Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**Департамент анализа данных,**

**принятия решений и финансовых технологий**

**Пояснительная записка к междисциплинарной курсовой работе**

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](#_Toc70176976)

[ПОСТАНОВКА ЗАДАЧИ 5](#_Toc70176977)

[ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc70176978)

[АКТУАЛЬНОСТЬ АВТОМАТИЗАЦИИ 8](#_Toc70176979)

[АЛГОРИТМИЧЕСКИЕ РЕШЕНИЯ 9](#_Toc70176980)

[Клиент 9](#_Toc70176981)

[Сервер 11](#_Toc70176982)

[ОПИСАНИЕ ИНТЕРФЕЙСА ПРОГРАММЫ 13](#_Toc70176983)

[Окно авторизации клиента 14](#_Toc70176984)

[Окно регистрации нового пользователя 15](#_Toc70176985)

[Окно восстановления пароля 16](#_Toc70176986)

[Окно успешного действия пользователя 17](#_Toc70176987)

[Главное окно программы 17](#_Toc70176988)

[Окно создания новой чат-комнаты 18](#_Toc70176989)

[Окно с информацией о текущей чат-комнате 19](#_Toc70176990)

[Окно с информацией об авторе и программе 20](#_Toc70176991)

[СОСТАВ ПРИЛОЖЕНИЯ 21](#_Toc70176992)

[Сервер 21](#_Toc70176993)

[База данных 22](#_Toc70176994)

[Клиент 23](#_Toc70176995)

[НАЗНАЧЕНИЕ И СОСТАВ КЛАССОВ ПРОГРАММЫ 25](#_Toc70176996)

[Сервер 25](#_Toc70176997)

[Клиент 29](#_Toc70176998)

[ЗАКЛЮЧЕНИЕ 34](#_Toc70176999)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 35](#_Toc70177000)

[ПРИЛОЖЕНИЯ 36](#_Toc70177001)

[Приложение 1 36](#_Toc70177002)

# 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

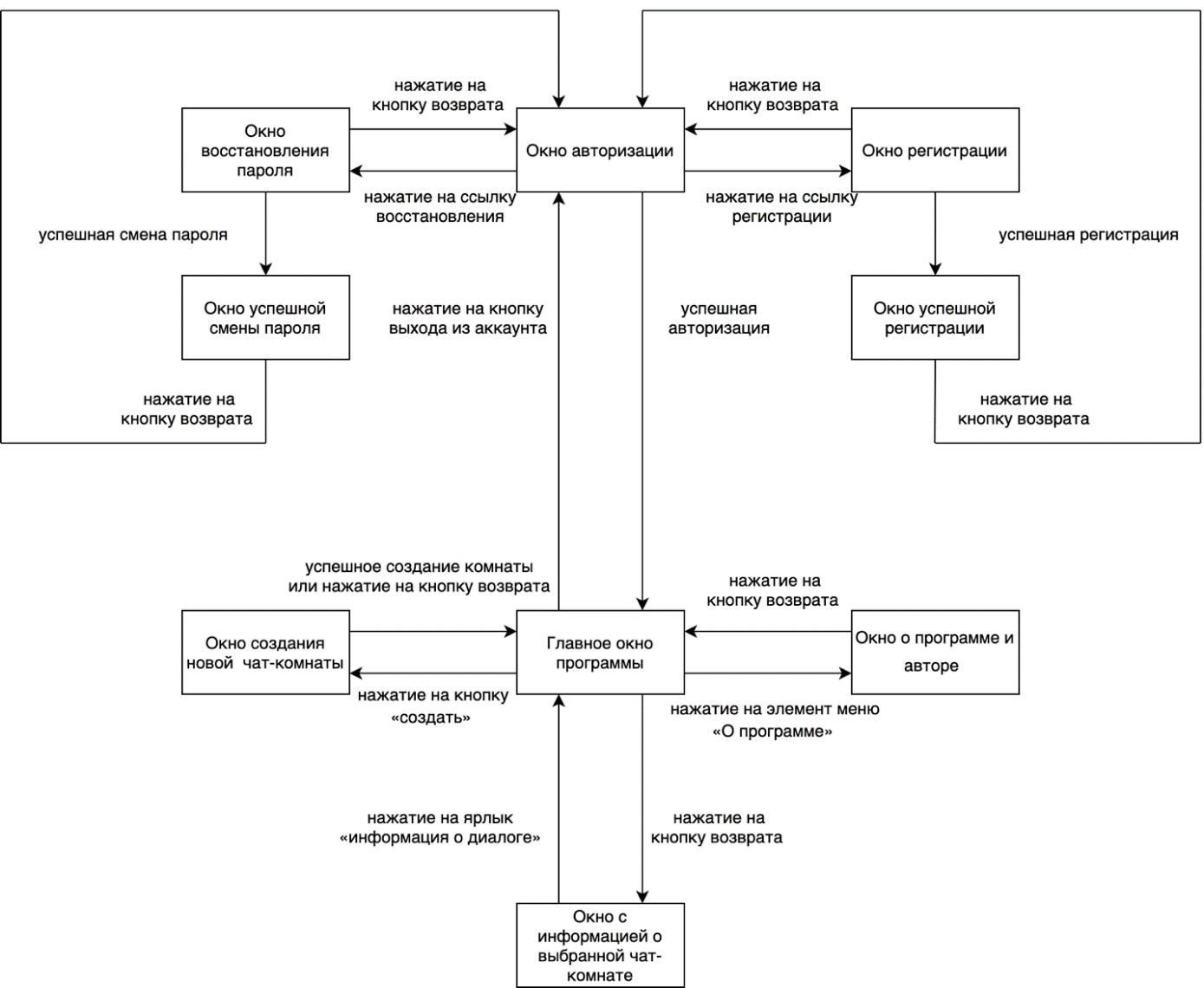
Автоматизация получения сообщений с помощью собственно разработанного решения позволяет оперативно передавать абсолютно любую информацию в текстовом виде, при этом храня все данные исключительно на своих серверах и базах данных, что критично для проектов с повышенным вниманием к конфиденциальности данных, 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

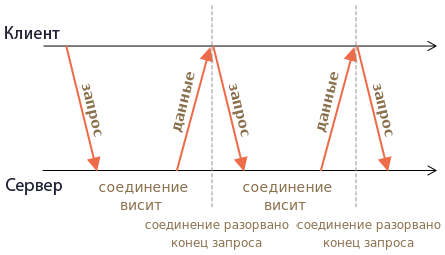
На схеме ниже представлены переходы пользователя между данными формами, кроме отдельного окна «Нет подключения к интернету» ‎. Оно не представлено на схеме т.к. переход на него может быть осуществлён абсолютно с любого окна программы и не зависит от действий пользователя. Данный элемент программы вызывается 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.



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. 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.

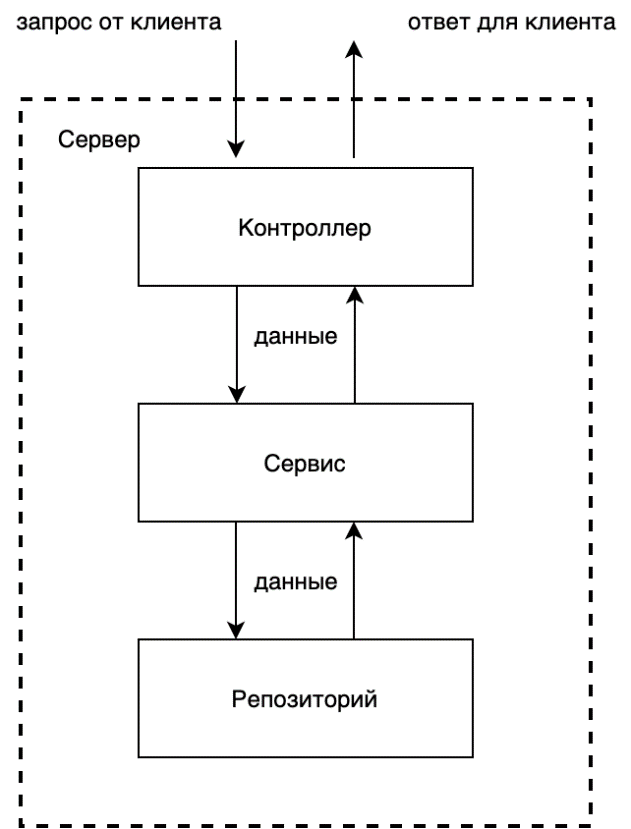


Если клиенту пришло событие о новом сообщении для пользователя, то он отобразит его в соответствующей комнате. Если такой комнаты нет, то добавит её в список чат-комнат пользователя. Если сервер сообщает клиенту, что время ожидания запроса истекло, то клиент обратится заново к серверу с просьбой создания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

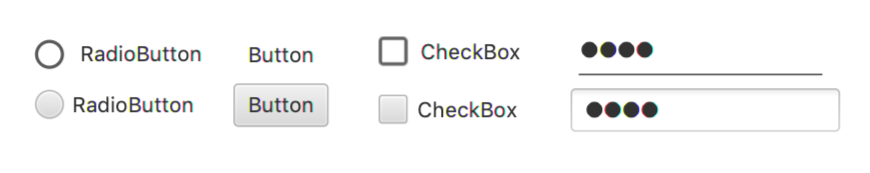


Если же все переданные данные корректны, то из соответствующего контроллера вызывается сервис с логикой, которая отвечает за обработку полученных данных и передачу их в методы репозитория (класса, который наследуется от интерфейса MongoRepository). После получения данных от репозитория, сервис обрабатывает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





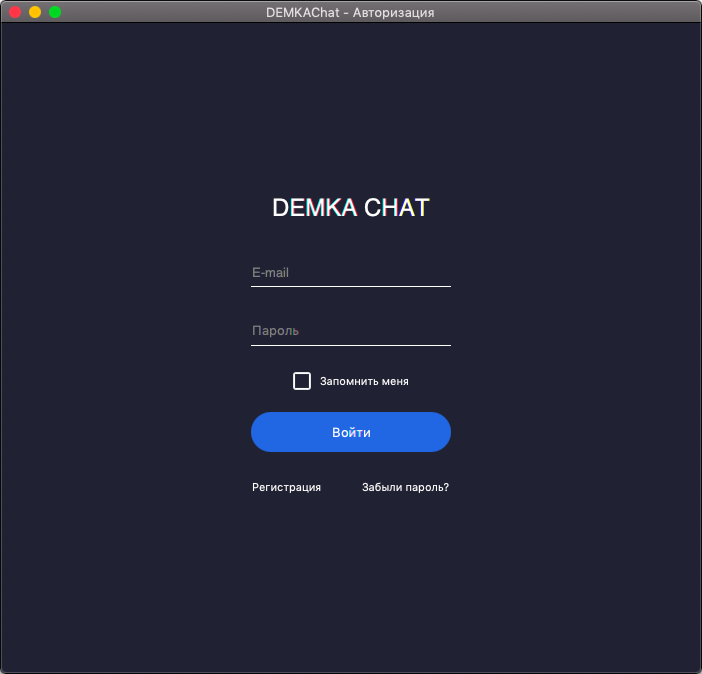
Как видим из рисунка выше, элементы JFoenix выглядят более аккуратно по сравнению со стандартным набором, что особенно хорошо видно на элементах Slider, ProgressBar и TextField.

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

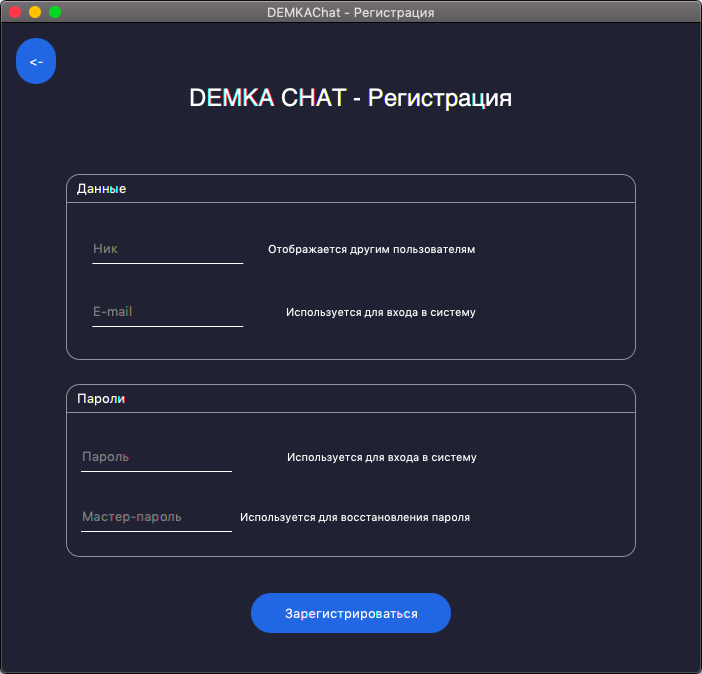
Запрашивает логин/пароль пользователя и предлагает запомнить информацию об авторизации для автоматического входа при следующем запуске программы. Если же пользователь еще не зарегистрирован в системе, то по нажатию на ссылку регистрации будет осуществлён переход на форму регистрации. Если пользователь 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.



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

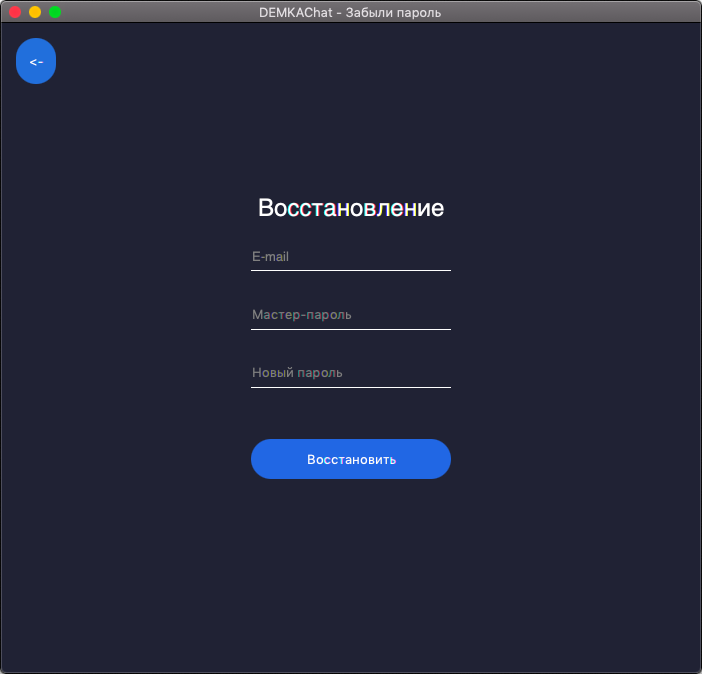
* Корректность ввода e-mail;
* Проверка на пустые поля;
* Проверка на длину пароля и мастер-пароля (не менее 8 символов);
* Проверка, чтоб пароль и мастер-пароль не совпадали между собой.

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.

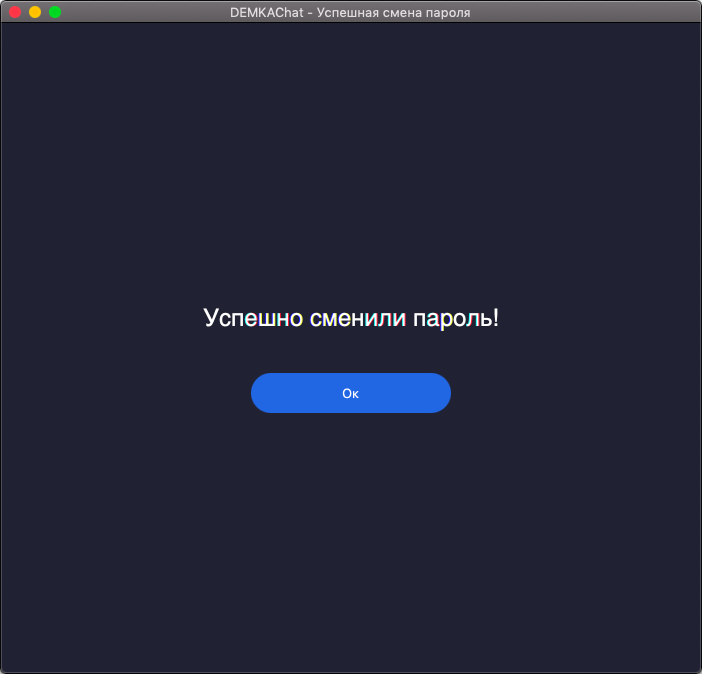
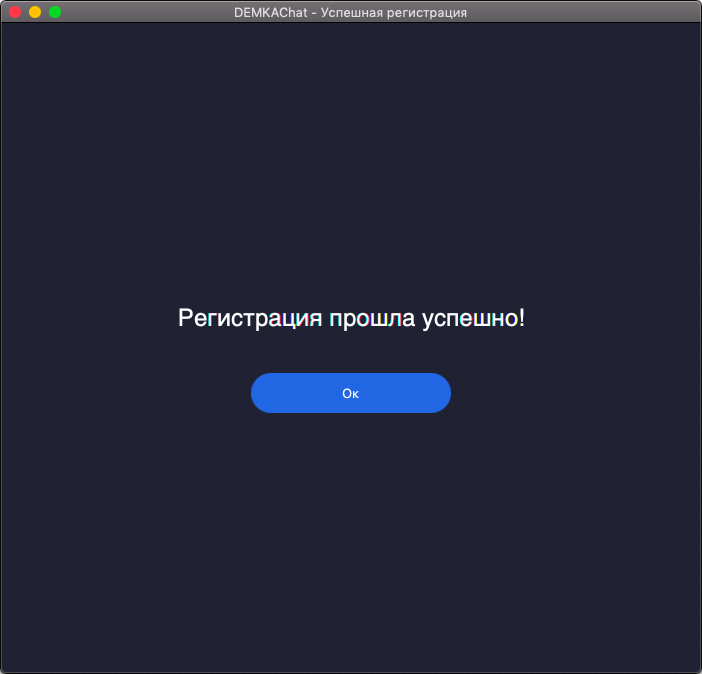
## Окно восстановления пароля



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. 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.

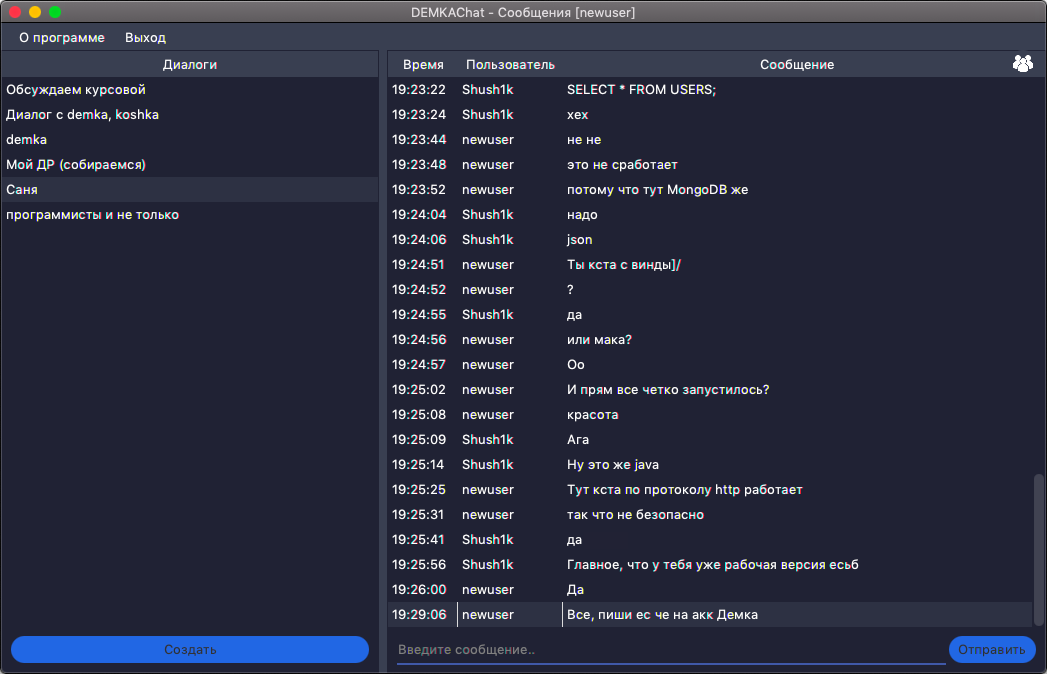
## Окно успешного действия пользователя

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.



## Главное окно программы

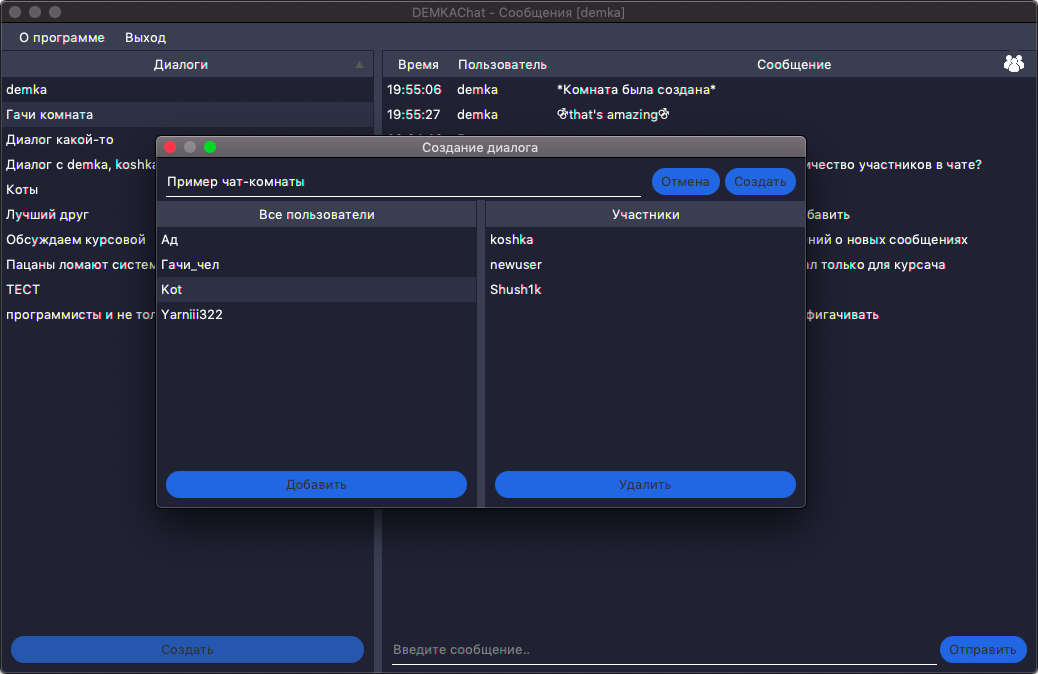
Слева окна расположены все чат-комнаты пользователя. При нажатии на какую-либо из них справа отобразятся сообщения, которые содержатся в данной комнате. Все новые сообщения и комнаты добавляются автоматически без каких-либо действий со стороны пользователя. Внизу находятся кнопки создания нового чата 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.



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.

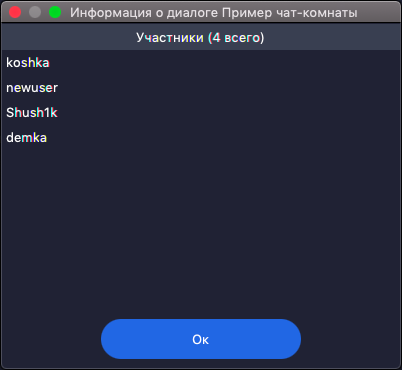
## Окно создания новой чат-комнаты

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.



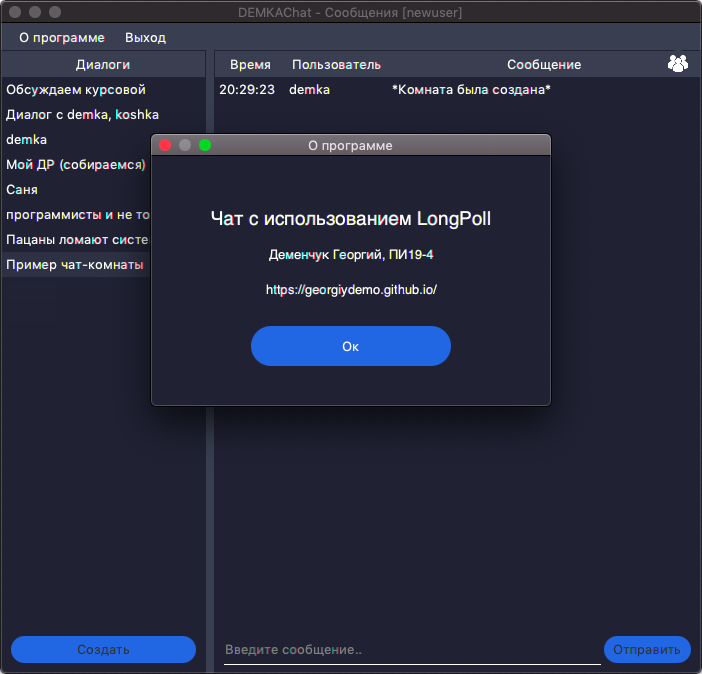
## Окно с информацией о текущей чат-комнате

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.



## 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.



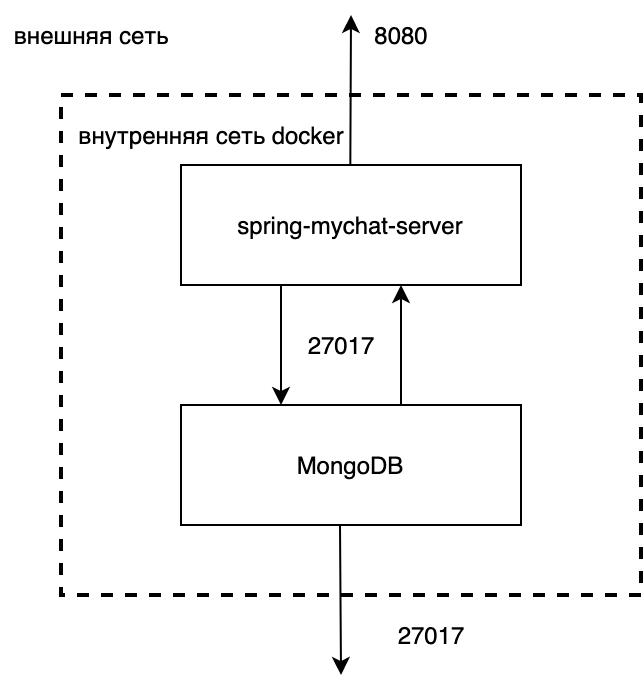
# СОСТАВ ПРИЛОЖЕНИЯ

## 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.



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 ниже. 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).

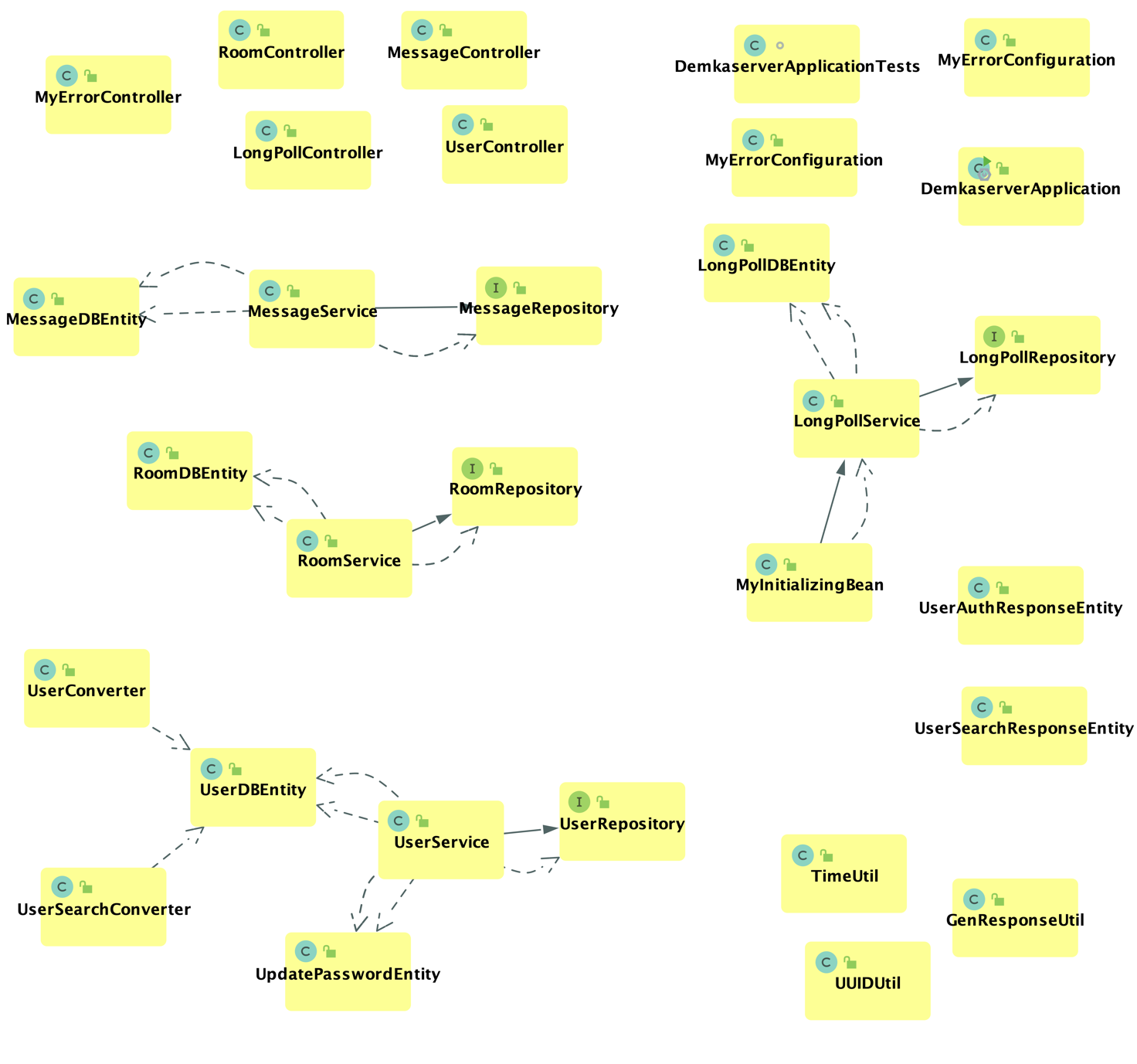
# НАЗНАЧЕНИЕ И СОСТАВ КЛАССОВ ПРОГРАММЫ

## Сервер

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

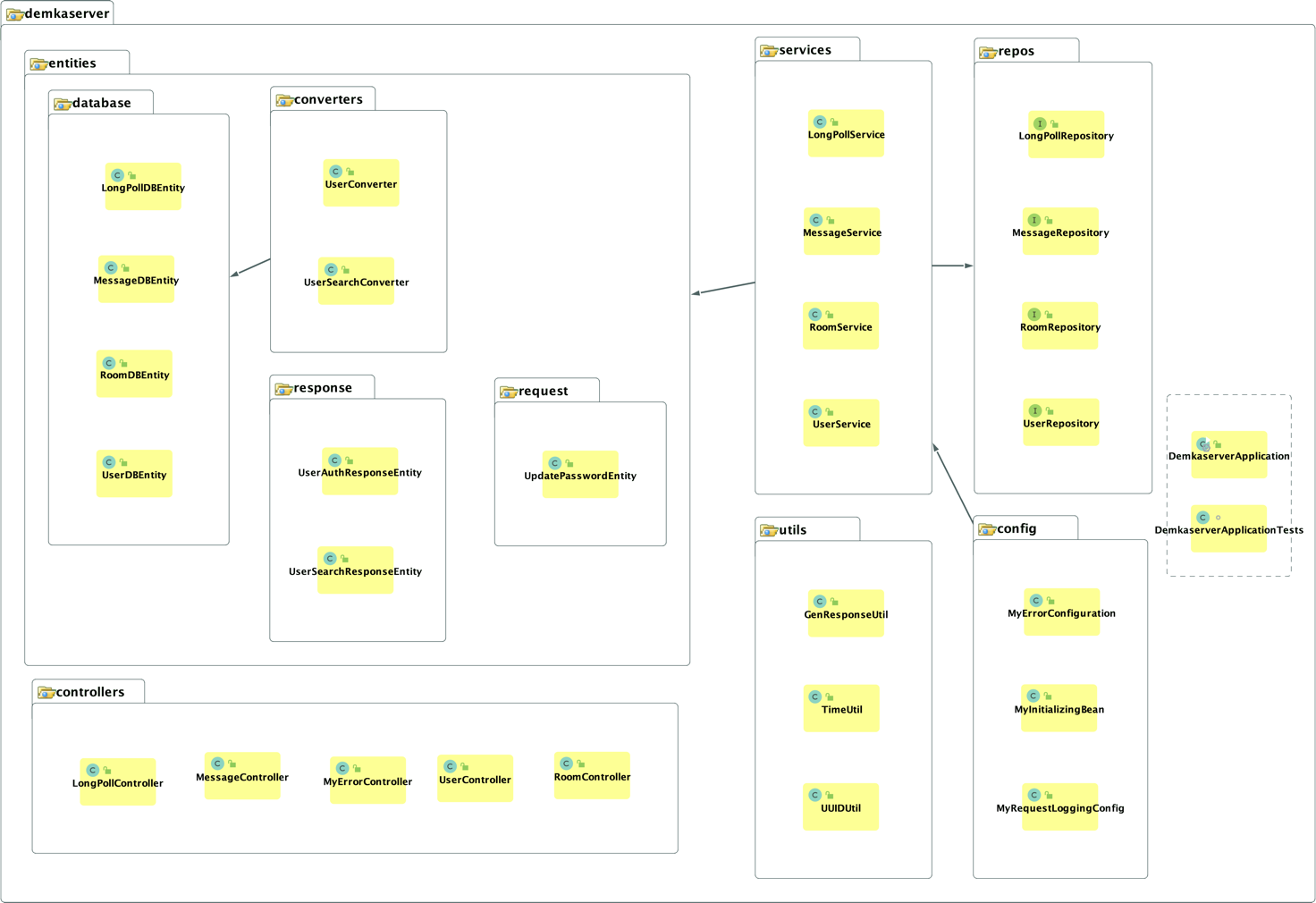
* Контроллеры (5 шт.) – классы для обработки входящих запросов от клиентов;
* Классы конфигураций и тестов (5 шт.) – необходимы для переопределения стандартной конфигурации Spring Boot в целях расширения существующего функционала;
* Классы-утилиты (3 шт.) – необходимы для обособленных решений простых 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 ниже:



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.

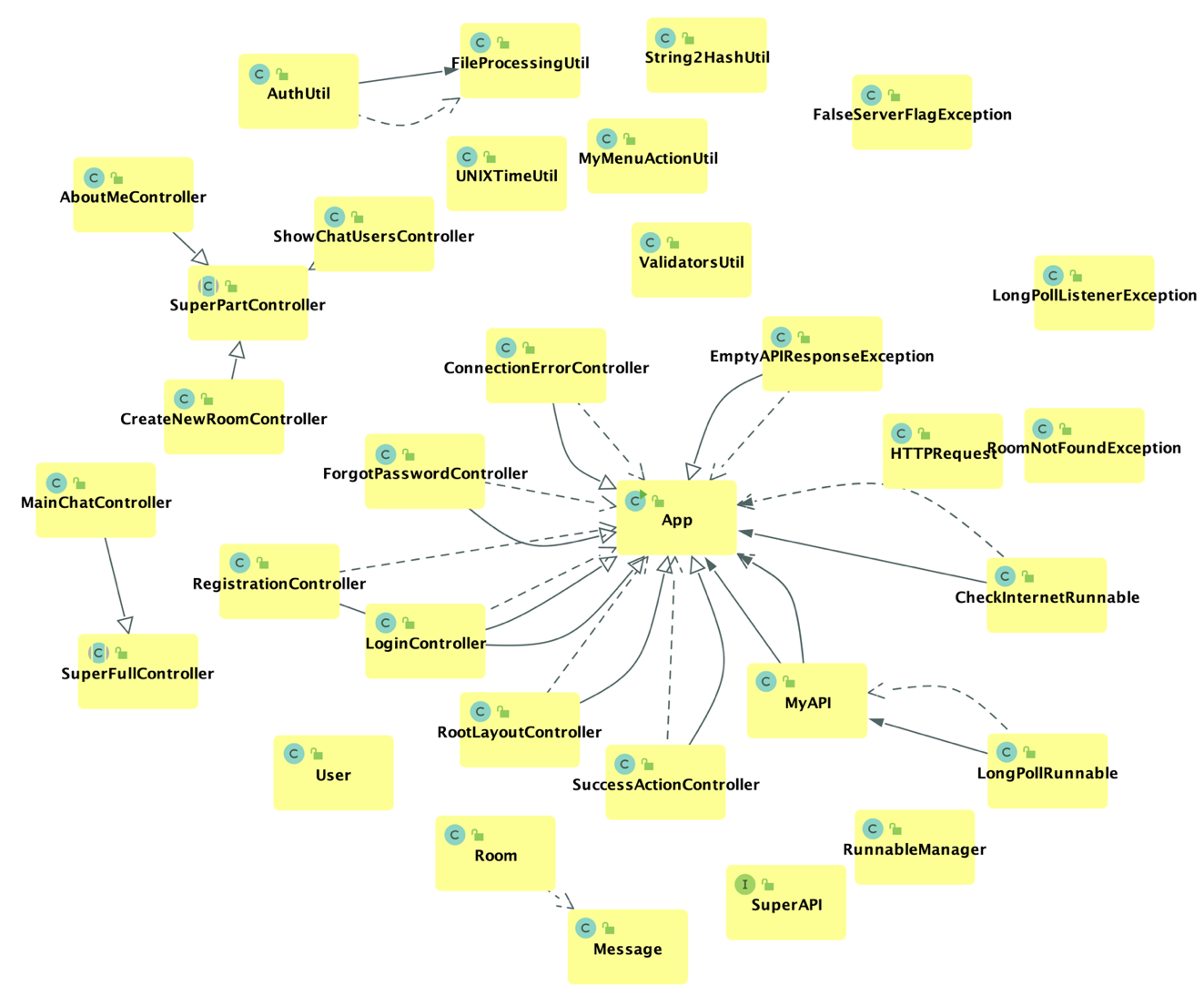


## Клиент

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

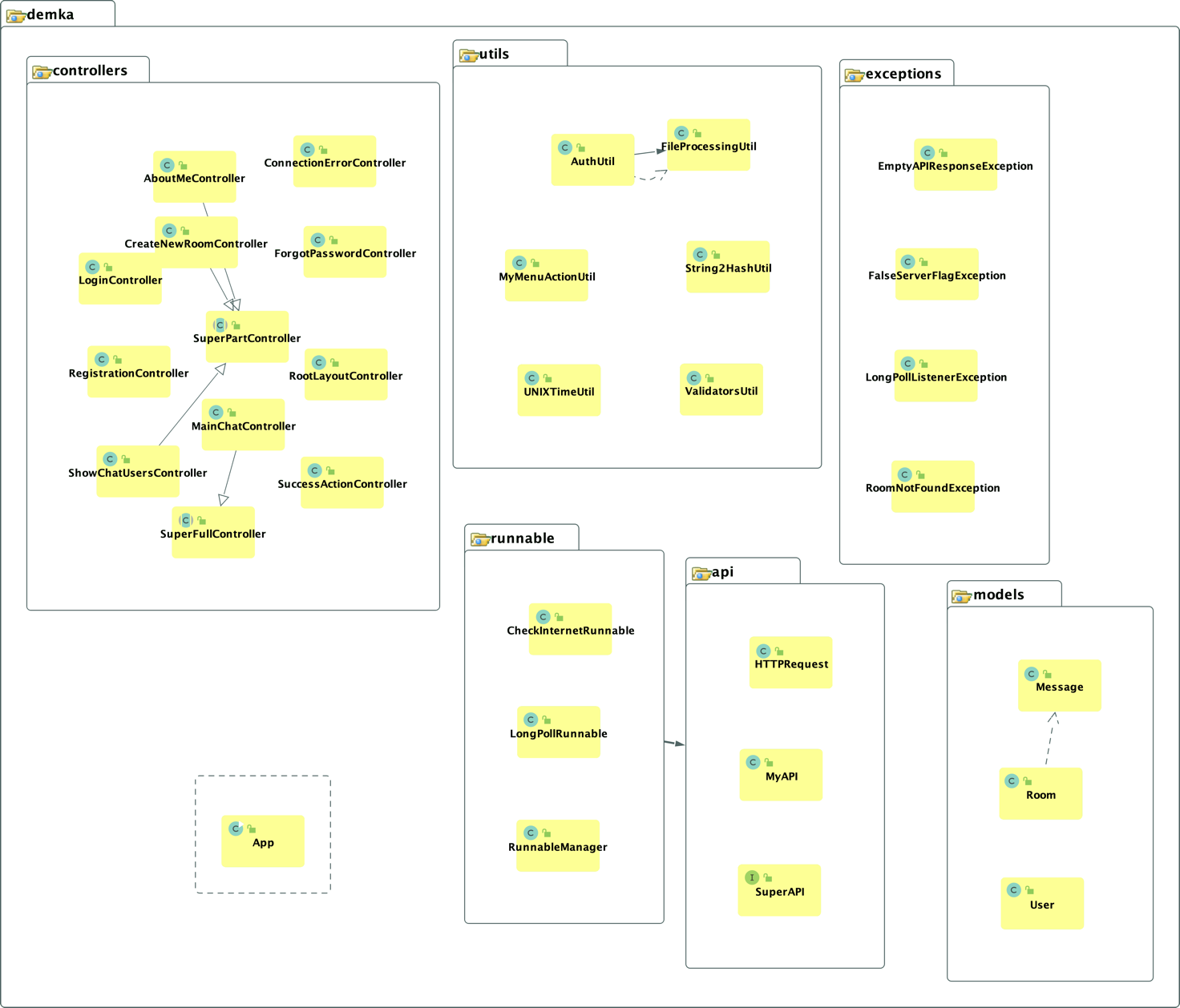
* Классы работы с API (3 шт.) – содержат логику для общения с сервером;
* Классы контроллеры (12 шт.) – отвечают за обработку действий пользователя на формах приложения;
* Классы ошибок (4 шт.) – специальные классы, наследованные от Exception. Вызываются при особых исключениях, которые возникли в программе;
* 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 ниже:



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.



# 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