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

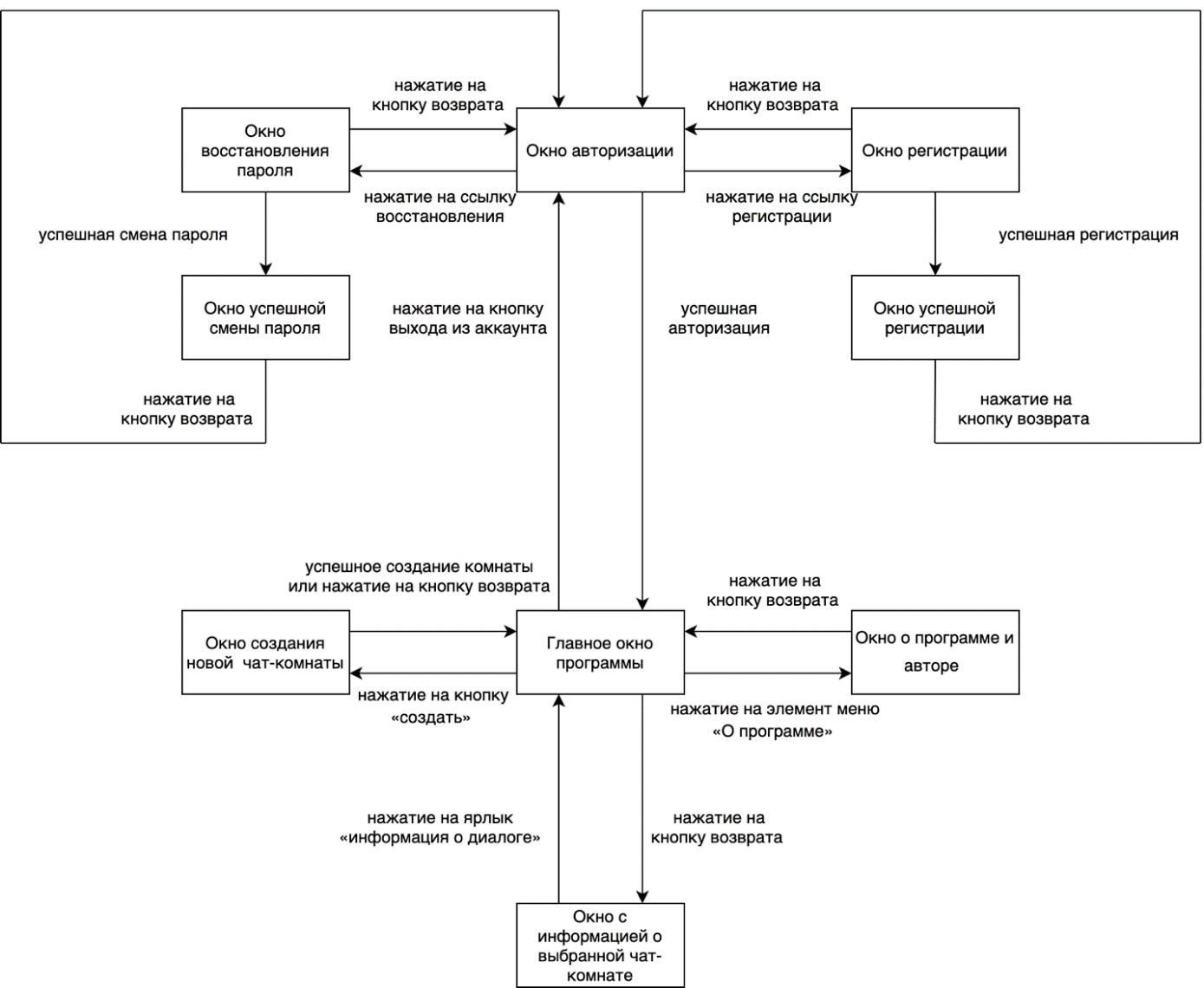
Автоматизация получения сообщений с помощью собственно разработанного решения позволяет оперативно передавать абсолютно любую информацию в текстовом виде, при этом храня все данные исключительно на своих серверах и базах данных, что критично для проектов с повышенным вниманием к конфиденциальности данных,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

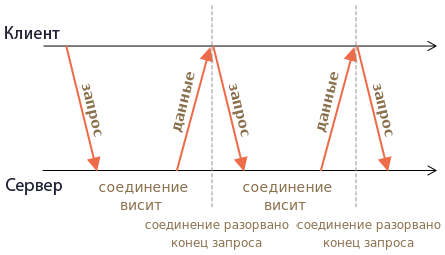
На схеме ниже представлены переходы пользователя между данными формами, кроме отдельного окна «Нет подключения к интернету» ‎. Оно не представлено на схеме т.к. переход на него может быть осуществлён абсолютно с любого окна программы и не зависит от действий пользователя. Данный элемент программы вызывается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

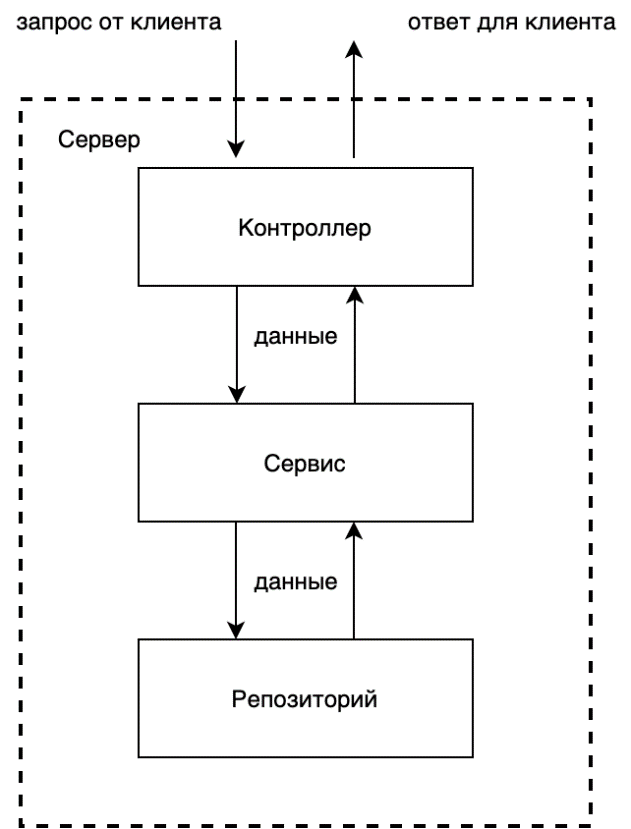


Если клиенту пришло событие о новом сообщении для пользователя, то он отобразит его в соответствующей комнате. Если такой комнаты нет, то добавит её в список чат-комнат пользователя. Если сервер сообщает клиенту, что время ожидания запроса истекло, то клиент обратится заново к серверу с просьбой создания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

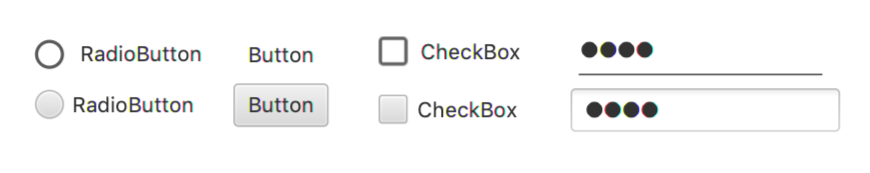


Если же все переданные данные корректны, то из соответствующего контроллера вызывается сервис с логикой, которая отвечает за обработку полученных данных и передачу их в методы репозитория (класса, который наследуется от интерфейса MongoRepository). После получения данных от репозитория, сервис обрабатывает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





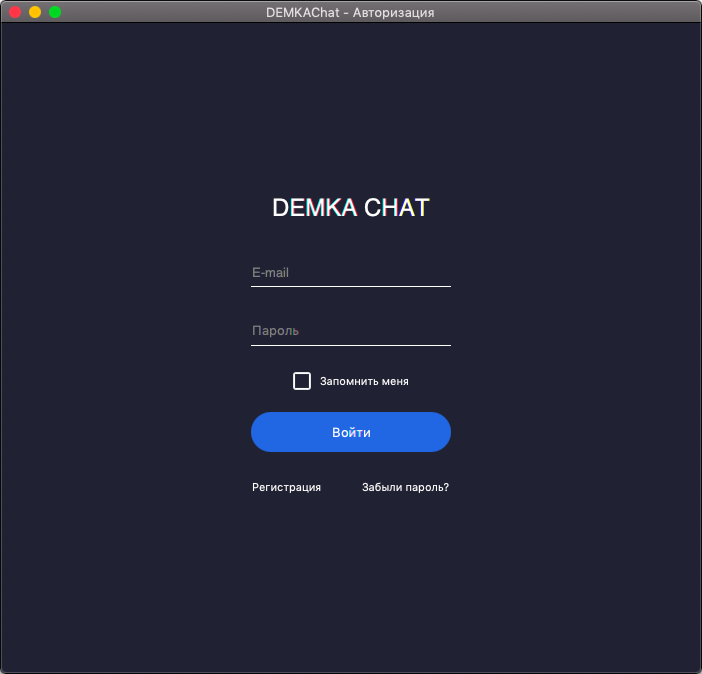
Как видим из рисунка выше, элементы JFoenix выглядят более аккуратно по сравнению со стандартным набором, что особенно хорошо видно на элементах Slider, ProgressBar и TextField.

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

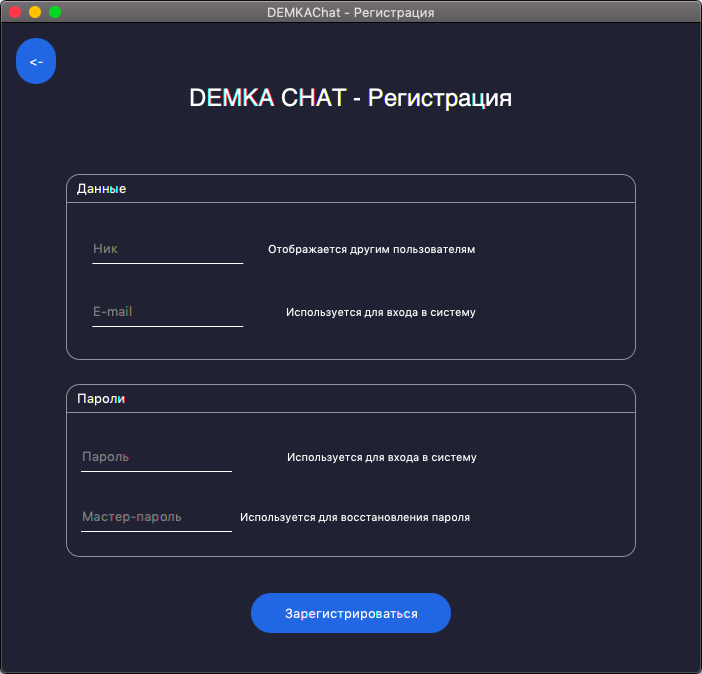
Запрашивает логин/пароль пользователя и предлагает запомнить информацию об авторизации для автоматического входа при следующем запуске программы. Если же пользователь еще не зарегистрирован в системе, то по нажатию на ссылку регистрации будет осуществлён переход на форму регистрации. Если пользователь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

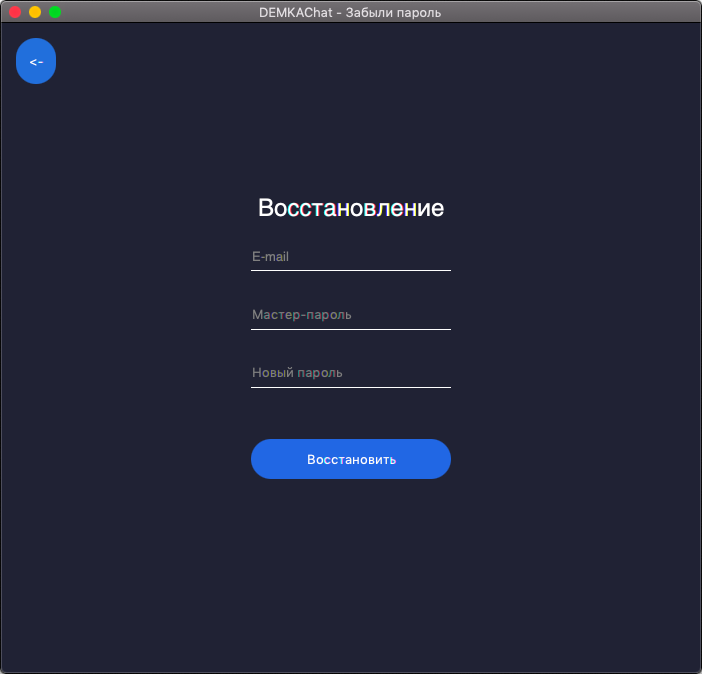
* Корректность ввода e-mail;
* Проверка на пустые поля;
* Проверка на длину пароля и мастер-пароля (не менее 8 символов);
* Проверка, чтоб пароль и мастер-пароль не совпадали между собой.

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.

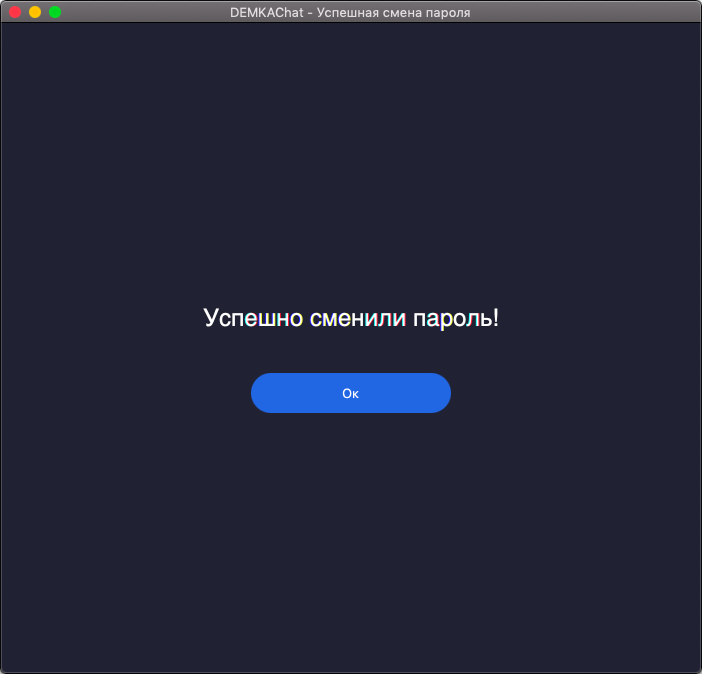
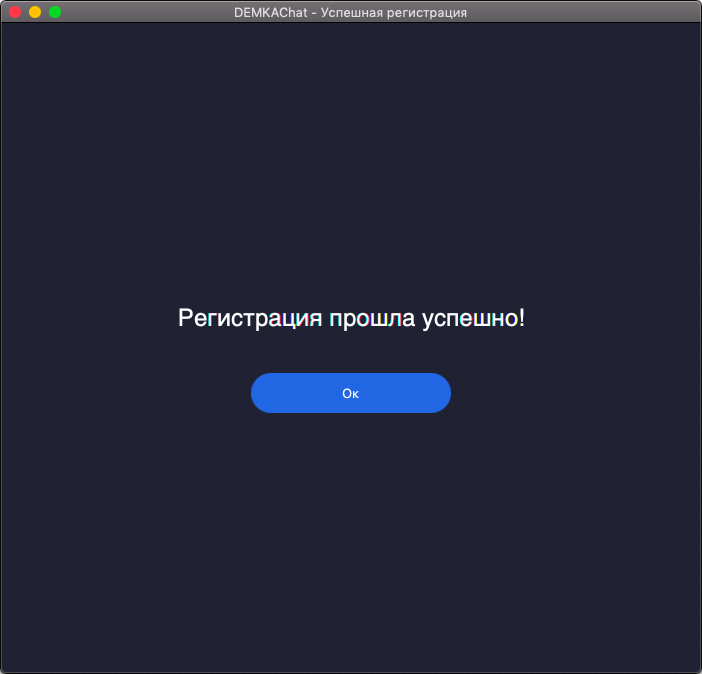
## Окно восстановления пароля



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.

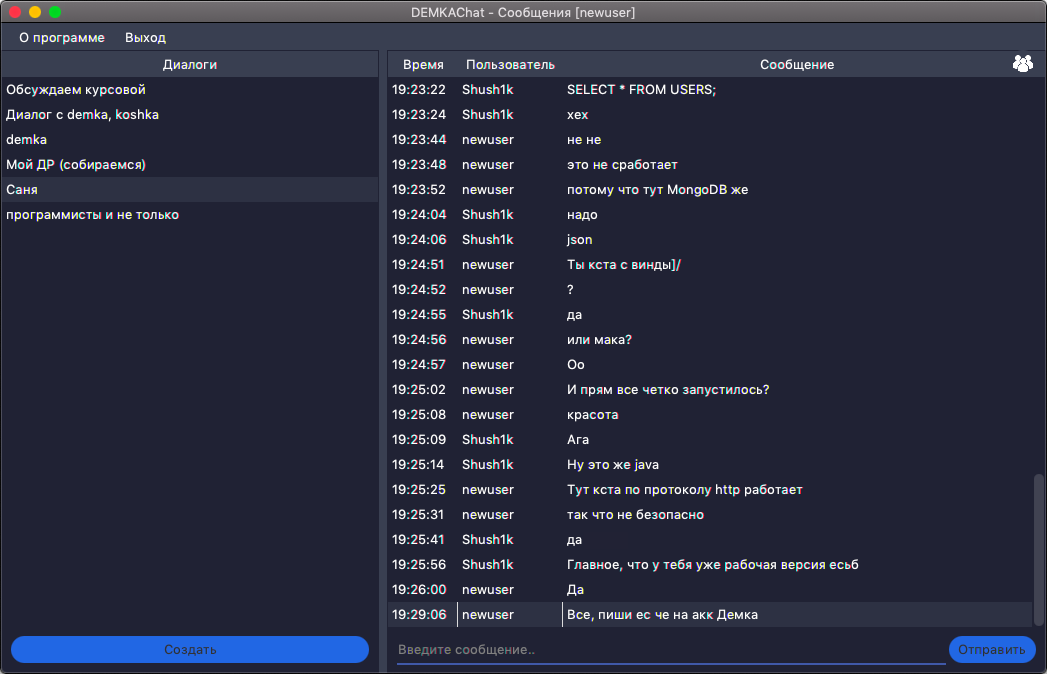
## Окно успешного действия пользователя

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

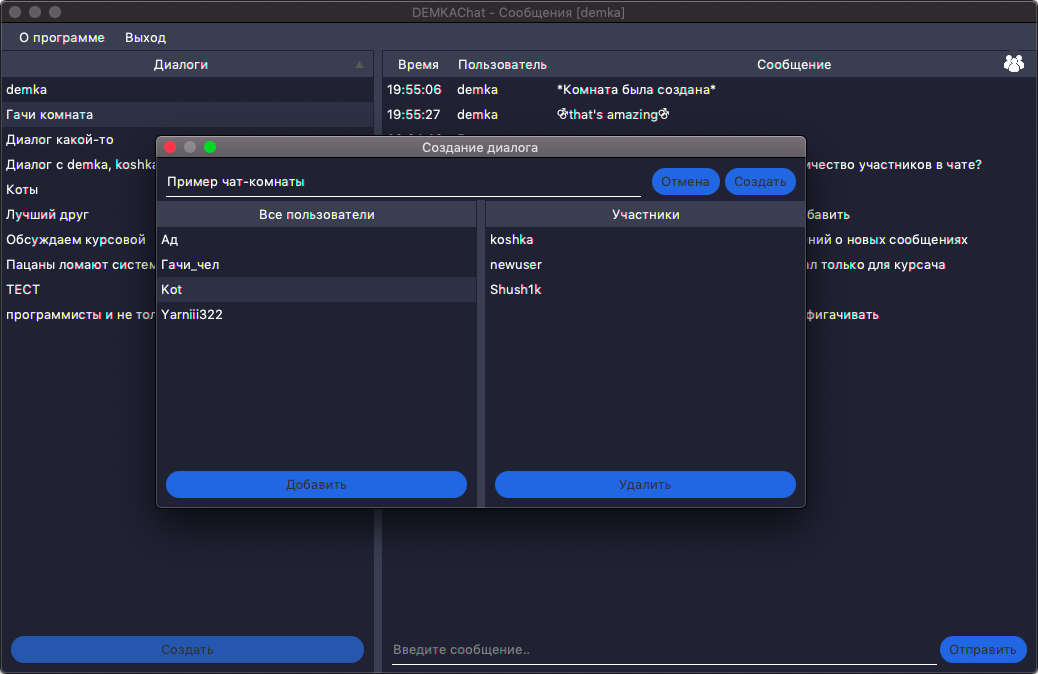
Слева окна расположены все чат-комнаты пользователя. При нажатии на какую-либо из них справа отобразятся сообщения, которые содержатся в данной комнате. Все новые сообщения и комнаты добавляются автоматически без каких-либо действий со стороны пользователя. Внизу находятся кнопки создания нового чата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

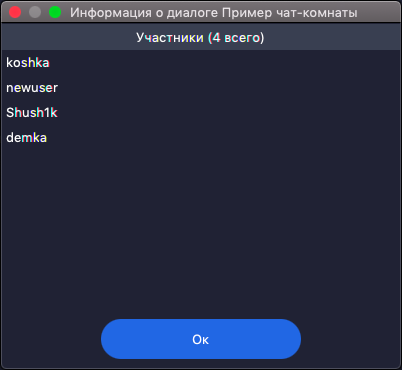
## Окно создания новой чат-комнаты

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.



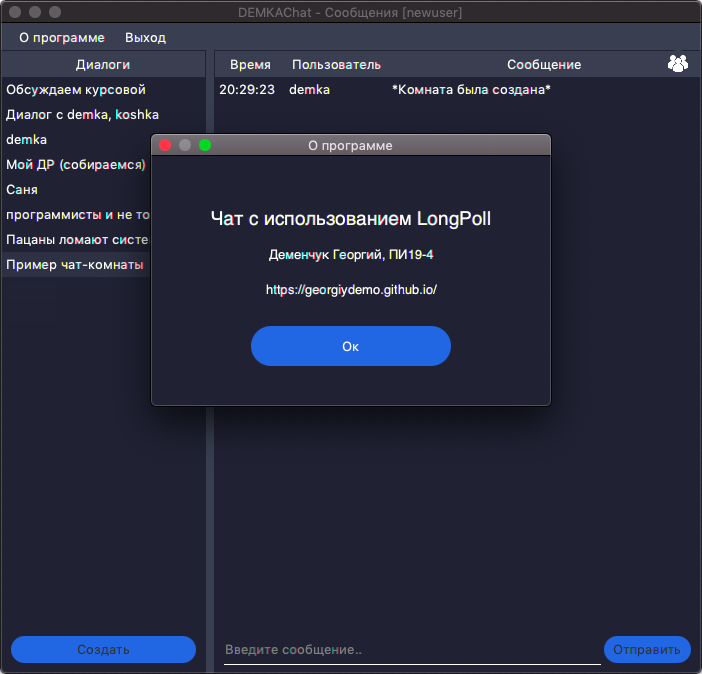
## Окно с информацией о текущей чат-комнате

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.



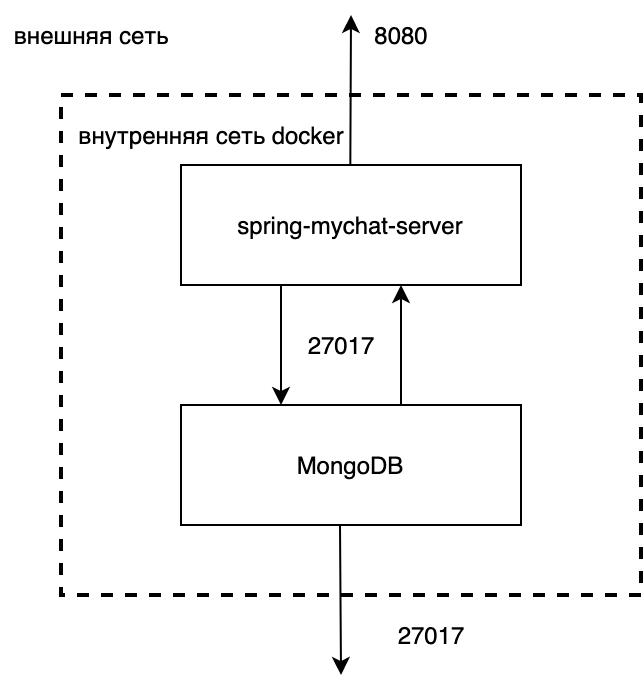
# СОСТАВ ПРИЛОЖЕНИЯ

## 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).

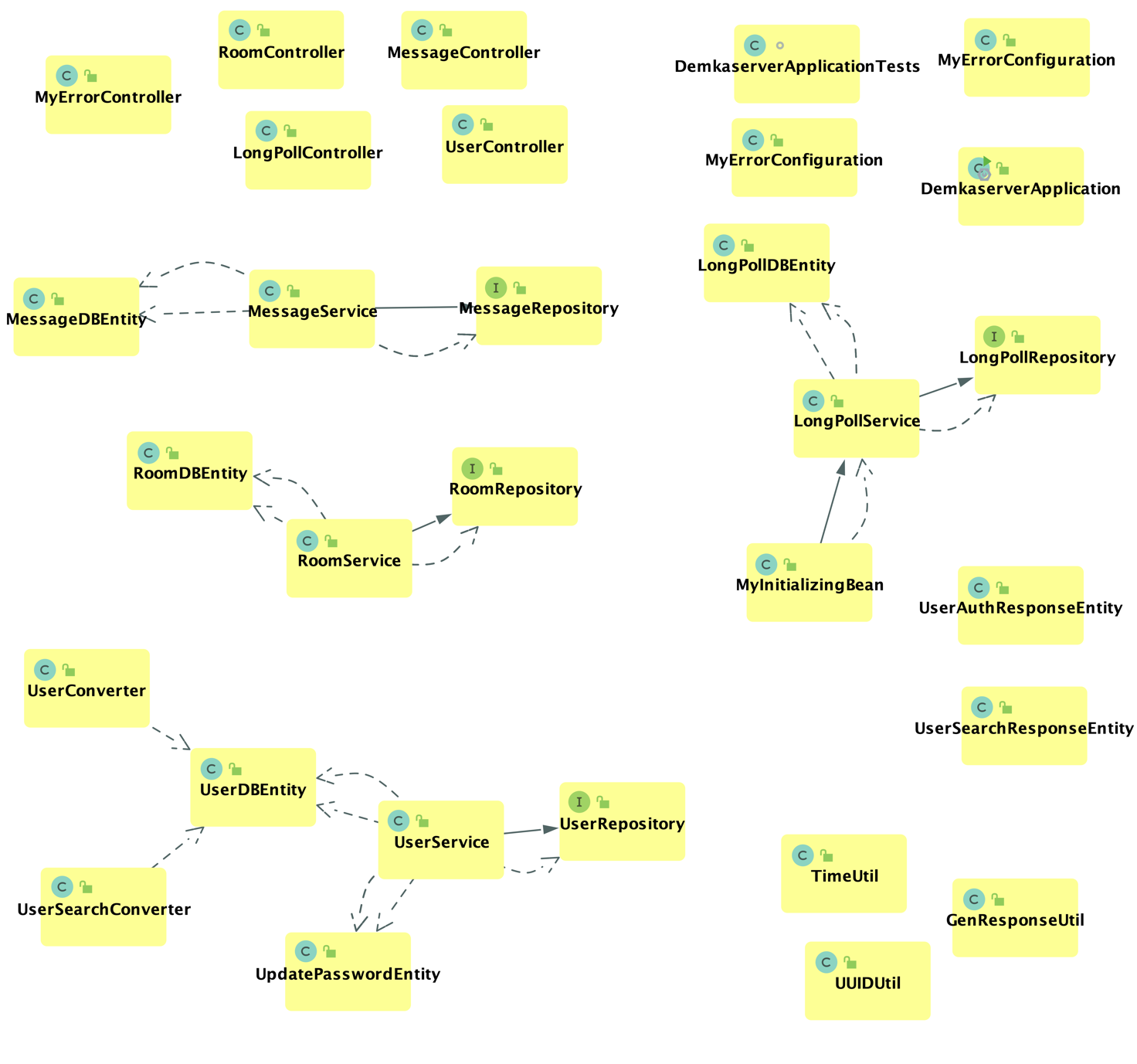
# НАЗНАЧЕНИЕ И СОСТАВ КЛАССОВ ПРОГРАММЫ

## Сервер

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

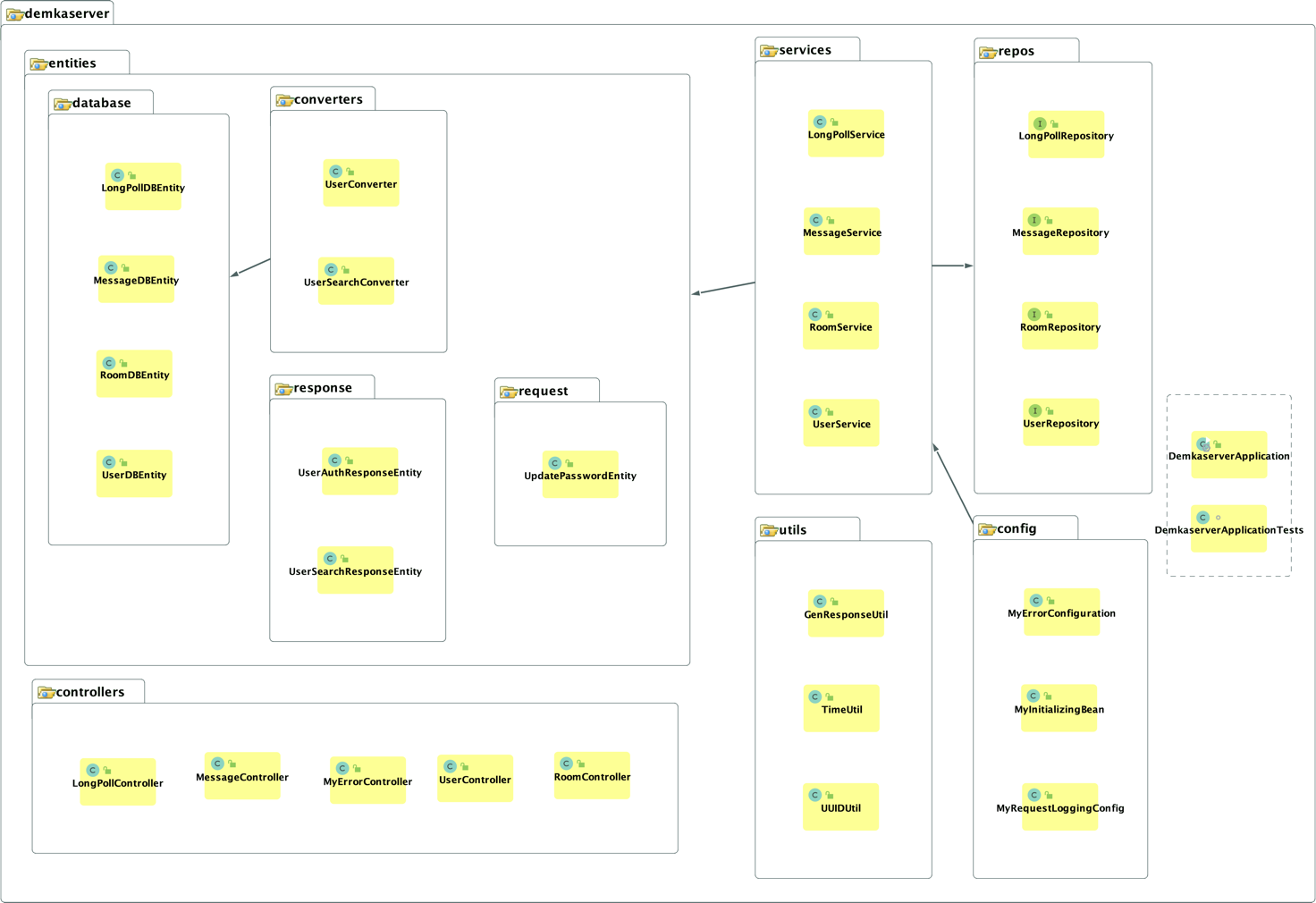
* Контроллеры (5 шт.) – классы для обработки входящих запросов от клиентов;
* Классы конфигураций и тестов (5 шт.) – необходимы для переопределения стандартной конфигурации Spring Boot в целях расширения существующего функционала;
* Классы-утилиты (3 шт.) – необходимы для обособленных решений простых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

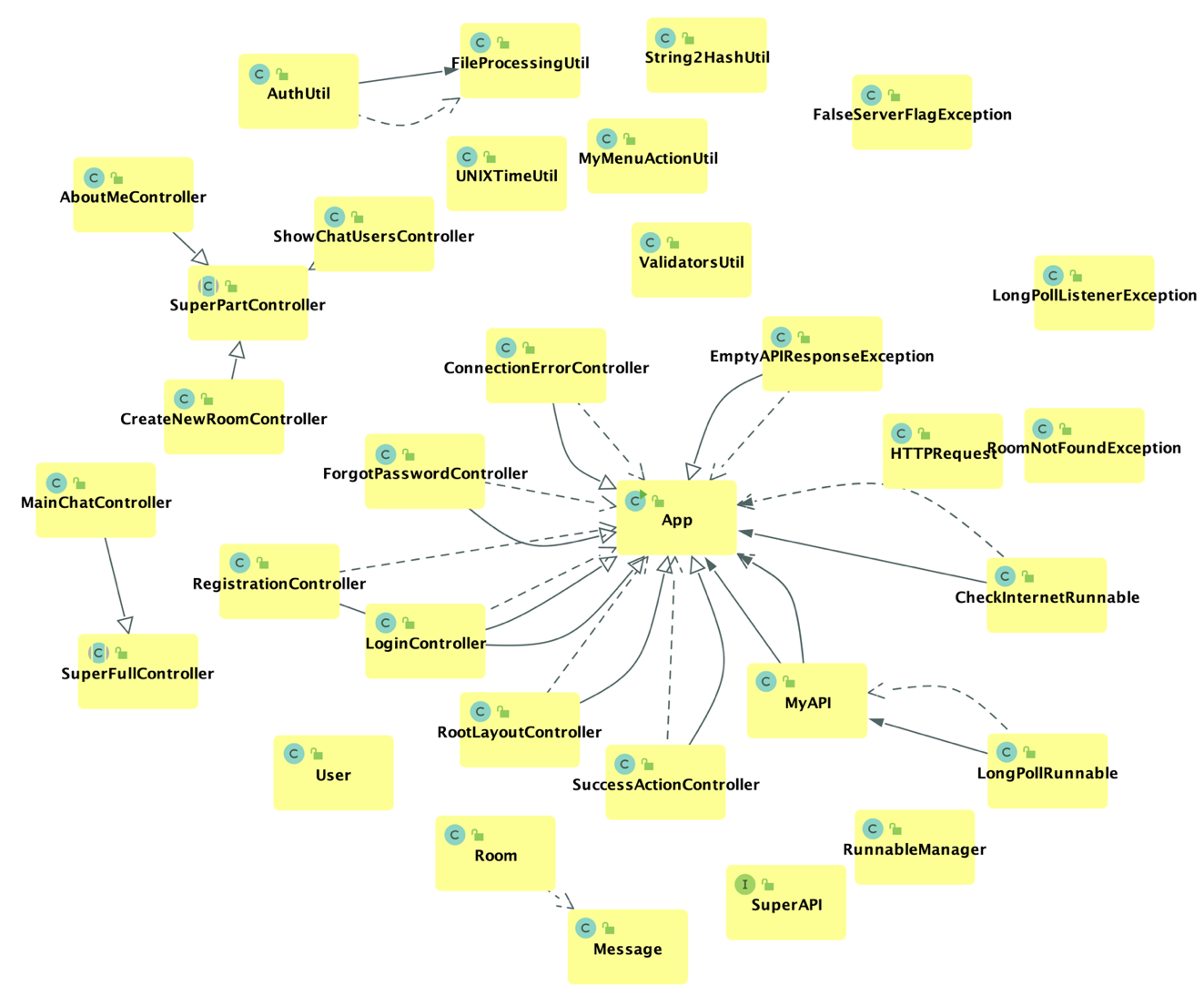


## Клиент

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

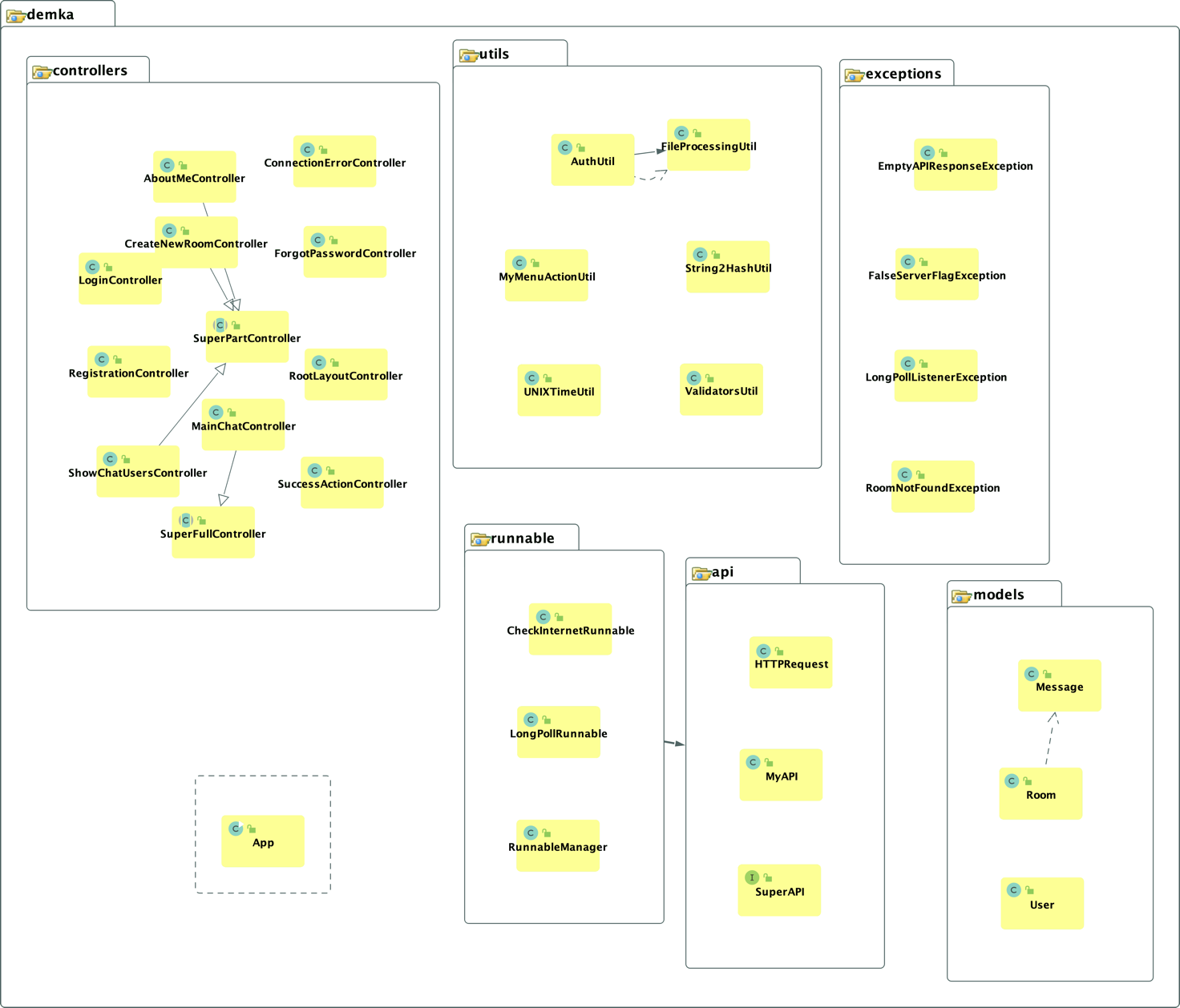
* Классы работы с API (3 шт.) – содержат логику для общения с сервером;
* Классы контроллеры (12 шт.) – отвечают за обработку действий пользователя на формах приложения;
* Классы ошибок (4 шт.) – специальные классы, наследованные от Exception. Вызываются при особых исключениях, которые возникли в программе;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

Исходный код программы тут