNIXTimeUtil;  
  
*/\*\*  
 \* Модель сообщения  
 \*/*public class Message {  
  
 private final StringProperty userId;  
 private final StringProperty userName;  
 private final StringProperty text;  
 private final StringProperty roomId;  
 private final StringProperty timeCreatedString;  
 private final LongProperty timeCreatedLong;  
 private final StringProperty id;  
  
 */\*\*  
 \* Конструктор сообщения  
 \*  
 \* @param userId - идентификатор пользователя, отправившего сообщение  
 \* @param userName - имя пользователя, отправившего сообщение  
 \* @param text - текст сообщения  
 \* @param roomId - идентификатор комнаты, куда отправляется сообщение  
 \* @param timeCreated - UNIX-время создания сообщения  
 \* @param id - идентификатор сообщения  
 \*/* public Message(String userId, String userName, String text, String roomId, long timeCreated, String id) {  
 this.userId = new SimpleStringProperty(userId);  
 this.userName = new SimpleStringProperty(userName);  
 this.text = new SimpleStringProperty(text);  
 this.roomId = new SimpleStringProperty(roomId);  
  
 this.timeCreatedLong = new SimpleLongProperty(timeCreated);  
 this.timeCreatedString = new SimpleStringProperty(UNIXTimeUtil.*convert*(timeCreated));  
 this.id = new SimpleStringProperty(id);  
 }  
  
 */\*\*  
 \* Gets text.  
 \*  
 \* @return the text  
 \*/* public String getText() {  
 return text.get();  
 }  
  
 */\*\*  
 \* Gets user name.  
 \*  
 \* @return the user name  
 \*/* public String getUserName() {  
 return userName.get();  
 }  
  
 */\*\*  
 \* Gets user name property.  
 \*  
 \* @return the user name property  
 \*/* public StringProperty getUserNameProperty() {  
 return userName;  
 }  
  
 */\*\*  
 \* Gets text property.  
 \*  
 \* @return the text property  
 \*/* public StringProperty getTextProperty() {  
 return text;  
 }  
  
 */\*\*  
 \* Gets time created property.  
 \*  
 \* @return the time created property  
 \*/* public StringProperty getTimeCreatedProperty() {  
 return timeCreatedString;  
 }  
  
 */\*\*  
 \* Gets time created string.  
 \*  
 \* @return the time created string  
 \*/* public String getTimeCreatedString() {  
 return timeCreatedString.get();  
 }  
  
 */\*\*  
 \* Gets time created long.  
 \*  
 \* @return the time created long  
 \*/* public long getTimeCreatedLong() {  
 return timeCreatedLong.get();  
 }  
  
 */\*\*  
 \* Gets room id.  
 \*  
 \* @return the room id  
 \*/* public String getRoomId() {  
 return roomId.get();  
 }  
  
 */\*\*  
 \* Gets id.  
 \*  
 \* @return the id  
 \*/* public String getId() {  
 return id.get();  
 }  
  
}

src/main/java/org/demka/models/Room.java

package org.demka.models;  
  
import javafx.beans.property.\*;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Модель комнаты  
 \*/*public class Room {  
  
 private final static String *NO\_NEW\_MESSAGE\_FLAG* = " ";  
 private final static String *NEW\_MESSAGE\_FLAG* = "●";  
 private final StringProperty creatorId;  
 private final StringProperty name;  
 private final IntegerProperty timeCreated;  
 private final ListProperty<String> users;  
 private final StringProperty id;  
 private final StringProperty newMessagesFlag;  
 private final List<Message> messages;  
  
 */\*\*  
 \* Конструктор комнаты  
 \*  
 \* @param creatorId - идентификатор пользователя-создателя комнаты  
 \* @param name - имя комнаты  
 \* @param timeCreated - UNIX-время создания комнаты  
 \* @param users - список идентификаторов создателей  
 \* @param id - идентификатор комнаты  
 \*/* public Room(String creatorId, String name, int timeCreated, List<String> users, String id) {  
 this.creatorId = new SimpleStringProperty(creatorId);  
 this.name = new SimpleStringProperty(name);  
 this.timeCreated = new SimpleIntegerProperty(timeCreated);  
 ObservableList<String> userData = FXCollections.*observableArrayList*();  
 userData.addAll(users);  
 this.users = new SimpleListProperty<>(userData);  
 this.id = new SimpleStringProperty(id);  
 this.messages = new ArrayList<>();  
 this.newMessagesFlag = new SimpleStringProperty(*NO\_NEW\_MESSAGE\_FLAG*);  
 }  
  
 */\*\*  
 \* Выставление флага прочитанных сообщений для комнаты  
 \*  
 \* @param flag булевое значение  
 \*/* public void setNewMessageFlag(boolean flag) {  
 if (flag)  
 newMessagesFlag.setValue(*NEW\_MESSAGE\_FLAG*);  
 else  
 newMessagesFlag.setValue(*NO\_NEW\_MESSAGE\_FLAG*);  
 }  
  
 */\*\*  
 \* New messages flag string string property.  
 \*  
 \* @return the string property  
 \*/* public StringProperty newMessagesFlagString() {  
 return newMessagesFlag;  
 }  
  
 */\*\*  
 \* Gets name property.  
 \*  
 \* @return the name property  
 \*/* public StringProperty getNameProperty() {  
 return name;  
 }  
  
 */\*\*  
 \* Gets name.  
 \*  
 \* @return the name  
 \*/* public String getName() {  
 return name.get();  
 }  
  
 */\*\*  
 \* Gets id.  
 \*  
 \* @return the id  
 \*/* public String getId() {  
 return id.get();  
 }  
  
 */\*\*  
 \* Add message.  
 \*  
 \* @param message the message  
 \*/* public void addMessage(Message message) {  
 this.messages.add(message);  
 }  
  
 */\*\*  
 \* Gets messages.  
 \*  
 \* @return the messages  
 \*/* public List<Message> getMessages() {  
 return messages;  
 }  
  
 */\*\*  
 \* Gets users.  
 \*  
 \* @return the users  
 \*/* public ObservableList<String> getUsers() {  
 return users.get();  
 }  
  
}

src/main/java/org/demka/models/User.java

package org.demka.models;  
  
import javafx.beans.property.IntegerProperty;  
import javafx.beans.property.SimpleIntegerProperty;  
import javafx.beans.property.SimpleStringProperty;  
import javafx.beans.property.StringProperty;  
  
*/\*\*  
 \* Модель пользователя  
 \*/*public class User {  
 private final StringProperty id;  
 private final StringProperty name;  
 private final IntegerProperty timeCreated;  
  
 */\*\*  
 \* Конструктор пользователя  
 \*  
 \* @param id - идентификатор пользователя  
 \* @param name - имя пользователя  
 \* @param timeCreated - время создания пользователя  
 \*/* public User(String id, String name, Integer timeCreated) {  
 this.id = new SimpleStringProperty(id);  
 this.name = new SimpleStringProperty(name);  
 this.timeCreated = new SimpleIntegerProperty(timeCreated);  
 }  
  
 */\*\*  
 \* Gets name property.  
 \*  
 \* @return the name property  
 \*/* public StringProperty getNameProperty() {  
 return name;  
 }  
  
 */\*\*  
 \* Gets id property.  
 \*  
 \* @return the id property  
 \*/* public StringProperty getIdProperty() {  
 return id;  
 }  
  
 */\*\*  
 \* Gets id.  
 \*  
 \* @return the id  
 \*/* public String getId() {  
 return id.get();  
 }  
  
 */\*\*  
 \* Gets name.  
 \*  
 \* @return the name  
 \*/* public String getName() {  
 return name.get();  
 }  
}

src/main/java/org/demka/runnable/CheckInternetRunnable.java

package org.demka.runnable;  
  
import javafx.application.Platform;  
import org.demka.App;  
import org.demka.controllers.ConnectionErrorController;  
import org.demka.exceptions.EmptyAPIResponseException;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.io.IOException;  
import java.net.InetSocketAddress;  
import java.net.Socket;  
  
*/\*\*  
 \* Отдельный поток для проверки интернет-соединения  
 \*/*public class CheckInternetRunnable implements Runnable {  
  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(CheckInternetRunnable.class);  
 private final App app;  
  
 */\*\*  
 \* Конструктор для потока проверки интернет-соединения  
 \*  
 \* @param app - экземпляр app  
 \*/* public CheckInternetRunnable(App app) {  
 this.app = app;  
 }  
  
 */\*\*  
 \* Проверка интернет-соединения  
 \*  
 \* @param addr - хост (ip/домен)  
 \* @param openPort - порт  
 \* @param timeOutMillis - таймаут ожидания ответа  
 \* @return  
 \*/* private static boolean isReachable(String addr, int openPort, int timeOutMillis) {  
  
 try {  
 try (Socket soc = new Socket()) {  
 soc.connect(new InetSocketAddress(addr, openPort), timeOutMillis);  
 }  
 return true;  
 } catch (IOException ex) {  
 return false;  
 }  
 }  
  
 */\*\*  
 \* When an object implementing interface {@code Runnable} is used  
 \* to create a thread, starting the thread causes the object's  
 \* {@code run} method to be called in that separately executing  
 \* thread.  
 \* <p>  
 \* The general contract of the method {@code run} is that it may  
 \* take any action whatsoever.  
 \*  
 \* @see Thread#run()  
 \*/* @Override  
 public void run() {  
  
 while (!Thread.*currentThread*().isInterrupted()) {  
  
 *//Проверяем интернет* boolean exceptionFlag = false;  
 if (!*isReachable*("149.248.54.195", 8080, 10000)) {  
 try {  
 exceptionFlag = true;  
 *//Вызываем страницу с ошибкой интернет-соединения, если еще не вызвали* if (!ConnectionErrorController.*isActive*) {  
 throw new EmptyAPIResponseException(app, "нет интернет-соединения");  
 }  
 } catch (EmptyAPIResponseException emptyAPIResponseException) {  
 emptyAPIResponseException.printStackTrace();  
 }  
 }  
  
 *//Спим* try {  
 Thread.*sleep*(4000);  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 *logger*.error("Поток '" + Thread.*currentThread*().getName() + "' с id " + Thread.*currentThread*().getId() + " убит");  
 return;  
 }  
  
 *//Если интернет появился, то обратно переходим в чат* if ((!exceptionFlag) && (ConnectionErrorController.*isActive*)) {  
 ConnectionErrorController.*isActive* = false;  
 Platform.*runLater*(() -> app.myStart(app.getPrimaryStage()));  
 RunnableManager.*interruptAll*();  
 break;  
 }  
 }  
  
 *//Переходим отбратно в чат, а данный поток завершает свою работу  
 logger*.error("Поток '" + Thread.*currentThread*().getName() + "' с id " + Thread.*currentThread*().getId() + " завершил работу");  
 }  
}

src/main/java/org/demka/runnable/LongPollRunnable.java

package org.demka.runnable;  
  
import javafx.application.Platform;  
import javafx.collections.ObservableList;  
import org.demka.App;  
import org.demka.api.MyAPI;  
import org.demka.controllers.ConnectionErrorController;  
import org.demka.controllers.MainChatController;  
import org.demka.exceptions.EmptyAPIResponseException;  
import org.demka.exceptions.FalseServerFlagException;  
import org.demka.exceptions.LongPollListenerException;  
import org.demka.exceptions.RoomNotFoundException;  
import org.demka.models.Message;  
import org.demka.models.Room;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Отдельный поток для получения сообщений через логику лонгпулинга  
 \*/*public class LongPollRunnable implements Runnable {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(LongPollRunnable.class);  
 private final ObservableList<Room> roomData;  
 private final ObservableList<Message> messageData;  
 private final MyAPI myAPI;  
 private final App app;  
 private final MainChatController mainChatController;  
  
 */\*\*  
 \* Конструктор LongPollRunnable  
 \*  
 \* @param roomData - ObservableList данных комнат, который синхронизирован с таблицей roomTable в JavaFX  
 \* @param messageData - ObservableList данных сообщений, который синхронизирован с таблицей messageTable в JavaFX  
 \* @param myAPI - объект текущей сессии API для сапросов с бека  
 \* @param app - объект app  
 \* @param mainChatController the main chat controller  
 \*/* public LongPollRunnable(ObservableList<Room> roomData, ObservableList<Message> messageData, MyAPI myAPI, App app, MainChatController mainChatController) {  
 this.roomData = roomData;  
 this.myAPI = myAPI;  
 this.messageData = messageData;  
 this.app = app;  
 this.mainChatController = mainChatController;  
 }  
  
 */\*\*  
 \* Перемещение комнаты на первую позицию списка  
 \*  
 \* @param roomData - список ObservableList  
 \* @param roomIndex - индекс комнаты, которую необходимо поднять  
 \*/* public void updateRoomData(ObservableList<Room> roomData, int roomIndex) {  
 if (roomIndex > 0) {  
 Room buffRoom = roomData.get(roomIndex);  
 ArrayList<Room> buffer = new ArrayList<>(roomData);  
 buffer.remove(buffRoom);  
 buffer.add(0, buffRoom);  
 roomData.clear();  
 roomData.addAll(buffer);  
 }  
 }  
  
 */\*\*  
 \* Проверка на существование комнаты по её id  
 \*  
 \* @param roomId - идентификатор комнаты  
 \* @return integer  
 \*/* public Integer isRoomExist(String roomId) {  
 for (int i = 0; i < roomData.size(); i++) {  
 if (roomData.get(i).getId().equals(roomId)) {  
 return i;  
 }  
 }  
 return -1;  
 }  
  
 */\*\*  
 \* When an object implementing interface {@code Runnable} is used  
 \* to create a thread, starting the thread causes the object's  
 \* {@code run} method to be called in that separately executing  
 \* thread.  
 \* <p>  
 \* The general contract of the method {@code run} is that it may  
 \* take any action whatsoever.  
 \*  
 \* @see Thread#run()  
 \*/* @Override  
 public void run() {  
  
 while (!Thread.*currentThread*().isInterrupted()) {  
 *logger*.info("LongPollRunnable - Работаем");  
 *//Пытаемся получить новые данные* boolean exceptionFlag = false;  
 try {  
 List<Message> newMessages = myAPI.longPollListener();  
  
 for (Message msg : newMessages) {  
 *//Проверка на существование комнаты* String messageRoomId = msg.getRoomId();  
 Integer existInt = isRoomExist(messageRoomId);  
  
 *//Существует* if (existInt != -1) {  
 *//Добавляем сообщение* Room currentRoom = roomData.get(existInt);  
 currentRoom.addMessage(msg);  
 *logger*.info("Добавили сообщение '" + msg.getText() + "' для комнаты " + msg.getRoomId());  
 updateRoomData(roomData, existInt);  
  
 *//Если открыт уже диалог с текущей конференцией* if (messageRoomId.equals(myAPI.getCurrentRoomId())) {  
 messageData.add(msg);  
 Platform.*runLater*(mainChatController::selectFirstSelectionModel);  
 }  
 *//Иначе выставляем флаг нового сообщения для комнаты* else {  
 currentRoom.setNewMessageFlag(true);  
 }  
 }  
  
 *//Значит это сообщение с новой комнаты* else {  
 try {  
 *//Создаем объект комнаты* Room newRoom = myAPI.getRoomInfo(messageRoomId);  
 newRoom.setNewMessageFlag(true);  
 *//Добавляем полученное сообщение* newRoom.addMessage(msg);  
 *//Добавляем саму комнату* roomData.add(0, newRoom);  
 *logger*.info("Получили новую комнату " + newRoom.getName() + " [" + newRoom.getId() + "]");  
 } catch (RoomNotFoundException e) {  
 *logger*.error("Случилось странное: бек отдал сообщение с комнаты " + messageRoomId + ", но её не существует");  
 } catch (EmptyAPIResponseException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 }  
  
 *//Если необходимо заново пройти авторизацию - проходим* catch (LongPollListenerException e) {  
 e.printStackTrace();  
 try {  
 myAPI.getLongPollServer();  
 } catch (EmptyAPIResponseException | FalseServerFlagException newE) {  
 exceptionFlag = true;  
 newE.printStackTrace();  
 }  
 }  
  
 try {  
 Thread.*sleep*(250);  
 } catch (InterruptedException e) {  
 Thread.*currentThread*().interrupt();  
 *logger*.error("Поток '" + Thread.*currentThread*().getName() + "' с id " + Thread.*currentThread*().getId() + " убит");  
 return;  
 }  
  
 *//Если находились в ConnectionErrorController, но при этом не получили ошибки, то обратно переходим в чат* if ((!exceptionFlag) && (ConnectionErrorController.*isActive*)) {  
 ConnectionErrorController.*isActive* = false;  
 Platform.*runLater*(() -> app.myStart(app.getPrimaryStage()));  
 RunnableManager.*interruptAll*();  
 break;  
 }  
 }  
  
 *//Переходим отбратно в чат, а данный поток завершает свою работу  
 logger*.error("Поток '" + Thread.*currentThread*().getName() + "' с id " + Thread.*currentThread*().getId() + " завершил работу");  
 }  
}

src/main/java/org/demka/runnable/RunnableManager.java

package org.demka.runnable;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Класс для менеджмента потоков LongPollRunnable и CheckInternetRunnable  
 \*/*public class RunnableManager {  
 private static final Logger *logger* = LoggerFactory.*getLogger*(RunnableManager.class);  
 */\*\*  
 \* The constant threadsList.  
 \*/* public static List<Thread> *threadsList* = new ArrayList<>();  
  
 */\*\*  
 \* Прерывание всех потоков, которые есть в списке threadsList  
 \*/* public static void interruptAll() {  
  
 for (Thread t : *threadsList*) {  
 t.interrupt();  
 *logger*.info("Отправили прерывание для потока с id " + t.getId() + " " + t.getName());  
 }  
 }  
}

src/main/java/org/demka/utils/AuthUtil.java

package org.demka.utils;  
  
import java.io.File;  
import java.io.IOException;  
  
*/\*\*  
 \* Класс для работы с автоавторизацией  
 \*/*public class AuthUtil {  
  
 private final FileProcessingUtil fileProcessingUtil;  
  
 */\*\*  
 \* Констркутор класса AuthUtil.  
 \* Пытается найти файл .tempdata, если не существует - создает его  
 \*/* public AuthUtil() {  
  
 *//Получаем путь файла, где хранится авторизация* String pathStr = null;  
 try {  
 pathStr = new File(".tempdata").getCanonicalPath();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 this.fileProcessingUtil = new FileProcessingUtil(pathStr);  
 }  
  
 */\*\*  
 \* Чтение данных из файла  
 \*  
 \* @return string  
 \*/* public String readKey() {  
 return fileProcessingUtil.DataRead();  
 }  
  
 */\*\*  
 \* Запись данных в файл  
 \*  
 \* @param data - данные для записи  
 \*/* public void writeKey(String data) {  
 fileProcessingUtil.DataWrite(data);  
 }  
  
  
}

src/main/java/org/demka/utils/FileProcessingUtil.java

package org.demka.utils;  
  
import java.io.\*;  
import java.util.stream.Collectors;  
  
*/\*\*  
 \* Класс для работы с файлами  
 \*/*public class FileProcessingUtil {  
  
 */\*\*  
 \* The File name.  
 \*/* String fileName;  
  
 */\*\*  
 \* Конструктор FileProcessingUtil  
 \*  
 \* @param fileName - имя файла для работы  
 \*/* public FileProcessingUtil(String fileName) {  
 this.fileName = fileName;  
  
 File tempFile = new File(fileName);  
 *//Если файла не существует - создаем его* if (!tempFile.exists()) {  
 try {  
 *//Создаем новый файл, если его не существует* tempFile.createNewFile();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 */\*\*  
 \* Чтение данных  
 \*  
 \* @return string  
 \*/* public String DataRead() {  
  
 try {  
 BufferedReader reader = new BufferedReader(new FileReader(this.fileName));  
 String lines = reader.lines().collect(Collectors.*joining*());  
 reader.close();  
 return lines;  
 } catch (IOException e) {  
 e.printStackTrace();  
 return null;  
 }  
 }  
  
 */\*\*  
 \* Запись данных  
 \*  
 \* @param data - данные для записи  
 \*/* public void DataWrite(String data) {  
 try {  
 BufferedWriter writer = new BufferedWriter(new FileWriter(this.fileName));  
 writer.write(data);  
 writer.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

src/main/java/org/demka/utils/MyMenuActionUtil.java

package org.demka.utils;  
  
import javafx.scene.control.Menu;  
import javafx.scene.control.MenuItem;  
  
*/\*\*  
 \* Класс для обработки нажатия на Menu без Menuitem'ов  
 \*/*public class MyMenuActionUtil {  
 */\*\*  
 \* Биндинг нажатия на Menu  
 \*  
 \* @param menu - экземпляр JavaFX controls menu  
 \*/* public static void onAction(Menu menu) {  
 final MenuItem menuItem = new MenuItem();  
 menu.getItems().add(menuItem);  
 menu.addEventHandler(Menu.*ON\_SHOWN*, event -> menu.hide());  
 menu.addEventHandler(Menu.*ON\_SHOWING*, event -> menu.fire());  
 }  
}

src/main/java/org/demka/utils/String2HashUtil.java

package org.demka.utils;  
  
import java.nio.charset.StandardCharsets;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
  
*/\*\*  
 \* Класс для хеширования данных в SHA-256  
 \*/*public class String2HashUtil {  
  
 */\*\*  
 \* Преобразование байтов в HEX-строку  
 \*  
 \* @param hash - байтовые данные  
 \* @return  
 \*/* private static String bytesToHex(byte[] hash) {  
 StringBuilder hexString = new StringBuilder(2 \* hash.length);  
 for (byte b : hash) {  
 String hex = Integer.*toHexString*(0xff & b);  
 if (hex.length() == 1) {  
 hexString.append('0');  
 }  
 hexString.append(hex);  
 }  
 return hexString.toString();  
 }  
  
 */\*\*  
 \* Конвертация хеширование данных в SHA-256  
 \*  
 \* @param input - строка для хеширования  
 \* @return string  
 \*/* public static String convert(String input) {  
  
 try {  
 MessageDigest digest = MessageDigest.*getInstance*("SHA-256");  
 return *bytesToHex*(digest.digest(input.getBytes(StandardCharsets.*UTF\_8*)));  
 } catch (NoSuchAlgorithmException e) {  
 return null;  
 }  
 }  
}

src/main/java/org/demka/utils/UNIXTimeUtil.java

package org.demka.utils;  
  
import java.time.Instant;  
import java.time.ZoneId;  
import java.time.format.DateTimeFormatter;  
  
*/\*\*  
 \* Класс для работы с UNIX-временем  
 \*/*public class UNIXTimeUtil {  
  
 private static final DateTimeFormatter *formatter* = DateTimeFormatter.*ofPattern*("HH:mm:ss").withZone(ZoneId.*systemDefault*());  
  
 */\*\*  
 \* Конвертирует UNIX-время в обычное время  
 \*  
 \* @param time - UNIX-время  
 \* @return string  
 \*/* public static String convert(long time) {  
 return *formatter*.format(Instant.*ofEpochSecond*(time));  
 }  
}

src/main/java/org/demka/utils/ValidatorsUtil.java

package org.demka.utils;  
  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
*/\*\*  
 \* The type Validators util.  
 \*/*public class ValidatorsUtil {  
  
 */\*\*  
 \* Проверка e-mail на корректность  
 \*  
 \* @param email - исходный e-mail  
 \* @return - результат валидации  
 \*/* public static boolean emailValidator(String email) {  
 String regex = "^[a-zA-Z0-9\_!#$%&'\*+/=?`{|}~^.-]+@[a-zA-Z0-9.-]+$";  
 Pattern pattern = Pattern.*compile*(regex);  
 Matcher matcher = pattern.matcher(email);  
 return matcher.matches();  
 }  
}

src/main/java/org/demka/App.java

package org.demka;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.layout.AnchorPane;  
import javafx.scene.layout.BorderPane;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
import org.demka.api.MyAPI;  
import org.demka.controllers.\*;  
import org.demka.utils.AuthUtil;  
  
import java.io.IOException;  
  
*/\*\*  
 \* JavaFX App  
 \*/*public class App extends Application {  
  
 private Stage primaryStage;  
 private BorderPane rootLayout;  
 private MyAPI myAPI;  
 private AuthUtil authUtil;  
  
 */\*\*  
 \* Точка входа в программу  
 \*  
 \* @param args the input arguments  
 \*/* public static void main(String[] args) {  
 *launch*(args);  
 }  
  
  
 @Override  
 public void start(Stage primaryStage) {  
 myStart(primaryStage);  
 }  
  
 */\*\*  
 \* Метод для отрисовки окна при инициализации/перезапуске.  
 \* Нужен отдельный метод из-за того, что при отсутствии интернет-соединения  
 \* и последующем его появлении, необходимо перезапустить приложение  
 \*  
 \* @param primaryStage the primary stage  
 \*/* public void myStart(Stage primaryStage) {  
  
 this.primaryStage = primaryStage;  
 this.primaryStage.setTitle("DEMKAChat - Авторизация");  
 this.authUtil = new AuthUtil();  
 initRootLayout();  
 *//Читаем токен из файла* String key = authUtil.readKey();  
 if (key != null) {  
 myAPI = new MyAPI(key, this);  
 }  
  
 *//Если пользователь не авторизован* if (!myAPI.getIsAuthenticated()) {  
 authUtil.writeKey("");  
 UserAuthorisation();  
 }  
 *//Если уже успешно авторизовался* else {  
 this.primaryStage.setTitle("DEMKAChat - Сообщения [" + myAPI.getUserName() + "]");  
 MainChat();  
 }  
 }  
  
 */\*\*  
 \* Логика авторизации в программе  
 \*/* public void UserAuthorisation() {  
  
 try {  
 this.primaryStage.setTitle("DEMKAChat - Авторизация");  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(App.class.getResource("views/LoginView.fxml"));  
 AnchorPane mainPage = loader.load();  
 rootLayout.setCenter(mainPage);  
 LoginController controller = loader.getController();  
 controller.initialize(this);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Показывает окно регистрации в программе  
 \*/* public void UserRegistration() {  
 try {  
 this.primaryStage.setTitle("DEMKAChat - Регистрация");  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(App.class.getResource("views/RegistrationView.fxml"));  
 AnchorPane mainPage = loader.load();  
 rootLayout.setCenter(mainPage);  
 RegistrationController controller = loader.getController();  
 controller.initialize(this);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Показывает окно смены пароля в программе  
 \*/* public void ForgotPassword() {  
 try {  
 this.primaryStage.setTitle("DEMKAChat - Забыли пароль");  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(App.class.getResource("views/ForgotPasswordView.fxml"));  
 AnchorPane mainPage = loader.load();  
 rootLayout.setCenter(mainPage);  
 ForgotPasswordController controller = loader.getController();  
 controller.initialize(this);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
  
 */\*\*  
 \* Динамическое окно успешного действия пользователя  
 \*  
 \* @param windowTitle - заголовок показываемого окна  
 \* @param mainText - текст, который отображается на форме  
 \*/* public void SuccessUserAction(String windowTitle, String mainText) {  
 try {  
 this.primaryStage.setTitle("DEMKAChat - " + windowTitle);  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(App.class.getResource("views/SuccessActionView.fxml"));  
 AnchorPane mainPage = loader.load();  
 rootLayout.setCenter(mainPage);  
 SuccessActionController controller = loader.getController();  
 controller.setMainText(mainText);  
 controller.initialize(this);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Показывает информацию о программе и авторе  
 \*/* public void AboutMe() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(App.class.getResource("views/AboutMeView.fxml"));  
 AnchorPane page = loader.load();  
  
 Stage dialogStage = new Stage();  
 dialogStage.setTitle("О программе");  
 dialogStage.initModality(Modality.*WINDOW\_MODAL*);  
 dialogStage.initOwner(primaryStage);  
 Scene scene = new Scene(page);  
 dialogStage.setScene(scene);  
 AboutMeController controller = loader.getController();  
 controller.initialize(this, dialogStage);  
 dialogStage.showAndWait();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Основное окно с чатами  
 \*/* public void MainChat() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(App.class.getResource("views/MainChatView.fxml"));  
 AnchorPane mainPage = loader.load();  
 rootLayout.setCenter(mainPage);  
 MainChatController controller = loader.getController();  
 controller.initialize(this);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Логика ошибки соединения  
 \*/* public void ConnectionError() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(App.class.getResource("views/ConnectionErrorView.fxml"));  
 AnchorPane mainPage = loader.load();  
 rootLayout.setCenter(mainPage);  
 ConnectionErrorController controller = loader.getController();  
 controller.initialize(this);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Окно создания новой комнаты  
 \*/* public void NewRoom() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(App.class.getResource("views/CreateNewRoomView.fxml"));  
 AnchorPane page = loader.load();  
  
 Stage dialogStage = new Stage();  
 dialogStage.setTitle("Создание диалога");  
 dialogStage.initModality(Modality.*WINDOW\_MODAL*);  
 dialogStage.initOwner(primaryStage);  
 Scene scene = new Scene(page);  
 dialogStage.setScene(scene);  
 CreateNewRoomController controller = loader.getController();  
 controller.initialize(this, dialogStage);  
 dialogStage.showAndWait();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Показывает участников текущей комнаты  
 \*/* public void showCurrentRoomUsers() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(App.class.getResource("views/ShowChatUsersView.fxml"));  
 AnchorPane page = loader.load();  
 Stage dialogStage = new Stage();  
 dialogStage.initModality(Modality.*WINDOW\_MODAL*);  
 dialogStage.initOwner(primaryStage);  
 Scene scene = new Scene(page);  
 dialogStage.setScene(scene);  
 ShowChatUsersController controller = loader.getController();  
 controller.initialize(this, dialogStage);  
 dialogStage.showAndWait();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Базовый Layout  
 \*/* public void initRootLayout() {  
 try {  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(App.class.getResource("views/RootLayoutView.fxml"));  
 rootLayout = loader.load();  
 Scene scene = new Scene(rootLayout);  
 primaryStage.setScene(scene);  
 RootLayoutController controller = loader.getController();  
 controller.initialize(this);  
 primaryStage.show();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 */\*\*  
 \* Gets my api.  
 \*  
 \* @return the my api  
 \*/* public MyAPI getMyAPI() {  
 return myAPI;  
 }  
  
 */\*\*  
 \* Sets my api.  
 \*  
 \* @param myAPI the my api  
 \*/* public void setMyAPI(MyAPI myAPI) {  
 this.myAPI = myAPI;  
 }  
  
 */\*\*  
 \* Gets primary stage.  
 \*  
 \* @return the primary stage  
 \*/* public Stage getPrimaryStage() {  
 return primaryStage;  
 }  
  
 */\*\*  
 \* Gets auth util.  
 \*  
 \* @return the auth util  
 \*/* public AuthUtil getAuthUtil() {  
 return authUtil;  
 }  
  
}

src/main/java/module-info.java

module org.demka {  
 requires javafx.controls;  
 requires javafx.fxml;  
 requires com.jfoenix;  
 requires com.google.gson;  
 requires org.slf4j;  
 requires java.desktop;  
  
 opens org.demka to javafx.fxml;  
 opens org.demka.controllers to javafx.fxml;  
  
 exports org.demka;  
 exports org.demka.controllers;  
}

src/main/resources/org/demka/views/AboutMeView.fxml

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import com.jfoenix.controls.JFXButton*?>  
<?*import javafx.scene.control.Hyperlink*?>  
<?*import javafx.scene.control.Label*?>  
<?*import javafx.scene.layout.\**?>  
<?*import javafx.scene.text.Font*?>*<AnchorPane xmlns:fx="http://javafx.com/fxml/1" prefHeight="250.0" prefWidth="400.0" styleClass="background"  
 stylesheets="@../style.css" xmlns="http://javafx.com/javafx/11.0.1"  
 fx:controller="org.demka.controllers.AboutMeController">  
 <Label alignment="CENTER" contentDisplay="CENTER" layoutX="60.0" layoutY="110.0"  
 text="Чат с использованием LongPoll" textFill="WHITE" AnchorPane.leftAnchor="50.0"  
 AnchorPane.rightAnchor="50.0" AnchorPane.topAnchor="50.0">  
 <font>  
 <Font name="Helvetica" size="19.0"/>  
 </font>  
 </Label>  
  
 <Hyperlink fx:id="siteLink" alignment="CENTER" contentDisplay="CENTER" layoutX="145.0" layoutY="186.0"  
 onAction="#siteLinkClicked" text="https://georgiydemo.github.io/" textFill="WHITE"  
 AnchorPane.leftAnchor="100.0" AnchorPane.rightAnchor="100.0" AnchorPane.topAnchor="120.0">  
 <font>  
 <Font name="Helvetica" size="13.0"/>  
 </font>  
 </Hyperlink>  
  
 <Label alignment="CENTER" contentDisplay="CENTER" layoutX="60.0" layoutY="110.0" text="Деменчук Георгий, ПИ19-4"  
 textFill="WHITE" AnchorPane.leftAnchor="50.0" AnchorPane.rightAnchor="50.0" AnchorPane.topAnchor="90.0">  
 <font>  
 <Font name="Helvetica" size="13.0"/>  
 </font>  
 </Label>  
  
 <JFXButton layoutX="250.0" layoutY="380.0" onAction="#okButtonClicked" prefHeight="40.0" prefWidth="158.0"  
 ripplerFill="#2491bf" text="Ок" textFill="WHITE" AnchorPane.bottomAnchor="40.0"  
 AnchorPane.leftAnchor="100.0" AnchorPane.rightAnchor="100.0"/>  
  
</AnchorPane>

src/main/resources/org/demka/views/ConnectionErrorView.fxml

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import javafx.scene.control.Label*?>  
<?*import javafx.scene.layout.\**?>  
<?*import javafx.scene.text.Font*?>*<AnchorPane xmlns:fx="http://javafx.com/fxml/1" prefHeight="650.0" prefWidth="700.0" styleClass="background"  
 stylesheets="@../style.css" xmlns="http://javafx.com/javafx/11.0.1"  
 fx:controller="org.demka.controllers.ConnectionErrorController">  
 <Label alignment="CENTER" contentDisplay="CENTER" layoutX="240.0" layoutY="180.0"  
 text="Нет подключения к серверу.." textFill="WHITE" AnchorPane.bottomAnchor="180.0"  
 AnchorPane.leftAnchor="150.0" AnchorPane.rightAnchor="150.0" AnchorPane.topAnchor="180.0">  
 <font>  
 <Font name="Helvetica" size="24.0"/>  
 </font>  
 </Label>  
</AnchorPane>

src/main/resources/org/demka/views/CreateNewRoomView.fxml

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import com.jfoenix.controls.\**?>  
<?*import javafx.scene.control.\**?>  
<?*import javafx.scene.layout.\**?>*<AnchorPane xmlns:fx="http://javafx.com/fxml/1" prefHeight="373.0" prefWidth="650.0" styleClass="background"  
 stylesheets="@../style.css" xmlns="http://javafx.com/javafx/15.0.1"  
 fx:controller="org.demka.controllers.CreateNewRoomController">  
 <SplitPane dividerPositions="0.5" layoutX="183.0" layoutY="42.0" prefHeight="258.0" prefWidth="600.0"  
 AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"  
 AnchorPane.topAnchor="42.0">  
 <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="100.0" styleClass="background">  
  
 <TableView fx:id="allUsersTable" layoutX="-39.0" prefHeight="206.0" prefWidth="321.0"  
 AnchorPane.bottomAnchor="80.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"  
 AnchorPane.topAnchor="0.0">  
 <columns>  
 <TableColumn fx:id="allUsersColumn" editable="false" prefWidth="75.0" text="Все пользователи"/>  
 </columns>  
 <columnResizePolicy>  
 <TableView fx:constant="CONSTRAINED\_RESIZE\_POLICY"/>  
 </columnResizePolicy>  
 </TableView>  
 <JFXTextField fx:id="searchUser" prefHeight="27.0" prefWidth="287.0" promptText="Поиск пользователя"  
 unFocusColor="WHITE" AnchorPane.bottomAnchor="50.0" AnchorPane.leftAnchor="15.0"  
 AnchorPane.rightAnchor="15.0"/>  
 <JFXButton fx:id="addUserToChat" onAction="#addUserToChat" text="Добавить" AnchorPane.bottomAnchor="10.0"  
 AnchorPane.leftAnchor="10.0" AnchorPane.rightAnchor="10.0"/>  
 </AnchorPane>  
 <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="100.0" styleClass="background">  
 <TableView fx:id="chatUsersTable" prefHeight="207.0" prefWidth="295.0" AnchorPane.bottomAnchor="49.0"  
 AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">  
 <columns>  
 <TableColumn fx:id="chatUsersColumn" editable="false" prefWidth="96.03500366210938"  
 text="Участники"/>  
 </columns>  
 <columnResizePolicy>  
 <TableView fx:constant="CONSTRAINED\_RESIZE\_POLICY"/>  
 </columnResizePolicy>  
 </TableView>  
 <JFXButton onAction="#removeUserFromChat" text="Удалить" AnchorPane.bottomAnchor="10.0"  
 AnchorPane.leftAnchor="10.0" AnchorPane.rightAnchor="10.0"/>  
 </AnchorPane>  
 </SplitPane>  
 <JFXTextField fx:id="chatName" layoutX="14.0" layoutY="14.0" prefHeight="27.0" prefWidth="287.0"  
 promptText="Введите название диалога" unFocusColor="WHITE" AnchorPane.leftAnchor="10.0"  
 AnchorPane.rightAnchor="165.0" AnchorPane.topAnchor="10.0"/>  
 <HBox prefHeight="10.0" prefWidth="144.0" spacing="5.0" AnchorPane.rightAnchor="10.0" AnchorPane.topAnchor="10.0">  
 <JFXButton onAction="#cancelButtonClicked" text="Отмена"/>  
 <JFXButton fx:id="createRoomButton" disable="true" layoutX="10.0" layoutY="10.0"  
 onAction="#createRoomButtonClicked" text="Создать"/>  
 </HBox>  
</AnchorPane>

src/main/resources/org/demka/views/ForgotPasswordView.fxml

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import com.jfoenix.controls.\**?>  
<?*import javafx.scene.control.Label*?>  
<?*import javafx.scene.layout.\**?>  
<?*import javafx.scene.text.Font*?>*<AnchorPane xmlns:fx="http://javafx.com/fxml/1" prefHeight="650.0" prefWidth="700.0" styleClass="background"  
 stylesheets="@../style.css" xmlns="http://javafx.com/javafx/11.0.1"  
 fx:controller="org.demka.controllers.ForgotPasswordController">  
 <JFXButton layoutX="14.0" layoutY="36.0" onAction="#backButtonClicked" prefHeight="46.0" prefWidth="40.0"  
 ripplerFill="#2491bf" text="&lt;-" textFill="WHITE" AnchorPane.leftAnchor="15.0"  
 AnchorPane.topAnchor="15.0"/>  
  
 <AnchorPane prefHeight="306.0" prefWidth="200.0" AnchorPane.leftAnchor="200.0" AnchorPane.rightAnchor="200.0"  
 AnchorPane.topAnchor="170.0">  
 <JFXButton onAction="#resetButtonClicked" prefHeight="40.0" prefWidth="143.0" ripplerFill="#2491bf"  
 text="Восстановить" textFill="WHITE" AnchorPane.bottomAnchor="20.0"  
 AnchorPane.leftAnchor="50.0" AnchorPane.rightAnchor="50.0"/>  
 <Label alignment="CENTER" contentDisplay="CENTER" text="Восстановление" textFill="WHITE"  
 AnchorPane.leftAnchor="50.0" AnchorPane.rightAnchor="50.0" AnchorPane.topAnchor="0.0">  
 <font>  
 <Font name="Helvetica" size="24.0"/>  
 </font>  
 </Label>  
  
 <JFXTextField fx:id="loginTextField" focusColor="WHITE" promptText="E-mail" unFocusColor="WHITE"  
 AnchorPane.leftAnchor="50.0" AnchorPane.rightAnchor="50.0" AnchorPane.topAnchor="50.0">  
 <font>  
 <Font name="Helvetica" size="13.0"/>  
 </font>  
 </JFXTextField>  
  
 <JFXPasswordField fx:id="masterKeyTextField" focusColor="WHITE" promptText="Мастер-пароль"  
 unFocusColor="WHITE" AnchorPane.leftAnchor="50.0" AnchorPane.rightAnchor="50.0"  
 AnchorPane.topAnchor="108.0"/>  
 <Label fx:id="errorDescription" alignment="CENTER" contentDisplay="CENTER" text="какая-нибудь ошибка"  
 textFill="WHITE" AnchorPane.bottomAnchor="75.0" AnchorPane.leftAnchor="0.0"  
 AnchorPane.rightAnchor="0.0">  
 <font>  
 <Font size="11.0"/>  
 </font>  
 </Label>  
 <JFXPasswordField fx:id="newPasswordTextField" focusColor="WHITE" layoutX="10.0" layoutY="134.0"  
 promptText="Новый пароль" unFocusColor="WHITE" AnchorPane.leftAnchor="50.0"  
 AnchorPane.rightAnchor="50.0" AnchorPane.topAnchor="166.0"/>  
 </AnchorPane>  
</AnchorPane>

src/main/resources/org/demka/views/LoginView.fxml

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import com.jfoenix.controls.JFXButton*?>  
<?*import com.jfoenix.controls.JFXCheckBox*?>  
<?*import com.jfoenix.controls.JFXPasswordField*?>  
<?*import com.jfoenix.controls.JFXTextField*?>  
<?*import javafx.scene.control.Hyperlink*?>  
<?*import javafx.scene.control.Label*?>  
<?*import javafx.scene.layout.\**?>  
<?*import javafx.scene.text.\**?>*<AnchorPane xmlns:fx="http://javafx.com/fxml/1" prefHeight="650.0" prefWidth="700.0" styleClass="background"  
 stylesheets="@../style.css" xmlns="http://javafx.com/javafx/11.0.1"  
 fx:controller="org.demka.controllers.LoginController">  
 <AnchorPane prefHeight="306.0" prefWidth="200.0" AnchorPane.leftAnchor="250.0" AnchorPane.rightAnchor="250.0"  
 AnchorPane.topAnchor="170.0">  
  
 <JFXButton onAction="#mainButtonClick" prefHeight="40.0" prefWidth="143.0" ripplerFill="#2491bf"  
 text="Войти" textFill="WHITE" AnchorPane.bottomAnchor="47.0" AnchorPane.leftAnchor="0.0"  
 AnchorPane.rightAnchor="0.0"/>  
 <Label alignment="CENTER" contentDisplay="CENTER" text="DEMKA CHAT" textFill="WHITE"  
 AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">  
 <font>  
 <Font name="Helvetica" size="24.0"/>  
 </font>  
 </Label>  
  
 <JFXTextField fx:id="loginTextField" focusColor="WHITE" promptText="E-mail" unFocusColor="WHITE"  
 AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="66.0">  
 <font>  
 <Font name="Helvetica" size="13.0"/>  
 </font>  
 </JFXTextField>  
  
 <JFXPasswordField fx:id="passwordTextField" focusColor="WHITE" promptText="Пароль" unFocusColor="WHITE"  
 AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"  
 AnchorPane.topAnchor="124.0">  
 </JFXPasswordField>  
  
 <JFXCheckBox fx:id="autoLoginCheckBox" alignment="CENTER" checkedColor="#2167e4" prefHeight="14.0"  
 prefWidth="88.0" text="Запомнить меня" textFill="WHITE" unCheckedColor="WHITE"  
 AnchorPane.bottomAnchor="109.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0">  
 <font>  
 <Font size="11.0"/>  
 </font>  
 </JFXCheckBox>  
 <Label fx:id="wrongAuth" alignment="CENTER" contentDisplay="CENTER" opacity="0.0"  
 text="Неверный логин/пароль!" textFill="WHITE" AnchorPane.leftAnchor="0.0"  
 AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="40.0">  
 <font>  
 <Font size="11.0"/>  
 </font>  
 </Label>  
  
 <Hyperlink alignment="CENTER\_RIGHT" contentDisplay="RIGHT" onAction="#forgotPasswordLinkClicked"  
 text="Забыли пароль?" textFill="WHITE" AnchorPane.bottomAnchor="0.0"  
 AnchorPane.rightAnchor="0.0">  
 <font>  
 <Font size="11.0"/>  
 </font>  
 </Hyperlink>  
 <Hyperlink onAction="#regLinkClicked" text="Регистрация" textFill="WHITE" AnchorPane.bottomAnchor="0.0"  
 AnchorPane.leftAnchor="0.0">  
 <font>  
 <Font size="11.0"/>  
 </font>  
 </Hyperlink>  
 </AnchorPane>  
  
</AnchorPane>

src/main/resources/org/demka/views/MainChatView.fxml

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import com.jfoenix.controls.JFXButton*?>  
<?*import com.jfoenix.controls.JFXTextField*?>  
<?*import javafx.scene.control.\**?>  
<?*import javafx.scene.layout.\**?>*<AnchorPane xmlns:fx="http://javafx.com/fxml/1" prefHeight="650.0" prefWidth="700.0" styleClass="background"  
 stylesheets="@../style.css" xmlns="http://javafx.com/javafx/11.0.1"  
 fx:controller="org.demka.controllers.MainChatController">  
 <MenuBar fx:id="mainMenuBar" prefHeight="27.0" prefWidth="600.0" AnchorPane.leftAnchor="0.0"  
 AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">  
 <Menu fx:id="aboutMenuItem" mnemonicParsing="false" onAction="#aboutMenuItemClicked" text="О программе">  
 </Menu>  
 <Menu fx:id="exitMenuItem" mnemonicParsing="false" onAction="#exitMenuItemClicked" text="Выход">  
 </Menu>  
 </MenuBar>  
 <SplitPane dividerPositions="0.298" layoutX="183.0" layoutY="27.0" prefHeight="273.0" prefWidth="600.0"  
 AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"  
 AnchorPane.topAnchor="27.0">  
 <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="174.0" styleClass="background">  
 <TableView fx:id="roomTable" layoutX="-39.0" layoutY="14.0" prefHeight="298.0" prefWidth="175.0"  
 AnchorPane.bottomAnchor="45.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"  
 AnchorPane.topAnchor="0.0">  
 <columns>  
 <TableColumn fx:id="roomColumn" editable="false" prefWidth="75.0" sortable="false" text="Диалоги"/>  
 <TableColumn fx:id="newMessageFlagColumn" editable="false" maxWidth="20.0" minWidth="20.0"  
 prefWidth="20.0" text=" "/>  
 </columns>  
 <columnResizePolicy>  
 <TableView fx:constant="CONSTRAINED\_RESIZE\_POLICY"/>  
 </columnResizePolicy>  
 </TableView>  
 <JFXButton onAction="#createRoomButtonClicked" text="Создать" AnchorPane.bottomAnchor="10.0"  
 AnchorPane.leftAnchor="10.0" AnchorPane.rightAnchor="10.0"/>  
 </AnchorPane>  
 <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="160.0" prefWidth="100.0" styleClass="background">  
 <TableView fx:id="messageTable" prefHeight="254.0" prefWidth="416.0" AnchorPane.bottomAnchor="45.0"  
 AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">  
 <columns>  
 <TableColumn fx:id="messageTimeColumn" editable="false" maxWidth="70" minWidth="70" prefWidth="70.0"  
 sortable="false" text="Время"/>  
 <TableColumn fx:id="messageUserColumn" editable="false" maxWidth="150" minWidth="100"  
 prefWidth="150.0" sortable="false" text="Пользователь"/>  
 <TableColumn fx:id="messageTextColumn" editable="false" prefWidth="317.9650573730469"  
 sortable="false" text="Сообщение"/>  
 </columns>  
 <columnResizePolicy>  
 <TableView fx:constant="CONSTRAINED\_RESIZE\_POLICY"/>  
 </columnResizePolicy>  
 </TableView>  
 <JFXButton fx:id="chatUsersButton" layoutX="211.0" layoutY="264.0" mnemonicParsing="false"  
 onAction="#chatUsersButtonClicked" prefHeight="20.0" prefWidth="20.0"  
 styleClass="chat-users-button" text=" " AnchorPane.rightAnchor="10.0"  
 AnchorPane.topAnchor="0.0"/>  
  
 <JFXButton fx:id="sendMessageButton" defaultButton="true" disable="true"  
 onAction="#sendMessageButtonClicked" text="Отправить" AnchorPane.bottomAnchor="10.0"  
 AnchorPane.rightAnchor="10.0"/>  
 <JFXTextField fx:id="newMessageText" prefHeight="27.0" prefWidth="300.0" promptText="Введите сообщение.."  
 unFocusColor="WHITE" AnchorPane.bottomAnchor="10.0" AnchorPane.leftAnchor="10.0"  
 AnchorPane.rightAnchor="100.0"/>  
 </AnchorPane>  
 </SplitPane>  
</AnchorPane>

src/main/resources/org/demka/views/RegistrationView.fxml

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import com.jfoenix.controls.\**?>  
<?*import javafx.scene.control.\**?>  
<?*import javafx.scene.layout.\**?>  
<?*import javafx.scene.text.\**?>*<AnchorPane xmlns:fx="http://javafx.com/fxml/1" prefHeight="650.0" prefWidth="700.0" styleClass="background"  
 stylesheets="@../style.css" xmlns="http://javafx.com/javafx/11.0.1"  
 fx:controller="org.demka.controllers.RegistrationController">  
 <Label alignment="CENTER" contentDisplay="CENTER" layoutX="206.0" layoutY="53.0" text="DEMKA CHAT - Регистрация"  
 textFill="WHITE" AnchorPane.leftAnchor="180.0" AnchorPane.rightAnchor="180.0"  
 AnchorPane.topAnchor="60.0">  
 <font>  
 <Font name="Helvetica" size="24.0"/>  
 </font>  
 </Label>  
  
 <JFXButton layoutX="14.0" layoutY="36.0" onAction="#backButtonClicked" prefHeight="46.0" prefWidth="40.0"  
 ripplerFill="#2491bf" text="&lt;-" textFill="WHITE" AnchorPane.leftAnchor="15.0"  
 AnchorPane.topAnchor="15.0"/>  
  
 <Label fx:id="errorDescription" alignment="CENTER" contentDisplay="CENTER" layoutX="310.0" layoutY="103.0"  
 text="ОПИСАНИЕ ОШИБКИ" textFill="WHITE" AnchorPane.leftAnchor="180.0" AnchorPane.rightAnchor="180.0"  
 AnchorPane.topAnchor="115.0">  
 <font>  
 <Font size="11.0"/>  
 </font>  
 </Label>  
  
 <JFXButton layoutX="289.0" layoutY="564.0" onAction="#regButtonClicked" prefHeight="40.0" prefWidth="158.0"  
 ripplerFill="#2491bf" text="Зарегистрироваться" textFill="WHITE" AnchorPane.bottomAnchor="40.0"  
 AnchorPane.leftAnchor="250.0" AnchorPane.rightAnchor="250.0"/>  
  
 <TitledPane animated="false" collapsible="false" layoutX="62.0" layoutY="361.0" prefHeight="173.0"  
 prefWidth="533.0" text="Пароли" AnchorPane.leftAnchor="65.0" AnchorPane.rightAnchor="65.0">  
 <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="180.0" prefWidth="200.0">  
 <JFXPasswordField fx:id="passwordTextField" focusColor="WHITE" layoutX="14.0" layoutY="30.0"  
 promptText="Пароль" unFocusColor="WHITE"/>  
 <JFXPasswordField fx:id="masterKey" focusColor="WHITE" layoutX="14.0" layoutY="90.0"  
 promptText="Мастер-пароль" unFocusColor="WHITE"/>  
 <Label layoutX="179.0" layoutY="37.0" prefHeight="14.0" prefWidth="193.0"  
 text="Используется для входа в систему" textFill="WHITE" AnchorPane.rightAnchor="145.0">  
 <font>  
 <Font size="11.0"/>  
 </font>  
 </Label>  
 <Label layoutX="179.0" layoutY="97.0" prefHeight="14.0" prefWidth="240.0"  
 text="Используется для восстановления пароля" textFill="WHITE"  
 AnchorPane.rightAnchor="145.0">  
 <font>  
 <Font size="11.0"/>  
 </font>  
 </Label>  
 </AnchorPane>  
 </TitledPane>  
  
 <TitledPane animated="false" collapsible="false" layoutX="62.0" layoutY="151.0" prefHeight="186.0"  
 prefWidth="533.0" text="Данные" AnchorPane.leftAnchor="65.0" AnchorPane.rightAnchor="65.0">  
 <AnchorPane minHeight="0.0" minWidth="0.0" prefHeight="180.0" prefWidth="200.0">  
 <JFXTextField fx:id="userNameTextField" focusColor="WHITE" layoutX="14.0" layoutY="32.0"  
 promptText="Ник" unFocusColor="WHITE" AnchorPane.leftAnchor="15.0"/>  
 <JFXTextField fx:id="emailTextField" focusColor="WHITE" layoutX="14.0" layoutY="95.0"  
 promptText="E-mail" unFocusColor="WHITE" AnchorPane.leftAnchor="15.0"/>  
 <Label layoutX="174.0" layoutY="39.0" prefHeight="14.0" prefWidth="212.0"  
 text="Отображается другим пользователям" textFill="WHITE" AnchorPane.rightAnchor="145.0">  
 <font>  
 <Font size="11.0"/>  
 </font>  
 </Label>  
 <Label layoutX="174.0" layoutY="102.0" prefHeight="14.0" prefWidth="194.0"  
 text="Используется для входа в систему" textFill="WHITE" AnchorPane.rightAnchor="145.0">  
 <font>  
 <Font size="11.0"/>  
 </font>  
 </Label>  
 </AnchorPane>  
 </TitledPane>  
  
</AnchorPane>

src/main/resources/org/demka/views/RootLayoutView.fxml

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import javafx.scene.layout.BorderPane*?>*<BorderPane xmlns:fx="http://javafx.com/fxml" prefHeight="650.0" prefWidth="700.0" styleClass="background"  
 stylesheets="@../style.css" xmlns="https://javafx.com/javafx"  
 fx:controller="org.demka.controllers.RootLayoutController">  
</BorderPane>

src/main/resources/org/demka/views/ShowChatUsersView.fxml

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import com.jfoenix.controls.JFXButton*?>  
<?*import javafx.scene.control.TableColumn*?>  
<?*import javafx.scene.control.TableView*?>  
<?*import javafx.scene.layout.\**?>*<AnchorPane xmlns:fx="http://javafx.com/fxml/1" prefHeight="346.0" prefWidth="400.0" styleClass="background"  
 stylesheets="@../style.css" xmlns="http://javafx.com/javafx/11.0.1"  
 fx:controller="org.demka.controllers.ShowChatUsersController">  
 <TableView fx:id="chatUserTable" layoutY="88.0" prefHeight="117.0" prefWidth="400.0"  
 AnchorPane.bottomAnchor="60.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"  
 AnchorPane.topAnchor="0.0">  
 <columns>  
 <TableColumn fx:id="chatUserNameColumn" editable="false" prefWidth="150.0" text="Участники (N всего)"/>  
 </columns>  
 <columnResizePolicy>  
 <TableView fx:constant="CONSTRAINED\_RESIZE\_POLICY"/>  
 </columnResizePolicy>  
 </TableView>  
  
 <JFXButton onAction="#backButtonClicked" layoutX="250.0" layoutY="380.0" prefHeight="40.0" prefWidth="158.0"  
 ripplerFill="#2491bf" text="Ок" textFill="WHITE" AnchorPane.bottomAnchor="10.0"  
 AnchorPane.leftAnchor="100.0" AnchorPane.rightAnchor="100.0"/>  
  
</AnchorPane>

src/main/resources/org/demka/views/SuccessActionView.fxml

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import com.jfoenix.controls.JFXButton*?>  
<?*import javafx.scene.control.Label*?>  
<?*import javafx.scene.layout.\**?>  
<?*import javafx.scene.text.Font*?>*<AnchorPane xmlns:fx="http://javafx.com/fxml/1" prefHeight="650.0" prefWidth="700.0" styleClass="background"  
 stylesheets="@../style.css" xmlns="http://javafx.com/javafx/11.0.1"  
 fx:controller="org.demka.controllers.SuccessActionController">  
 <Label fx:id="mainText" alignment="CENTER" contentDisplay="CENTER" layoutX="145.0" layoutY="186.0"  
 text="ТУТ ДИНАМИЧЕСКИЙ ТЕКСТ" textFill="WHITE" AnchorPane.leftAnchor="150.0"  
 AnchorPane.rightAnchor="150.0" AnchorPane.topAnchor="280.0">  
 <font>  
 <Font name="Helvetica" size="24.0"/>  
 </font>  
 </Label>  
 <JFXButton layoutX="250.0" layoutY="380.0" onAction="#okButtonClicked" prefHeight="40.0" prefWidth="158.0"  
 ripplerFill="#2491bf" text="Ок" textFill="WHITE" AnchorPane.leftAnchor="250.0"  
 AnchorPane.rightAnchor="250.0" AnchorPane.topAnchor="350.0"/>  
</AnchorPane>

src/main/resources/org/demka/style.css

.background {  
 -fx-background-color: #202234;  
}  
  
.titled-pane {  
 -fx-background-color: transparent;  
}  
  
.titled-pane > .title {  
 -fx-background-color: transparent;  
 -fx-border-color: rgba(255, 255, 255, 0.5);  
 -fx-font-weight: bold;  
 -fx-border-radius: 1em 1em 0 0;  
 -fx-background-radius: 1em 1em 0 0;  
  
}  
  
.titled-pane > \*.content {  
 -fx-border-color: rgba(255, 255, 255, 0.5);  
 -fx-background-color: #202234;  
 -fx-border-radius: 0 0 1em 1em;  
 -fx-background-radius: 0 0 1em 1em;  
}  
  
  
.titled-pane > .title > .text {  
 -fx-fill: WHITE;  
}  
  
.hyperlink {  
 -fx-border-color: transparent;  
 -fx-padding: 4 0 4 0;  
}  
  
.split-pane {  
 -fx-background-color: -fx-control-inner-background;  
 -fx-background-insets: 0;  
 -fx-padding: 0;  
 -fx-border-insets: 0;  
 -fx-background-radius: 0;  
 -fx-border-radius: 0;  
 -fx-border-color: transparent;  
 -fx-border-width: 0;  
}  
  
.button {  
 -fx-background-color: #2167e4;  
 -fx-border-radius: 5em;  
 -fx-background-radius: 5em;  
}  
  
.chat-users-button {  
  
 -fx-background-color: #393f51;  
 -fx-background-image: url('images/group-icon.png');  
 -fx-background-size: 20px;  
 -fx-background-repeat: no-repeat;  
 -fx-background-position: center;  
}  
  
.chat-users-button:hover {  
 -fx-background: #202234;  
 -fx-opacity: 0.5;  
}  
  
.text-field {  
 -fx-text-inner-color: white;  
 -fx-text-fill: white;  
 -fx-prompt-text-fill: gray;  
}  
  
.menu-bar {  
 -fx-background-color: #393f51;  
 -fx-border-width: 2.0;  
  
}  
  
.menu-bar .label {  
 -fx-text-fill: white;  
 -fx-font-weight: bold;  
}  
  
.menu-bar .menu-button:hover, .menu-bar .menu-button:focused, .menu-bar .menu-button:showing {  
 -fx-background: #393f51;  
 -fx-text-fill: white;  
}  
  
.menu-bar .menu-button:hover, .menu-bar .menu-button:focused, .menu-bar .menu-button:showing {  
 -fx-background: #202234;  
 -fx-opacity: 0.5;  
 -fx-border-radius: 3em;  
 -fx-background-radius: 3em;  
}  
  
.table-view {  
 -fx-background-color: transparent;  
 -fx-table-cell-border-color: transparent;  
}  
  
.table-view .column-header-background .filler {  
 -fx-background-color: transparent;  
}  
  
.table-view .corner {  
 -fx-background-color: transparent;  
}  
  
.table-view .column-header {  
 -fx-background-color: transparent;  
}  
  
.table-view .cell {  
 -fx-background-color: transparent;  
}  
  
.root .table-view {  
 -fx-control-inner-background: #202234;  
}  
  
.table-row-cell:selected .table-cell {  
 -fx-background-color: #2d3143;  
 -fx-background: #2d3143;  
}  
  
*/\*  
Хедеры таблиц  
 \*/*.table-view .column-header,  
.table-view .column-header .filler,  
.table-view .column-header-background .filler {  
 -fx-background-color: #393f51;  
}  
  
.table-view .column-header .label {  
 -fx-text-fill: white;  
 -fx-font-weight: bold;  
}  
  
  
*/\*  
Сплитер  
 \*/*.split-pane:horizontal > .split-pane-divider,  
.split-pane:vertical > .split-pane-divider {  
 -fx-background-color: #393f51;  
}  
  
  
*/\*  
Scrollbar  
 \*/*.scroll-bar:horizontal .track,  
.scroll-bar:vertical .track {  
 -fx-background-color: transparent;  
 -fx-border-color: transparent;  
 -fx-background-radius: 0.0em;  
 -fx-border-radius: 2.0em;  
}  
  
.scroll-bar:horizontal .increment-button,  
.scroll-bar:horizontal .decrement-button {  
 -fx-background-color: transparent;  
 -fx-background-radius: 0.0em;  
 -fx-padding: 0.0 0.0 10.0 0.0;  
}  
  
.scroll-bar:vertical .increment-button,  
.scroll-bar:vertical .decrement-button {  
 -fx-background-color: transparent;  
 -fx-background-radius: 0.0em;  
 -fx-padding: 0.0 10.0 0.0 0.0;  
}  
  
.scroll-bar .increment-arrow,  
.scroll-bar .decrement-arrow {  
 -fx-shape: " ";  
 -fx-padding: 0.15em 0.0;  
}  
  
.scroll-bar:vertical .increment-arrow,  
.scroll-bar:vertical .decrement-arrow {  
 -fx-shape: " ";  
 -fx-padding: 0.0 0.15em;  
}  
  
.scroll-bar:vertical .thumb,  
.scroll-bar:vertical .thumb:hover {  
 -fx-background-color: #414352;  
 -fx-background-insets: 2.0, 0.0, 0.0;  
 -fx-background-radius: 2.0em;  
}  
  
.scroll-bar:horizontal .thumb,  
.scroll-bar:horizontal .thumb:hover {  
 -fx-background-color: transparent;  
 -fx-background-insets: 2.0, 0.0, 0.0;  
 -fx-background-radius: 2.0em;  
}  
  
  
.scroll-pane {  
 -fx-background-color: transparent;  
}  
  
.scroll-bar .increment-button {  
 visibility: hidden;  
}  
  
.scroll-bar .decrement-button {  
 visibility: hidden;  
}  
  
.scroll-bar:horizontal,  
.scroll-bar:vertical {  
 -fx-background-color: transparent;  
}  
  
.scroll-bar:horizontal .track-background,  
.scroll-bar:vertical .track-background {  
 visibility: hidden;  
}  
  
.scroll-bar:horizontal .track,  
.scroll-bar:vertical .track {  
 visibility: hidden;  
}  
  
.hide-thumb .scroll-bar:horizontal .thumb,  
.hide-thumb .scroll-bar:vertical .thumb {  
 -fx-background-color: transparent;  
}