МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	Вычислительная техника

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Профиль Вычислительные машины, комплексы, системы и сети

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему

Микросервис «Адресная книга» для интернет-мессенджера

Студент 18.06.18 Пучкова Ульяна Николаевна
(подпись, дата)

Руководитель Пащенко Д.В.
(подпись, дата)

Нормоконтролёр 18.06.18 Кучин А.В.
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от 25.05.2018 г. № 38-09-12)

Заведующий кафедрой Пащенко Д.В.
(подпись)

Работа защищена с отметкой отл. (протокол заседания ГЭК от 20.06.18 № 12)

Секретарь ГЭК Гурин Е.И.
(подпись)

Пенза, 2018

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	Вычислительная техника

«Утверждаю»
Заведующий кафедрой
_____Д.В. Пащенко
«31» 03 2018г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Пучкова Ульяна Николаевна гр. 14ВВ1 факультета вычислительной техники.

2.Тема работы: «Микросервис «Адресная книга» для интернет-мессенджера».

Тема утверждена приказом ПГУ № 398а/о от "3" мая 2018г.

3. Руководитель работы д.т.н., профессор каф. ВТ ПГУ Пащенко Д.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.).

Разработать структуру базы данных для СУБД PostgreSQL. Разработать и отладить прикладной интерфейс (RESTAPI) для взаимодействия с остальными микросервисами приложения. Разработать и отладить микросервис «Адресная книга». Система должна взаимодействовать с базой данных и обрабатывать запросы на создание, обновление и удаление пользователей, организаций и групп.

5. Перечень подлежащих разработке вопросов.

Требуется произвести анализ предметных областей: изучить типы СУБД, виды архитектуры для веб-приложений и рассмотреть основные языки для веб-разработки. Необходимо изучить методологии проектирования веб-приложений, а также изучить функционал фреймворка Spring.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	10	14.03.2017 - 21.03.2017	
2.	Анализ требований	10	22.03.2017 - 28.03.2017	
3.	Выбор технологий, применяемых при проектировании и разработке	10	29.03.2017 - 03.04.2017	
4.	Разработка БД	15	04.04.2017 - 12.04.2017	
5.	Разработка REST API	20	13.04.2017 - 22.04.2017	
6.	Разработка приложения «Адресная книга»	25	23.04.2017 - 27.05.2017	
7.	Оформление пояснительной записки	10	28.05.2017 - 08.06.2017	

Дата выдачи задания "3" марта 2018 г.

Руководитель бакалаврской работы _____ Пащенко Д.В.
(подпись, дата)

Задание к исполнению принял студент 03.03.2018 Пучкова У.Н.
(подпись, дата)

Работу к защите допустить
Декан факультета 18.6.18 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 92 листа, 13 рисунков, 14 таблиц, 12 источников, 4 приложения.

МИКРОСЕРВИС, БД, СУБД, REST, API, JAVA, МЕССЕНДЖЕР.

Цель работы – разработать приложение «Адресная книга».

Объектом разработки является приложение, обеспечивающее выполнение следующих функций: создание, удаление, редактирование пользователей, организаций, групп. Хранение информации о пользователях, организациях и группах должно осуществляться в БД. Также необходимо реализовать *API* для взаимодействия с остальными микросервисами.

В результате проведенной работы было разработано приложение, удовлетворяющее поставленной задаче, а именно: разработана структура БД, создан *REST API*, реализующий необходимый функционал. Приложение было отлажено и успешно интегрировано в микросервисную структуру мессенджера.

Перечень принятых сокращений

API (Application Programming Interface) – программный интерфейс приложения;

HTML (HyperText Markup Language) – язык гипертекстовой разметки;

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста;

ID (Identifier) – идентификатор;

JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript;

NoSQL (not only SQL) – не только SQL;

OLAP (Online Analytical Processing) – интерактивная аналитическая обработка;

REST (Representational State Transfer) – передача состояния представления;

SQL (Structured Query Language) – язык структурированных запросов;

UML (Unified Modeling Language) – унифицированный язык моделирования;

URI (Uniform Resource Identifier) – унифицированный (единообразный) идентификатор ресурса;

XML (eXtensible Markup Language) – расширяемый язык разметки;

БД – База данных;

ОРСУБД – объектно-реляционная система управления базами данных;

РСУБД – система управления реляционными базами данных;

СУБД – система управления базами данных.

Содержание

Перечень принятых сокращений 5
Введение 7
1 Анализ предметной области 8
1.1 Анализ типов СУБД 8
1.2 Выбор реляционной СУБД 10
1.3 Выбор архитектуры приложения 13
1.4 Выбор языка программирования 17
1.5 Описание архитектуры REST API 19
2 Реализация 23
2.1 Структура БД 24
2.2 Описание REST API 29
2.3 Описание классов 31
2.4 Программная реализация 36
3 Описание результата 38
4 Заключение 42
Список используемых источников 43
Приложение A. Листинги программы на Java 45
Приложение B. Листинг SQL-файла 70
Приложение C. Описание REST API для работы с пользователями 81
Приложение D. Презентация 87

Введение

В настоящее время общение посредством мессенджеров является неотъемлемой частью жизни современного человека. Различные фирмы используют свои корпоративные внутренние средства для общения, чтобы сделать общение своих сотрудников безопаснее. Чтобы обеспечить бесперебойную и стабильную работу мессенджеров, многие разработчики используют различные подходы к организации архитектуры приложения.

Одной из самых распространенных и удобных архитектур является микросервисная архитектура. Удобство заключается в том, что приложение разбито на функциональные блоки – микросервисы, каждый из которых может масштабироваться, тем самым распределяя нагрузку, повышая отказоустойчивость всего приложения.

В данной работе рассматривается реализация одного из микросервисов мессенджера – микросервис «Адресная книга». Функциональное назначение данного сервиса – хранение информации о пользователях и организациях, с привязкой к организационной структуре предприятия. В контексте данной работы не рассматриваются остальные микросервисы мессенджера, так как это является коммерческой тайной.

1 Анализ предметной области

При разработке программных продуктов необходимо провести анализ существующих вариантов для решения задачи. В данном разделе рассматриваются общие теоретические вопросы, освещающие технологические аспекты, необходимые для реализации программного продукта корпоративного класса. В частности производится выбор СУБД, архитектуры приложения, языка программирования и архитектуры *API*. В каждом подразделе обосновывается выбор того или иного технического решения задачи.

1.1 Анализ типов СУБД

Существуют два вида СУБД: реляционные (*SQL*) и нереляционные (*NoSQL*) [1]. Чтобы выбрать тот или иной вид СУБД, нужно знать характерные отличия их друг от друга.

Можно выделить следующие отличительные признаки:

1. Разница в структуре и типе хранимых данных. В реляционных СУБД должна быть однозначно определена структура хранения данных, в то время как в *NoSQL* БД нет таких строгих ограничений, и они допускают хранение любого типа данных. Для микросервиса «Адресная книга» подходит структура данных с однозначно определенными связями между объектами таблиц.
2. Различия в структуре запросов. В *SQL* БД данные могут быть получены только посредством языка *SQL*, так как реляционные СУБД соответствуют стандартам *SQL*. В нереляционных СУБД нет четкого определения, как должен выглядеть запрос и поэтому для каждой *NoSQL* БД реализуется свой способ работы с данными. Для микросервиса «Адресная книга» требуется построение множества запросов с выборками по условию, и

использование нереляционной СУБД будет сопряжено с проблемами с созданием запросов, поэтому выбор реляционной СУБД для работы здесь очевиден.

3. Масштабируемость. Оба вида СУБД могут хорошо масштабироваться по вертикали, однако по горизонтали лучше растягиваются именно нереляционные СУБД. Для микросервиса «Адресная книга» больше подходит вариант с лучшим горизонтальным масштабированием.
4. Надежность. *SQL* базы данных являются более надежными, по сравнению с *NoSQL* БД. Поэтому реляционные БД более предпочтительны для выбора в качестве решения для микросервиса.
5. Поддержка. Реляционные СУБД появились гораздо раньше *NoSQL*, поэтому существуют уже готовые платные и бесплатные решения и поддержка, и найти ответ на свой вопрос гораздо проще, чем когда речь идет о сложном решении в нереляционной СУБД. И снова выбор в пользу *SQL*.
6. Хранение и доступ к сложным структурам данных. *SQL* базы данных выигрывают в производительности по сравнению с нереляционными СУБД, так как изначально предполагалось, что они будут работать именно со сложными структурами данных. Для микросервиса «Адресная книга» снова больше подходит *SQL* БД.

Подводя итоги анализа возможностей реляционных и нереляционных СУБД, для микросервиса «Адресная книга» очевиден выбор в пользу реляционной СУБД.

1.2 Выбор реляционной СУБД

Для выбора наиболее подходящей РСУБД необходимо проанализировать имеющиеся на рынке популярные и развивающиеся *SQL* базы данных. Таковыми являются: *MySQL*, *PostgreSQL*, *MS SQL Server*, *Oracle Database*.

1.2.1 MS SQL Server

Microsoft SQL Server – разработанная корпорацией майкрософт, система управления реляционными базами данных [2]. Основным языком для написания запросов является *Transact-SQL*. Данная РСУБД подходит как для создания персональных баз данных, так и для реализации БД масштаба компании. Последняя версия продукта – *SQL* Server 2016. В данной версии появилась возможность отслеживать, как меняются данные с течением времени (*temporal data support* – временная поддержка данных). А также эта версия поддерживает динамическую маскировку данных (*dynamic data masking*), которая дает гарантию того, что только авторизованные пользователи могут получить доступ к конфиденциальным данным.

Достоинства РСУБД *MS SQL Server*:

- простота использования;
- возможность регулировки и отслеживания уровня производительности, чтобы уменьшить загрузку;
- последняя версия продукта работает стабильно;
- возможность интеграции с другими продуктами *Microsoft*;
- возможность визуализации на мобильных устройствах.

Недостатки РСУБД *MS SQL Server*:

- очень высокая стоимость продукта для юридических лиц;
- возможны проблемы в работе служб интеграции импорта файлов;
- высокая ресурсоемкость *SQL* Server (даже при настройке производительности).

При всех упомянутых достоинствах, у данной РСУБД есть очень существенный недостаток, который делает ее недоступной для использования – ее высокая цена для юридических лиц. Бесплатная версия не обладает достаточными возможностями для полной реализации базы данных для микросервиса.

1.2.2 Oracle Database

Oracle Database – объектно-реляционная система управления базами данных, разработанная компанией *Oracle* [3]. История данной ОРСУБД берет начало с конца 70-х годов прошлого века. В настоящее время доступна последняя версия продукта, которая вышла в 2013 году – 12c (суффикс «c» обозначает английское слово «*cloud*» – облако). Эта версия поддерживает подключение баз данных через облачные сервисы, что позволяет размещать базу данных на нескольких серверах и хранить огромный объем данных. Возможность использования подключаемых БД обеспечивает мультиарендность (обслуживание множества организаций-клиентов – «арендаторов») и живую миграцию баз данных. Улучшена безопасность проведения операций – каждая транзакция проводится изолированно от остальных.

Достоинства ОРСУБД *Oracle Database*:

- надежность;
- современный функционал и новые разработки, доступные в последней версии;

Недостатки ОРСУБД *Oracle Database*:

- высокая стоимость продукта для юридических лиц;
- для системы необходимы аппаратные ресурсы, обладающие достаточной мощностью и производительностью;

К сожалению, высокая стоимость не позволяет использовать данную ОРСУБД, бесплатная версия позволяет развертывать небольшие по объему базы данных.

1.2.3 MySQL

MySQL – свободная реляционная система управления базами данных [4]. С 2010 года права на торговую марку получила компания *Oracle*. *MySQL* является бесплатным программным обеспечением, регулярно выходят новые версии, дополняющие функционал. Существуют и платные версии для коммерческих организаций. В бесплатной версии сделан упор на надежность и скорость работы, но не на полноту функционала. *MySQL* является довольно гибкой СУБД за счёт возможности использовать множество типов таблиц.

Достоинства *MySQL*:

- бесплатная для личного использования;
- имеет хорошую документацию;
- безопасность;
- возможность использования множества функций в бесплатной версии;
- возможность взаимодействия с другими базами данных, например *DB2* и *Oracle*.

Недостатки *MySQL*:

- проблемы с надежностью;
- платная для коммерческого использования
- некоторые ограничения функционала;
- нет встроенной поддержки *XML* и *OLAP*;
- для бесплатной версии доступна только платная поддержка.

Несмотря на кажущиеся преимущества, система в основном используется для мелких и средних проектов и не подходит для крупных, а также не бесплатная.

1.2.4 PostgreSQL

PostgreSQL – свободная объектно-реляционная система управления базами данных с открытым исходным кодом, разработчиком которой является сообщество *PostgreSQL* [5]. Базируется на языке *SQL*. Последняя доступная версия программы – 10.4. *PostgreSQL* активно разрабатывается сообществом, в него постоянно добавляются новые функции. Последнее время все больше используется в крупных коммерческих проектах.

Достоинства *PostgreSQL*:

- бесплатная и для личного и для коммерческого использования;
- хорошо масштабируется;
- возможность обрабатывать терабайты данных;
- поддержка формата *JSON*;
- существует множество готовых, встроенных функций;
- хорошая документация;
- большое сообщество с поддержкой.

Недостатки *PostgreSQL*:

- неэффективная репликация данных;
- трудности с обновлением.

PostgreSQL выигрывает по всем параметрам и самое главное – программа бесплатно распространяется. Поэтому было принято решение использовать именно эту ОРСУБД.

В проекте используется версия 9.6, на момент начала разработки она была выбрана как наиболее стабильная.

1.3 Выбор архитектуры приложения

Существует два основных типа архитектуры для проектирования веб-приложения: монолитная и микросервисная.

Чтобы понять, какую архитектуру выбрать, нужно проанализировать особенности каждой.

1.3.1 Монолитная архитектура приложения

Монолитная архитектура подразумевает такой подход к разработке приложения, когда приложение проектируется как один большой связанный модуль [6]. Компоненты этого модуля взаимосвязаны между собой, имеют общую память и ресурсы. На рисунке 1 показано графическое представление монолитной архитектуры.

Рисунок 1 — графическое представление монолитной архитектуры.

Преимущества монолитной архитектуры:

- сравнительно простая реализация, развертывание и управление;
- проще поддерживать согласованность кода и обрабатывать ошибки;
- легко добавить типичный функционал к компонентам;
- достаточно высокая производительность для малых и средних приложений.

Но при кажущихся преимуществах монолитной архитектуре присущи достаточно серьезные недостатки:

- возрастающая сложность и запутанность приложения;

- с увеличением приложения все сложнее поддерживать изолированность сервисов, чтобы масштабировать каждый из них независимо или осуществлять поддержку кода;
- внесение изменений сопряжено с проблемами и занимает много времени;
- сбой одного сервиса приводит к сбою всего приложения.

Последний недостаток, при котором от сбоя в одном сервисе начинает сбоить всё приложение, является неприемлемым, если рассматривать его в контексте работы интернет-мессенджера.

1.3.2 Микросервисная архитектура приложения

Микросервисная архитектура является современным представлением сервис-ориентированной архитектуры и используется для проектирования распределенных систем [7]. То есть приложение с микросервисной архитектурой состоит из небольших независимых компонентов (микросервисов), каждый из которых может независимо развиваться и масштабироваться, и взаимодействовать со своими собственными ресурсами. Отдельные микросервисы могут располагаться на разных серверных узлах. Схематичное сравнение микросервисной и монолитной архитектуры представлено на рисунке 2.

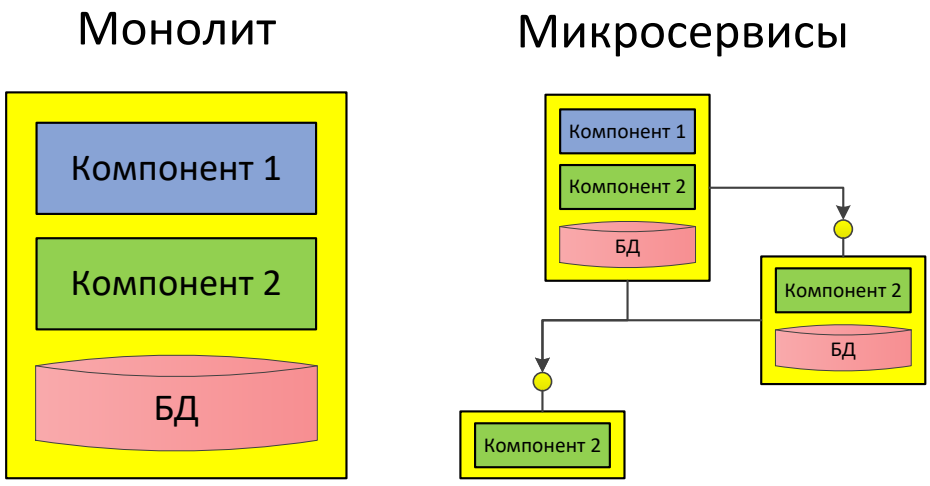


Рисунок 2 — Сравнение монолитной и микросервисной архитектур

Преимущества использования микросервисов:

- возможность обновления приложения по частям, независимое развертывание;
- при сбое одного микросервиса, приложение не сбоит целиком;
- возможность независимого масштабирования отдельных микросервисов;
- микросервисная архитектура позволяет размещать разные модули на разных серверах, тем самым обеспечивая их логическое разделение;
- возможность использования нескольких языков программирования за счет расположения модулей на разных серверах;
- разнообразие технологий, которые можно использовать, чтобы подобрать наиболее подходящий функционал;
- децентрализованное управление;
- децентрализованное управление данными;

Как и у всякой технологии и подхода к разработке у микросервисов есть и ряд недостатков:

- сложность разработки;
- требуют тщательного управления, так как используют *API* и развертываются на разных серверных узлах;
- требуют четкого разграничения модулей;
- увеличение сетевых задержек

Вывод: несмотря на сложность разработки и управления, микросервисная архитектура отлично подходит как для реализации функционала «Адресной книги», так и для проектирования мессенджера в целом.

1.4 Выбор языка программирования

При реализации любого программного продукта одним из самых важных этапов является выбор языка программирования. От этого зависит продуктивность и чистота кода, а также длительность периода создания минимально жизнеспособного продукта.

В настоящее время наиболее популярными языками программирования для разработки веб-приложений являются: *Go*, *Java*, *JavaScript*, *Python*. Такие языки как *C++, Php* и *Ruby* в последнее время теряют популярность в веб-разработке.

1.4.1 Язык программирования Go

Go (также известный как *Golang*) — компилируемый многопоточный язык программирования, разработанный компанией *Google* [8]. Язык разрабатывался для проектирования высокопроизводительных программ и для решения проблем большинства языков. Язык проектировался с целью нивелировать такие проблемы при разработке как медленная сборка программы, сложность при разработке инструментария, проблемы межъязыкового взаимодействия и другие.

Но данный язык не является объекто-ориентированным в полном смысле этого понятия, так как в язык сознательно не была включена концепция наследования. Язык обладает меньшей переносимостью на разные платформы, так как компилируется в машинный код.

1.4.2 Язык программирования Python

Python – высокоуровневый язык программирования, который ориентирован на повышение производительности разработчика и читаемости кода [9]. Этот язык очень гибкий и обладает широким функционалом, а код высокой степенью читаемости. Но у данного языка есть существенный недостаток – он обладает низким быстродействием и поэтому часто используется как язык второго плана.

1.4.3 Язык программирования JavaScript

JavaScript – мультипарадигменный язык программирования [4][10]. Язык не принадлежит какой-либо организации или компании. Наиболее часто используется в качестве языка сценариев веб-страниц. *JavaScript* поддерживает императивный, декларативный и объектно-ориентированный стили программирования. *JavaScript* часто используется в таких программных продуктах, которые в свою очередь используются в веб-разработке. Так как язык является интерпретируемым, то производительность и безопасность зависит от конкретной реализации интерпретатора («движка»).

1.4.4 Язык программирования Java

Java – сильно типизированный объектно-ориентированный язык программирования [11]. В настоящее время разработкой занимается компания *Oracle*. Основной особенностью данного языка является то, что программы на *Java* транслируются в байт-код, который выполняется на виртуальной машине *Java*. Виртуальная машина *Java* в свою очередь является интерпретатором и передает инструкции оборудованию, что обеспечивает легкую переносимость программы на разные платформы. В настоящее время *Java* является одним из самых популярных языков программирования.

1.4.5 Выбор языка программирования

В качестве языка программирования был выбран язык *Java*. В отличие от программ написанных на *Python* программы на *Java* обладают большей производительностью, по сравнению с языком *Go Java* давно используется программистами, имеет большое сообщество, хорошую документацию, множество популярных фреймворков. *JavaScript* проигрывает *Java* в безопасности. Для упрощения и ускорения процесса разработки был выбран

фреймворк Spring. Данный фрейворк является универсальным и используется для создания бизнес-приложений.

1.5 Описание архитектуры REST API

REST – это архитектурный стиль взаимодействия компонентов распределенного веб-приложения. Принципы *REST* сформулировал Рой Филдинг в своей диссертации в 2000 году, но не существует официально принятого стандарта или спецификации *REST* (в отличие, например, от протокола *SOAP*) [12]. *API*-интерфейс может считаться *RESTful* только в том случае, если соблюдены все требования. При создании микросервиса «Адресная книга» (как и при создании всего интернет-мессенджера) были учтены данные требования, и поэтому веб-приложение является *RESTful*.

Существует шесть обязательных ограничений для проектирования *REST*-приложений:

1.5.1 Модель клиент-сервер

Система должна быть поделена на клиентов и серверов, то есть в основе данного ограничения лежит принцип разграничения потребностей. Это позволяет упростить серверную часть приложения, что в будущем упрощает ее масштабируемость. При этом у клиентского интерфейса повышается переносимость кода на другие платформы. Микросервис «Адресная книга» выступает в качестве сервера, но клиенты общаются с ним через сервер мессенджера. Клиентами являются приложения на мобильных устройствах.

1.5.2 Отсутствие состояния

Взаимодействие между клиентом и сервером строится по следующему условию: в моменты времени между запросами никакая информация о клиенте не должна храниться на сервере, а каждый запрос от клиента должен быть составлен таким образом, чтобы на сервер поступила вся необходимая

информация о транзакции для ее завершения. При этом информация о состоянии сессии сохраняется у клиента. В то время как происходит обработка запросов от клиента, считается, что он находится в переходном состоянии. Данное ограничение позволяет системе лучше масштабироваться, так как отсутствие необходимости хранить информацию о состояниях высвобождает ресурсы сервера, которые могут быть направлены на обработку одновременно большего количества клиентов. Микросервис «Адресная книга» не сохраняет сессии клиентов на своей стороне и не знает в каком состоянии находятся клиенты.

1.5.3 Кэширование

Клиенты могут выполнять кэширование ответов сервера. В этом случае ответы сервера должны быть обозначены как кэшируемые, если поступила новая информация, или некэшируемые, если поступившие данные уже передавались ранее. Грамотно построенное кэширование способно уменьшить взаимодействие между клиентом и сервером, что увеличивает производительность и масштабируемость распределенной веб-системы в целом. В микросервисе «Адресная книга» невозможно сделать кэширование запросов на данном этапе разработки, в дальнейшем планируется рассмотрение этого вопроса.

1.5.4 Единообразие интерфейса

Для эффективного взаимодействия компонентов и кэширования в сети необходим унифицированный интерфейс, позволяющий независимо развиваться отдельным сервисам.

В свою очередь для унифицированных интерфейсов существует четыре условия:

1. Идентификация ресурсов. Любая информация может быть ресурсом – для этого она должна иметь имя. Каждый ресурс должен быть идентифицирован с помощью идентификатора, не

меняющегося при изменении состояния ресурса. Идентификатором в *REST* является *URI*. В микросервисе «Адресная книга» для каждой сущности существует свой идентификатор. Пример для сущности «Пользователь» идентификатор выглядит как «*/addressbook/user/…*»

2. Манипуляция ресурсами через представление. Ресурсы могут быть представлены различными способами, например: *JSON*, *HTML*, *XML* описание и так далее. Представление является описанием текущего состояния ресурса и используется для выполнения операций над ресурсом. Взаимодействия клиентов с ресурсами происходит посредством представлений. В «Адресной книге» в качестве способа представления используется описание в формате *JSON*.
3. «Самоописываемые» сообщения. Каждый запрос (ответ) хранит в себе всю необходимую информацию, чтобы понять, как его нужно обработать. Для обработки одного запроса не должно быть дополнительных сообщений. В «Адресной книге» полностью удовлетворяется данное условие.
4. Гипермедиа, как средство изменения состояния приложения. Для навигации по *API* должен быть использован гипертекст, что позволяет клиентам обнаруживать ресурсы посредством гиперссылок. В микросервисе «Адресная книга» используются гиперссылки.

1.5.5 Многослойная система

В архитектуре *REST* возможно разделение системы на иерархию слоев, но с условием: отдельный компонент системы может видеть только компоненты следующего уровня. Использование промежуточных прокси-серверов позволяет увеличить масштабируемость, сбалансировав нагрузку и

распределив кэширование. Также это позволяет использовать политику безопасности для обеспечения конфиденциальности данных.

В мессенджере клиент не общается напрямую с сервером «Адресной книги», взаимодействие идет с сервером мессенджера. Для распределения нагрузки предполагается возможность увеличения количества серверов «Адресной книги».

1.5.6 Код по требованию

Является необязательным требованием. В архитектуре *REST* данное условие разрешает загружать программы (код) для выполнения на стороне клиента.

В мессенджере запрещено загружать код на сторону клиента из соображений безопасности.

2 Реализация

В данной работе приведены результаты разработки одного из микросервисов интернет-мессенджера – «Адресная книга» (рисунок 3). Функциональное назначение данного микросервиса – хранение и управление аккаунтами пользователей мессенджера.

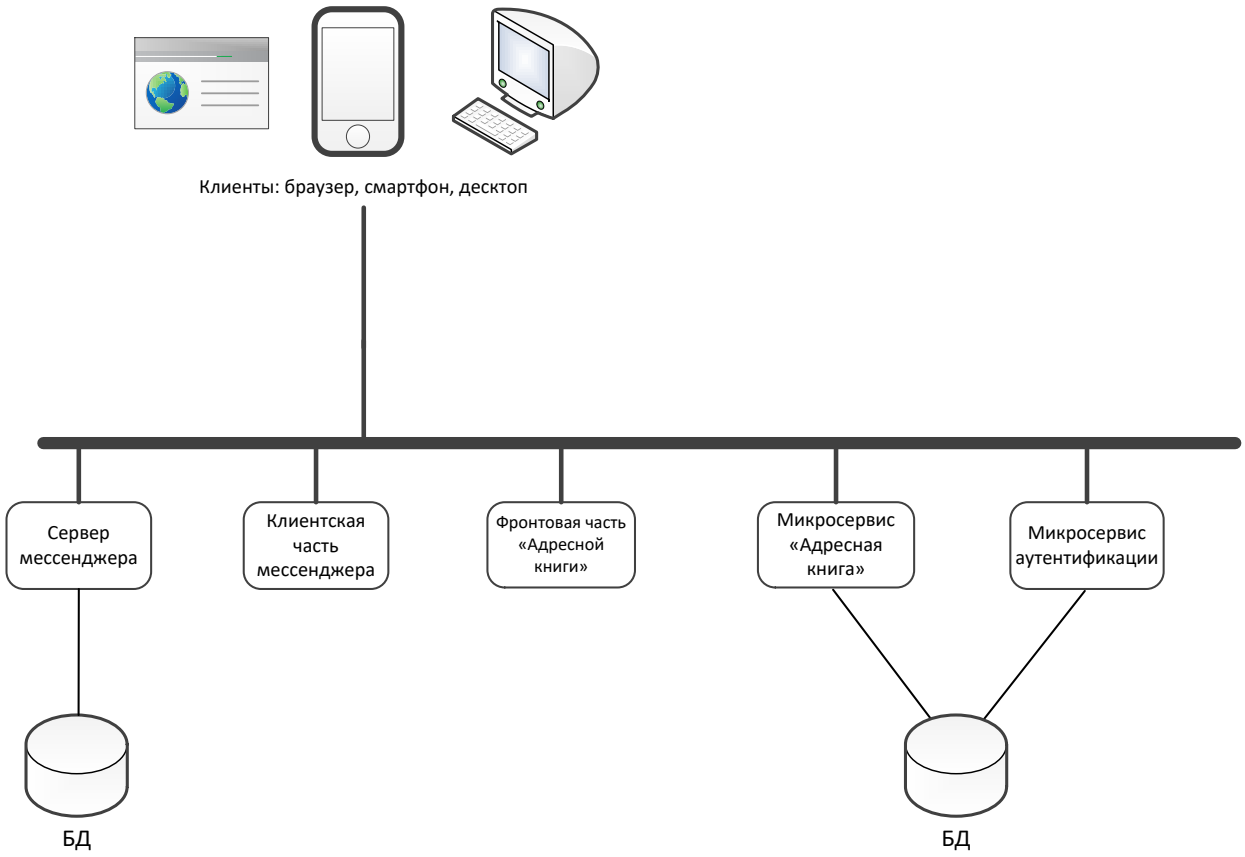


Рисунок 3 — Структурная схема интернет-мессенджера

Процесс разработки микросервиса можно разделить на следующие этапы:

- разработка структуры БД;
- разработка *REST API*;
- разработка программы.

2.1 Структура БД

При разработке сервиса адресная книга была использована СУБД *PostgreSQL*. Далее представлено описание структуры разработанной БД. Таблица 1 содержит описание структуры таблицы «Организации (ab_company)».

Таблица 1 — Структура таблицы «Организации (ab_company)».

Название поля	Описание поля	Тип данных
id	Идентификатор организации	Integer
name	Наименование организации	VarChar(255)
entry_name	Короткое наименование организации для входа	VarChar(255)
description	Описание	VarChar(512)
company_hierarchy	Иерархия организации	ltree()
is_blocked	Признак блокировки организации	Boolean
updated_at	Дата последнего обновления	TimeStamp
synced_at	Дата последней синхронизации	TimeStamp
registered_at	Дата создания организации	TimeStamp
is_active_directory	Признак выгрузки данных организации из Active Directory	Boolean
user_count	Количество пользователей организации/департамента	Integer
avatar	Аватар	VarChar(255)
avatar_origin	Изначальный файл	VarChar(255)
avatar_small	Аватар (маленький файл)	VarChar(255)

Таблица 2 содержит описание структуры таблицы «Атрибуты организации (ab_company_attribute)».

Таблица 2 — Структура таблицы «Атрибуты организации (ab_company_attribute)».

Название поля	Описание поля	Тип данных
id	Идентификатор атрибута организации	Integer
company_id	Идентификатор организации	Integer
attribute_type_id	Идентификатор атрибута	Integer
value	Значение атрибута	VarChar()
parent_company_attribute_id	Идентификатор родительского атрибута компании	Integer

Таблица 3 содержит описание структуры таблицы «Перечень атрибутов для организации (ab_company_attribute_type)».

Таблица 3 — Структура таблицы «Перечень атрибутов для организации (ab_company_attribute_type)».

Название поля	Описание поля	Тип данных
id	Идентификатор атрибута	Integer
name	Наименование атрибута	VarChar(255)
code	Наименование поля в *JSON*	VarChar(255)
description	Описание поля	VarChar(512)
reg_exp	Регулярное выражение для проверки валидности	VarChar(4096)

Таблица 4 содержит описание структуры таблицы «Роли пользователей в системе (ab_user_role)».

Таблица 4 — Структура таблицы «Роли пользователей в системе (ab_user_role)».

Название поля	Описание поля	Тип данных
id	Идентификатор роли	Integer
name	Наименование роли пользователя	VarChar(255)

Таблица 5 содержит описание структуры таблицы «Права пользователей в системе (ab_user_grant)».

Таблица 5 — Структура таблицы «Права пользователей в системе (ab_user_grant)».

Название поля	Описание поля	Тип данных
id	Идентификатор прав	Integer
name	Наименование прав пользователя	VarChar(255)

Таблица 6 содержит описание структуры таблицы «Права роли пользователя в системе (ab_user_grant)».

Таблица 6 — Структура таблицы «Права роли пользователя в системе (ab_user_grant)».

Название поля	Описание поля	Тип данных
id	Идентификатор	Integer
role_id	Идентификатор роли	Integer
grant_id	Идентификатор прав	Integer

Таблица 7 содержит описание структуры таблицы «Группы рассылки (ab_group)».

Таблица 7 — Структура таблицы «Группы рассылки (ab_group)».

Название поля	Описание поля	Тип данных
id	Идентификатор группы	Integer
name	Наименование группы	VarChar(255)
company_id	Идентификатор компании	Integer
description	Описание	VarChar(512)
avatar	Аватар	VarChar(255)
avatar_origin	Изначальный файл	VarChar(255)
avatar_small	Аватар (маленький файл)	VarChar(255)
created_at	Дата создания группы	TimeStamp
user_count	Количество пользователей группы	Integer

Таблица 8 содержит описание структуры таблицы «Пользователи и контакты системы (ab_user)».

Таблица 8 — Структура таблицы «Пользователи и контакты системы (ab_user)».

Название поля	Описание поля	Тип данных
id	Идентификатор пользователя	Integer
company_id	Идентификатор организации	Integer
name	Имя пользователя	VarChar(255)
login	Логин для входа в систему (login)	VarChar(255)
password_hash	Пароль для входа в систему	VarChar(255)
avatar	Аватар	VarChar(255)
avatar_origin	Изначальный файл	VarChar(255)
avatar_small	Аватар (маленький файл)	VarChar(255)
role_id	Роль пользователя	Integer
disk_space_limit	Доступное дисковое пространство	Integer
can_create_conference	Возможность создания конференции	Boolean
priority	Приоритет пользователей (в рамках организации)	Integer
is_blocked	Состояние заблокированности пользователя	Boolean
registered_at	Дата регистрации пользователя	TimeStamp
last_login_at	Дата последнего входа	TimeStamp
updated_at	Дата последнего изменения данных пользователя	TimeStamp
synced_at	Дата последней синхронизации данных	TimeStamp
description	Описание пользователя	VarChar(512)
is_active_directory	Признак загрузки из ActiveDirectory	Boolean

Таблица 9 содержит описание структуры таблицы «Группы пользователя (ab_user_group)».

Таблица 9 — Структура таблицы «Группы пользователя (ab_user_group)».

Название поля	Описание поля	Тип данных
id	Идентификатор группы пользователя	Integer
user_id	Идентификатор пользователя	Integer
group_id	Идентификатор группы	Integer

Таблица 10 содержит описание структуры таблицы «Перечень атрибутов для пользователей (ab_user_attribute_type)».

Таблица 10 — Структура таблицы «Перечень атрибутов для пользователей (ab_user_attribute_type)».

Название поля	Описание поля	Тип данных
id	Идентификатор атрибута	Integer
name	Наименование атрибута	VarChar(255)
code	Наименование поля в *JSON*	VarChar(255)
description	Описание поля	VarChar(512)
reg_exp	Регулярное выражение для проверки валидности	VarChar(4096)

Таблица 11 содержит описание структуры таблицы «Атрибуты пользователя (ab_user_attribute)».

Таблица 11 — Структура таблицы «Атрибуты пользователя (ab_user_attribute)».

Название поля	Описание поля	Тип данных
id	Идентификатор атрибута пользователя	Integer
user_id	Идентификатор пользователя	Integer
attribute_type_id	Идентификатор атрибута	Integer
value	Значение атрибута	VarChar()
parent_user_attribute_id	Идентификатор родительского атрибута (для сложных объектов)	Integer

Таблица 12 содержит описание структуры таблицы «Адресная книга пользователя (ab_user_favorite)».

Таблица 12 — Структура таблицы «Адресная книга пользователя (ab_user_favorite)».

Название поля	Описание поля	Тип данных
id	Идентификатор контакта	Integer
user_id	Идентификатор пользователя	Integer
user_favorite_id	Идентификатор контакта	Integer
rewritten_name	Переопределенное пользователем имя контакта	VarChar(255)
priority	Приоритет контакта	Integer

Таблица 13 содержит описание структуры таблицы «Контакты пользователя (ab_contact)».

Таблица 13 — Структура таблицы «Контакты пользователя (ab_contact)».

Название поля	Описание поля	Тип данных
id	Идентификатор контакта пользователя	Integer
name	Имя контакта	VarChar(255)
avatar	Аватар	VarChar(255)
avatar_origin	Изначальный файл	VarChar(255)
avatar_small	Аватар (маленький файл)	VarChar(255)
priority	Приоритет отображения контакта	Integer
registered_at	Дата регистрации контакта	TimeStamp
updated_at	Дата последнего изменения данных пользователя	TimeStamp
user_id	Идентификатор пользователя - владельца контакта	Integer

Таблица 14 содержит описание структуры таблицы «Атрибуты контактов (ab_contact_attribute)».

Таблица 14 — Структура таблицы «Атрибуты контактов (ab_contact_attribute)».

Название поля	Описание поля	Тип данных
id	Идентификатор атрибута контакта	Integer
contact_id	Идентификатор контакта	Integer
attribute_type_id	Идентификатор атрибута	Integer
value	Значение атрибута	VarChar()
parent_contact_attribute_id	Идентификатор родительского атрибута контакта	Integer

Листинг *SQL*-файла, формирующего БД находится в приложении В.

2.2 Описание REST API

Разработка *API* один из важнейших этапов создания микросервиса, так как от полноты реализации функционала *API* зависит удобство реализации остальных микросервисов и производительность мессенджера. *API*, описанное в данной выпускной работе, соответствует архитектурному стилю *REST* и поддерживает следующие функции:

- Создание пользователя;
- Обновление пользователя;
- Смена пароля пользователя;
- Удаление пользователя;
- Получение пользователя по *ID*;
- Получение атрибутов пользователя;
- Получение атрибутов пользователя всех разом;
- Создание атрибута пользователя;
- Обновление атрибута пользователя;
- Удаление атрибута пользователя;
- Получение атрибута пользователя по *ID*;
- Получение авторизованного пользователя;
- Получение пользователей;
- Получение пользователей по списку идентификаторов.

Полное описание *REST API* для работы с пользователями находится в приложении С.

Для примера рассмотрим запрос на создание пользователя.

Спецификация *REST API* требует того, чтобы при операциях создания использовался *HTTP* метод *POST*. Также представляются требования к формированию *URI* запроса, например для данного запроса *URI* имеет следующую структуру: название микросервиса/название сущности/операция

(или идентификатор объекта). Тело запроса состоит из полей, которые необходимы для создания пользователя. При успешном создании пользователя возвращается код 200 и объект, описывающий созданного пользователя (который включает в себя поля, переданные при создании и новые поля, добавленные после создания). В случае ошибки выдается код 400 и сообщение «*Bad Request*».

Листинг 1. Запрос на создание пользователя:

```
Метод: POST

URI:/addressbook/user/create

Тело запроса:
        {
          "name": " string",
          "login": " string",
          "password": " string",
          "companyId": int ,
          "accountType": "string",
          "diskSpaceLimit": int,
          "priority": int,
          "blocked": "boolean",
          "attributes": [
                   {"#attributeType#": "string"}
                 ],
          "grants": {
            "canCreateConference": "boolean"
                },
          "avatarUri": "string"
        }
Ответ:
     200   OK
             {
              "id": "string",
              "name": " string",
              "login": " string"
              "companyId": " string" ,
              "companyName": " string" ,
              "accountType": "string",
              "blocked": boolean,
```

```
            "priority": int,
            "diskSpaceLimit": int,
            "avatarUri": "string",
            "avatarOriginUri": "string",
            "avatarSmallUri": "string",
            "registeredAt":"int",
            "groups": [ int ],
            "attributes": [
              {"#attributeType#": "string"}
            ],
            "grants": {
                    "canCreateConference": boolean
                  }
          }
400   Bad Request
      { "errorMessage": "string" }
```

2.3 Описание классов

В микросервисе «Адресная книга» для взаимодействия ранее описанных СУБД и *REST API* архитектура разделена на основные функциональные блоки:

- классы-котроллеры – выполняют запросы *REST API*;
- классы-сущности – отвечают за описание сущностей БД в программе. Классы-сущности облегчают работу с данными, полученными, как результат использования функций классов-репозиториев;
- классы-репозитории – выполняют взаимодействие СУБД с сущностями. Данные передаются в эти классы из классов-сервисов;
- классы-сервисы – обеспечивают взаимодействие классов-контроллеров с классами-репозиториями посредством подготовки данных.

На примере сущности «Пользователь» рассмотрим взаимодействие всех типов классов.

Класс-сущность описывает структуру таблицы «Пользователь», а также описывается отношение сущности к остальным сущностям (сущности «Организация», «Атрибут пользователя», «Группа», «Контакт»). Класс содержит методы, позволяющие получить или установить то или иное свойство или поле объекта (логин, имя пользователя, хэш суммы пароль, аватар пользователя и др.). Классы-сущности облегчают работу с данными, полученными, как результат использования функций классов-репозиториев.

UML диаграмма классов сущности «Пользователь» представлена на рисунке 4.

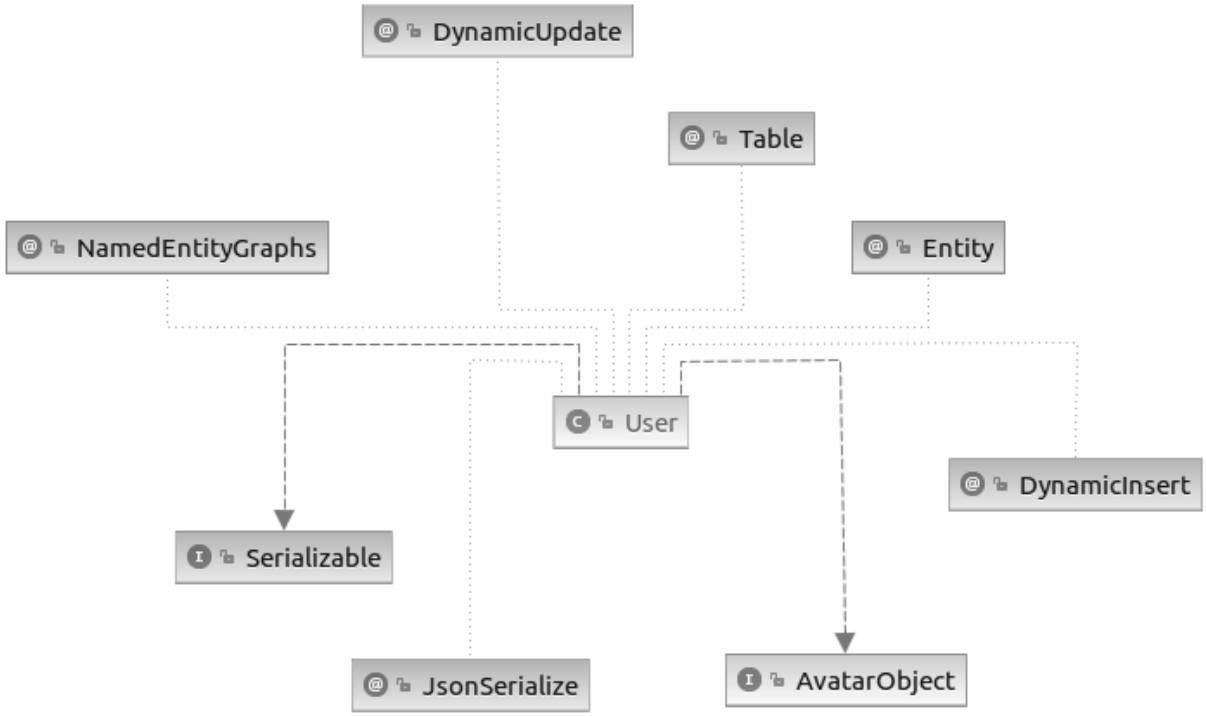


Рисунок 4 — *UML* диаграмма классов сущности «Пользователь»

Класс-контроллер «UserRestController» осуществляет обработку входящих запросов от других микросервисов, валидацию входных данных, вызов методов класса-сервиса («UserService») и возврат результата.

UML диаграмма, показывающая структуру класса-контроллера «UserRestController», представлена на рисунке 5.

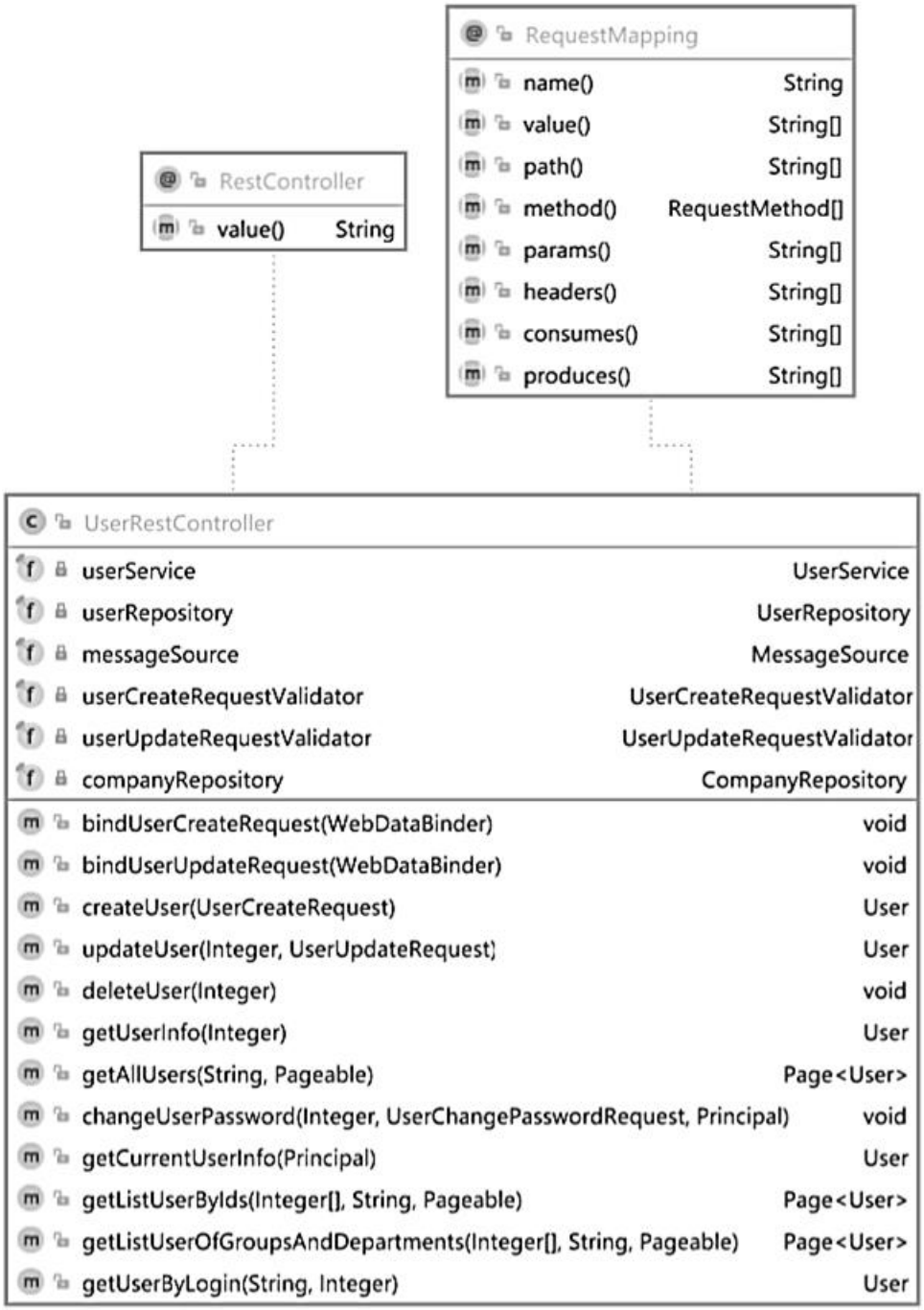


Рисунок 5 — *UML* диаграмма класса-контроллера «UserRestController»

Класс-репозиторий «UserRepository» работает с СУБД и осуществляет операции создания, обновления, удаления и поиска пользователей. При этом операции поиска представляют собой набор специфичных для каждого запроса, обрабатываемого контроллером («UserRestController») функций,

которые ищут пользователя на основе ограничений, передаваемых при запросе. Например, поиск по ID пользователя, получение всех пользователей группы (или групп). Ниже на рисунке 6 представлена *UML* диаграмма класса-репозитория «UserRepository».

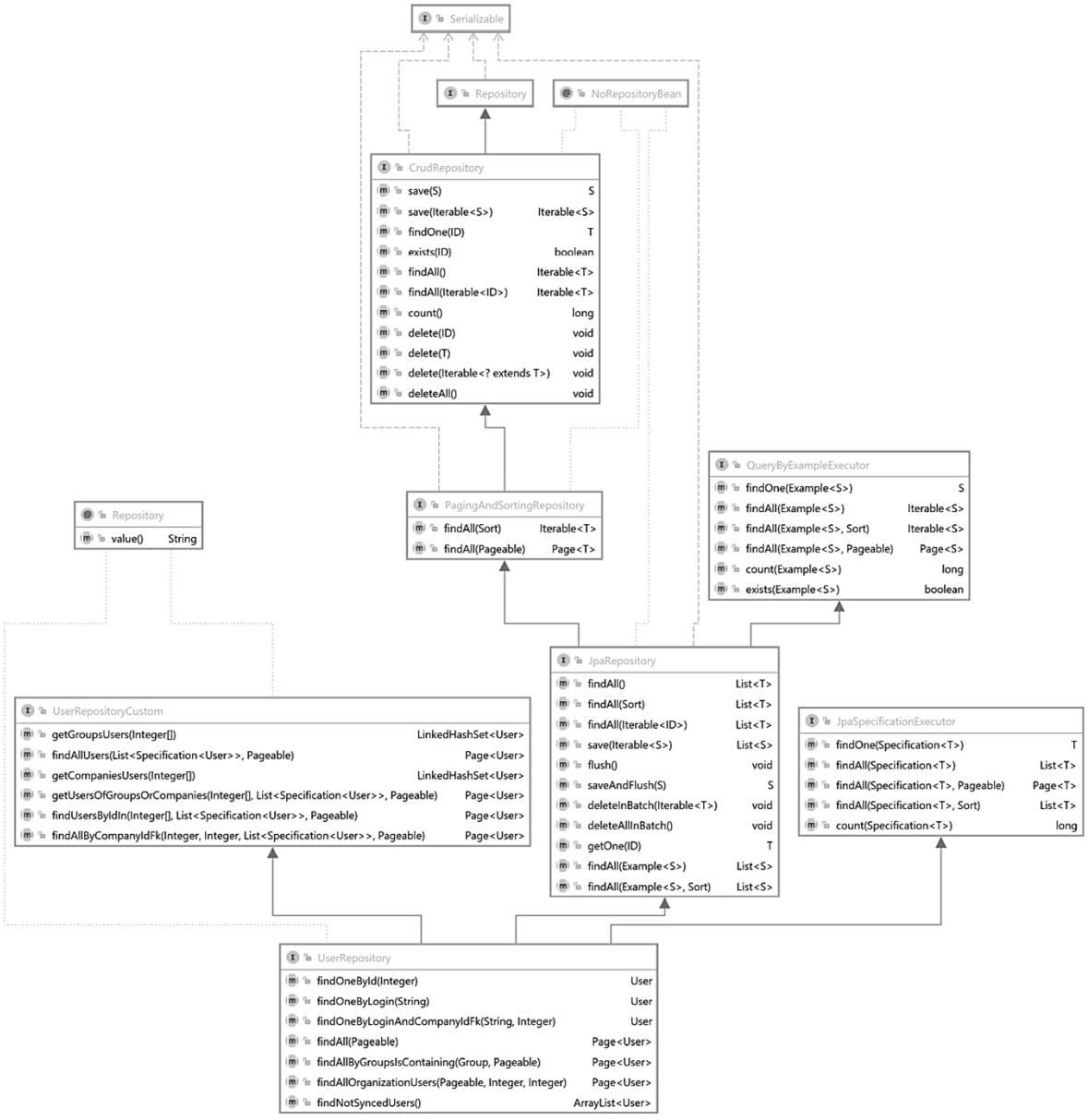


Рисунок 6 — *UML* диаграмма класса-контроллера «UserRepository»

Класс-сервис «UserServiceImpl» реализует класс «UserService», наследуя из него методы. Функции данного класса позволяют работать с объектами, описывающими пользователей, производя подготовку объектов к

передаче для обработки в класс-репозиторий. *UML*-диаграмма класса-репозитория «UserServiceImpl» представлена на рисунке 7.

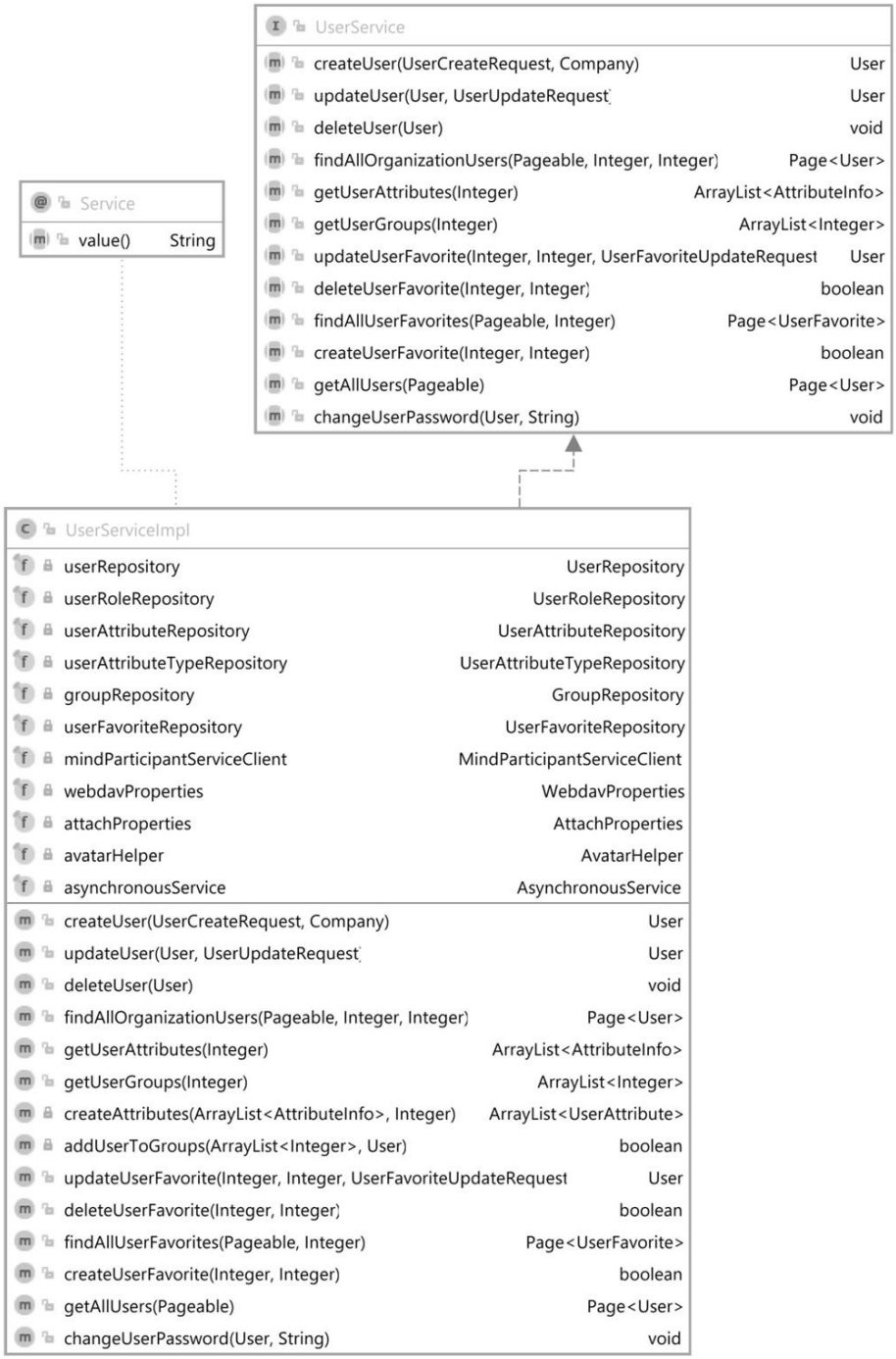


Рисунок 7 — *UML* диаграмма класса-контроллера «UserServiceImpl»

2.4 Программная реализация

Как упоминалось в предыдущем разделе, для реализации данного микросервиса используется язык программирования *Java* и фреймворк *Spring*. Был реализован *API* для работы с пользователями, компаниями, группами и контактами, а также функции для работы с ним. Исходный код описанных в предыдущем разделе классов приведен в приложении X.

На *UML* диаграмме (Рисунок 8) показана последовательность взаимодействия классов при обработке запроса «Получение пользователя по ID».

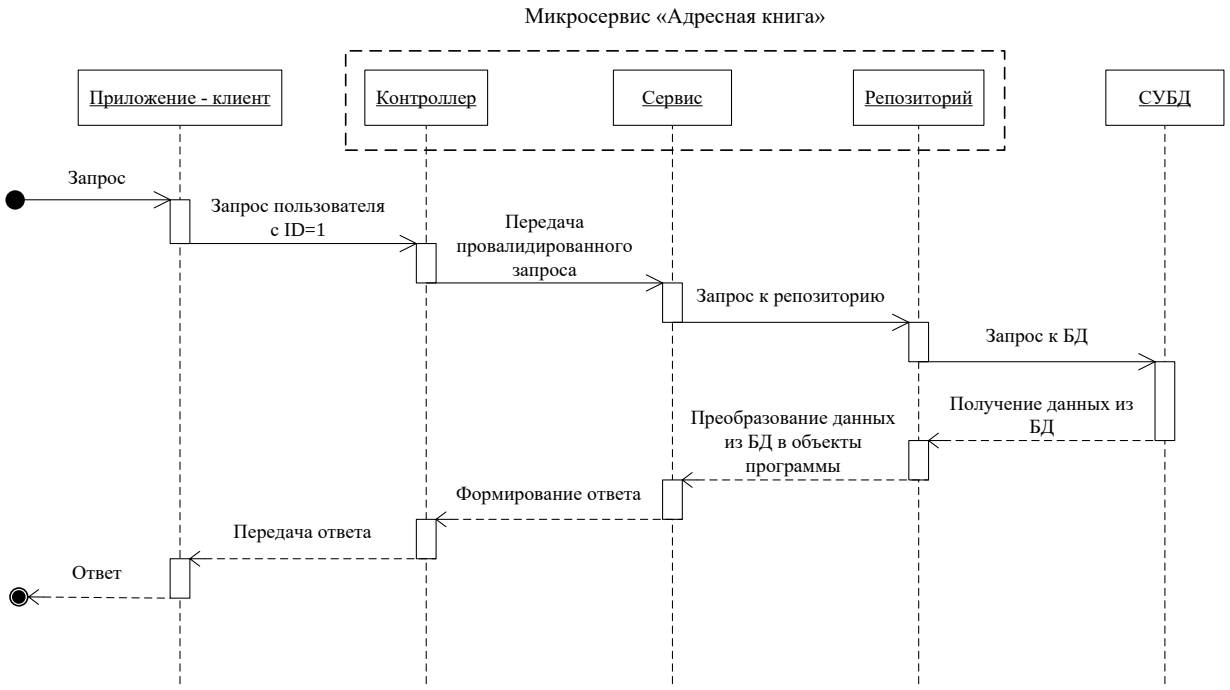


Рисунок 8 — Последовательность взаимодействия классов при обработке запроса «Получение пользователя по ID»

Пользователь использует функционал клиентского приложения (нажимает на кнопку), который в свою очередь генерирует запрос к клиентскому приложению. С клиентского приложения в микросервис «Адресная книга» в объект класса «Контроллер» поступает запрос на получение пользователя по ID. Класс-контроллер осуществляет проверку

запроса на правильность, и если запрос удовлетворяет требованиям валидации, то класс-контроллер передает провалидированный запрос в класс-сервис. Класс-сервис подготавливает запрос к взаимодействию с объектами класса-репозитория. Класс-репозиторий отправляет запрос к БД, осуществляя взаимодействие с ней. Затем, когда необходимые данные получены из БД, класс-репозиторий преобразует данные в объекты программы и передает их в класс-сервис. Класс-сервис формирует ответ для класса-контроллера, класс-контроллер выдает результат, который передается клиентскому приложению. Пользователю виден результат использования того или иного функционала (нажатие на определенную кнопку).

Исходный код на языке *Java* класса-сущности «Пользователь», класса-контроллера, сервиса и репозитория находится в приложении А.

3 Описание результата

В результате компиляции программы был получен исполняемый файл «adressbook.jar». Для его запуска можно использовать команду «*Java* –jar adressbook.jar». Результат запуска представлен на рисунке 9.

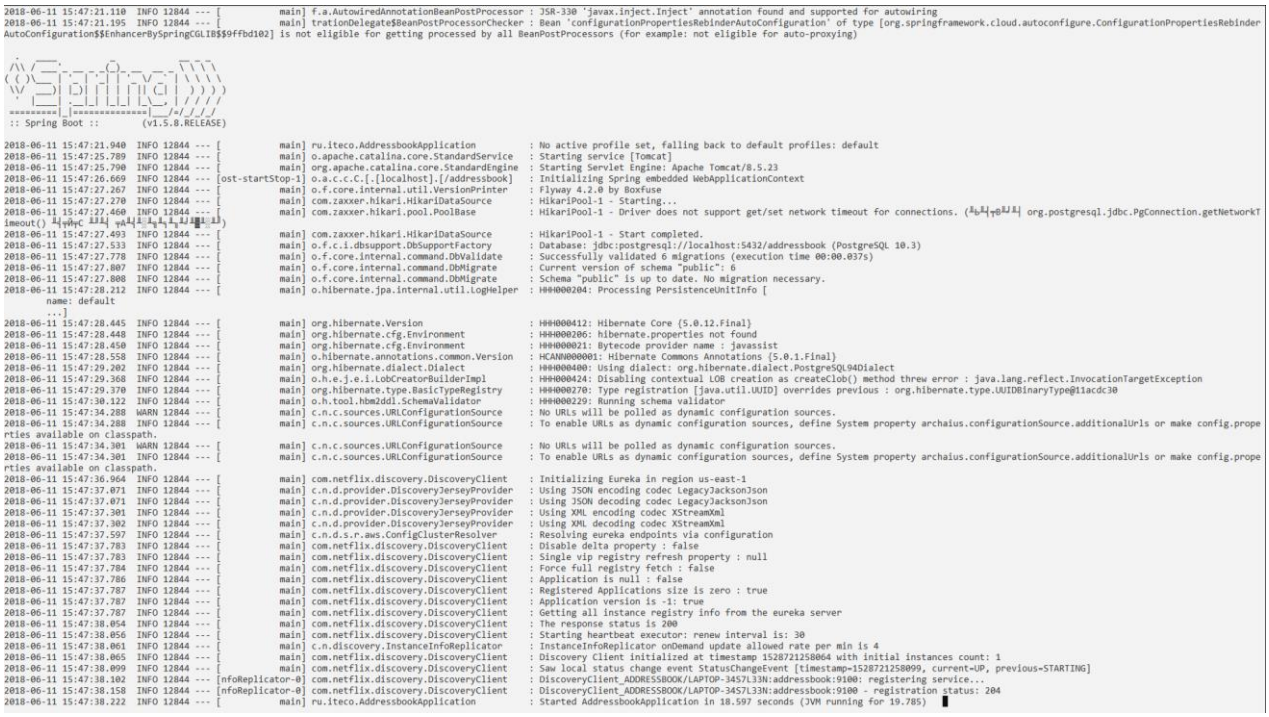


Рисунок 9 — Результат работы исполняемого файла *«adressbook.jar»* сразу после запуска.

Для проверки работоспособности приложения используется программа *Insomnia REST Client*, поскольку она бесплатная и обладает широким функционалом по отладке *REST API*.

Структура базы данных адресной книги подразумевает, что пользователи не могут существовать в отрыве от организации, поэтому перед созданием первого пользователя необходимо создать организацию, в которую будет добавлен пользователь. Для этого можно использовать запрос «*POST /addressbook/company/create*» в тело которого добавляется описание создаваемой организации в формате *JSON*. При успешном выполнении запроса возвращается объект с описанием организации, в котором особое внимание стоит уделить полю «*id*». Значение этого поля необходимо указать

в запросе создания пользователя в поле «*companyId*», чтобы привязать пользователя к организации. Результат выполнения запроса на создание организации представлен на рисунке 10.

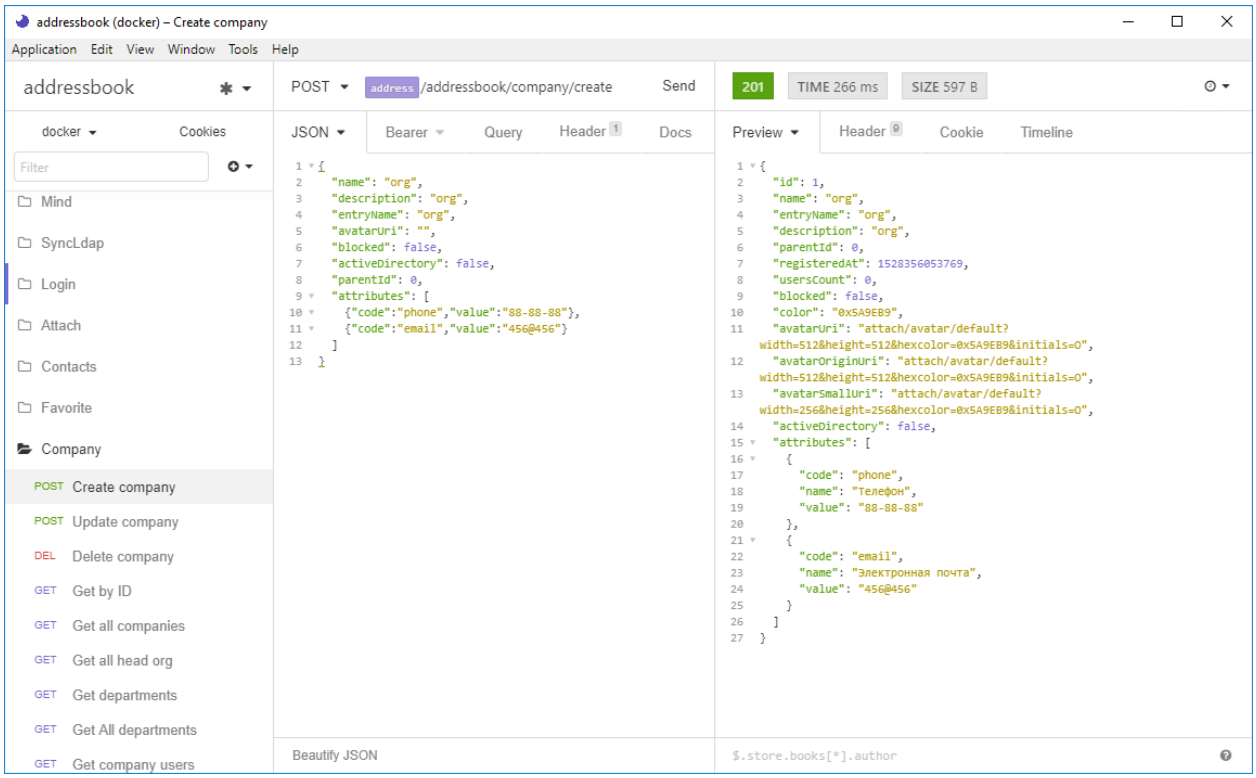


Рисунок 10 — Результат выполнение запроса "Создание организации" в программе *Insomnia REST Client*

Теперь, когда организация создана, можно создать в ней первого пользователя. Для этого в *API* «Адресной книги» предусмотрен запрос «*POST /addressbook/user/create*» в тело запроса которого добавляется описание в формате *JSON* с данными создаваемого пользователя. В случае успешного создания «Адресная книга» возвращает объект с описанием созданного пользователя так же в формате *JSON* (рисунок 11).

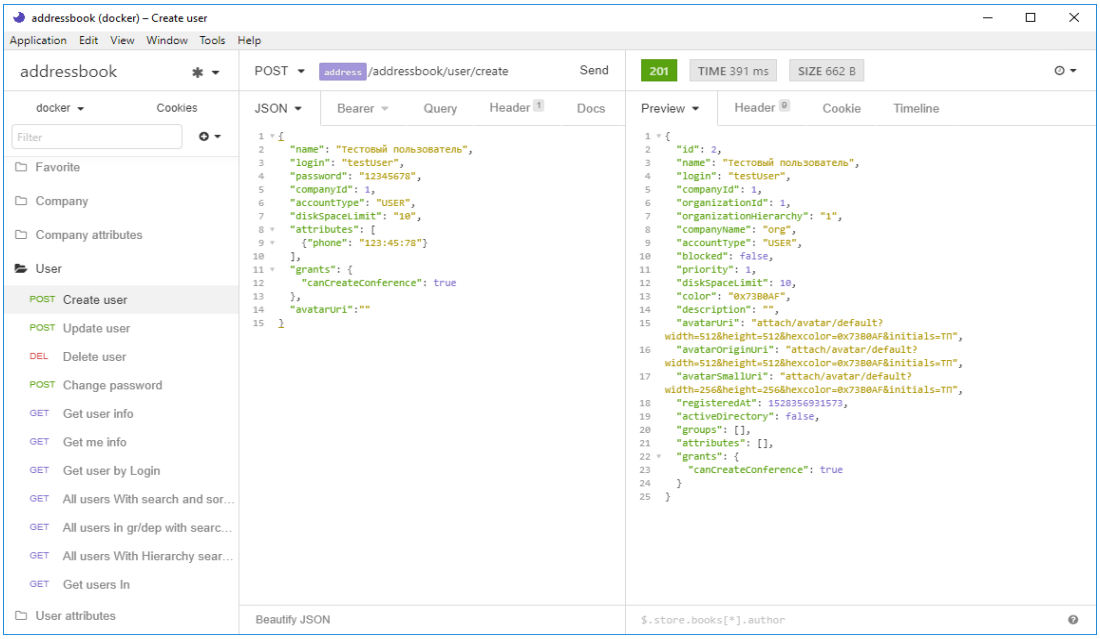


Рисунок 11 — Результат выполнение запроса "Создание пользователя" в программе *Insomnia REST Client*

После успешного создания мы можем запрашивать инфомацию о пользователе используя запрос «*GET /addressbook/user/[userId]*», где вместо *[userId]* подставляется идентификатор нужного пользователя, в нашем примере *userId* равен 2. Результат выполнения запроса «Получение пользователя по *ID*» показан на рисунке 12.

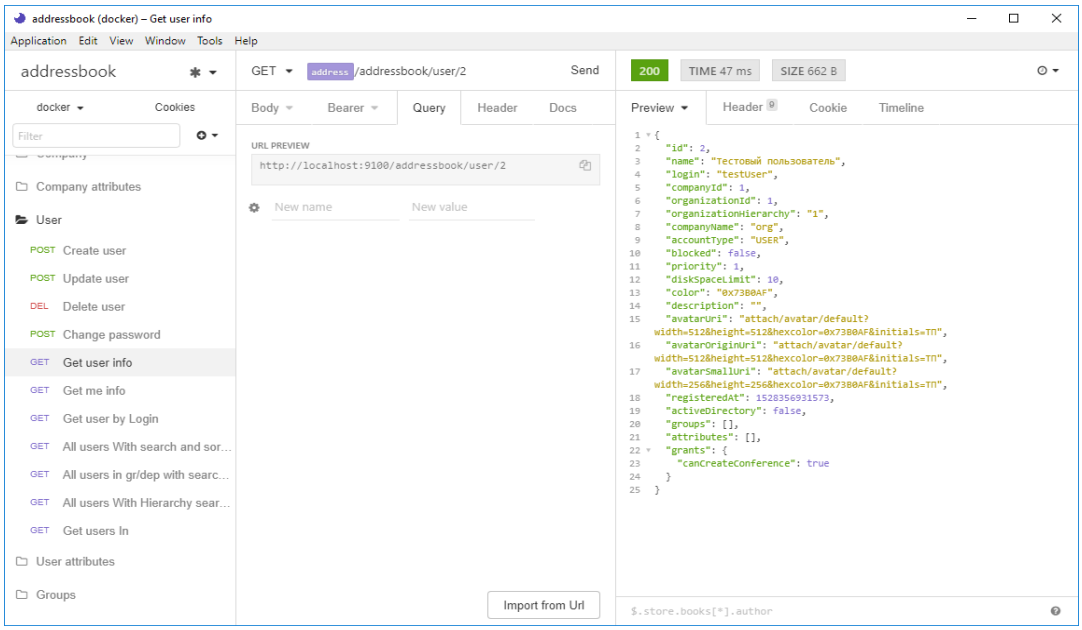


Рисунок 12 — Результат выполнения запроса «Получение пользователя по ID» в программе *Insomnia REST Client*

Часто необходимо обновлять информацию о пользователе, который был создан. Для таких случаев в *API* предусмотрен запрос на обновление пользователя «*POST /addressbook/user/[userId]*», в тело которого добавляется измененное описание пользователя в формате *JSON*. Вместо *[userId]* подставляется идентификатор нужного пользователя, в нашем примере *userId* как и в запросе ранее равен 2. В «Адресной книге» стоит запрет на смену логина пользователя. Результат выполнения запроса «Обновление пользователя по *ID*» в программе *Insomnia REST Client* показан на рисунке 13.

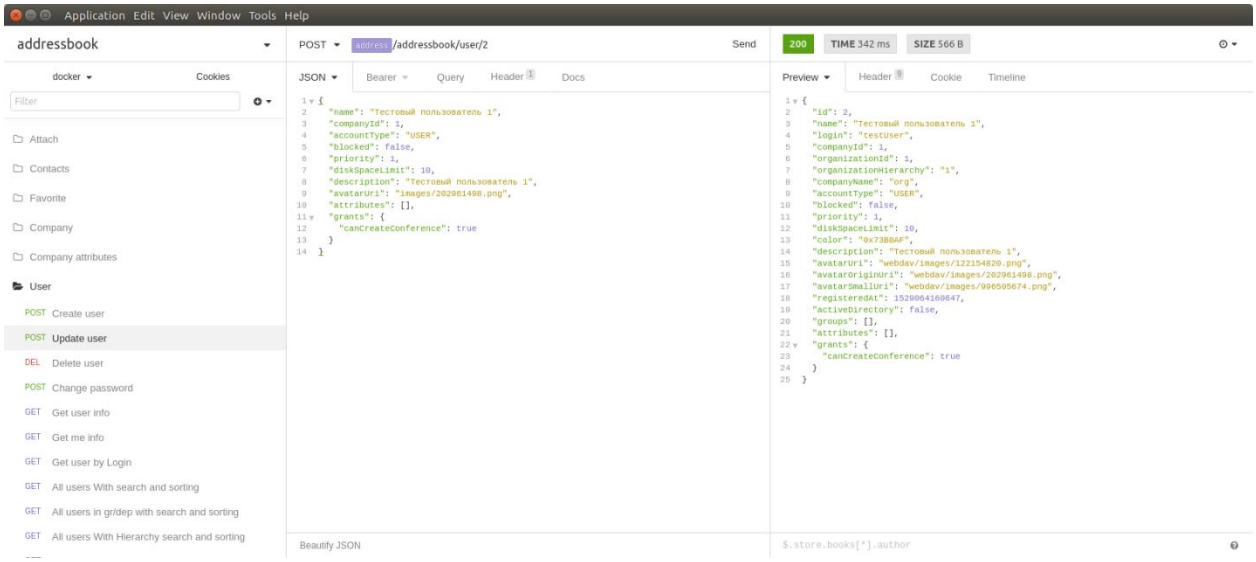


Рисунок 13 — Результат выполнения запроса «Обновление пользователя по ID» в программе *Insomnia REST Client*

В запросе передается новое имя пользователя (не путать с логином), описание, и прикрепляется аватар (в поле avatarUri добавляется ссылка на ранее загруженный файл).

4 Заключение

В ходе выполнения данной работы была изучена методология проектирования приложения с микросервисной архитектурой и приобретены навыки работы с фрейворком *Spring*. При реализации поставленной задачи были получены следующие результаты:

- разработана структура базы данных для СУБД *PostgreSQL*

- разработан прикладной интерфейс для взаимодействия с остальными микросервисами приложения (*REST API*)

- Разработано и отлажено приложение «Адресная книга».

Сервис поддерживает полный функционал для интеграции с остальными микросервисами мессенджера.

Список используемых источников

1. Мартин Фаулер, Прамодкумар Дж. Садаладж. NoSQL: новая методология разработки нереляционных баз данных = NoSQL Distilled. — М.: «Вильямс», 2013. — С. 50- 192.
2. Роберт Э. Уолтерс, Майкл Коулс. SQL Server 2008: ускоренный курс для профессионалов = Accelerated SQL Server 2008. — М.: «Вильямс», 2008. — С. 25-105.
3. Том Кайт. Oracle для профессионалов: архитектура, методики программирования и особенности версий 9i, 10g и 11g, 2-е издание = Expert Oracle Database Architecture: Oracle Database Programming 9i, 10g, and 11g Techniques and Solutions, Second Edition. — М.: «Вильямс», 2011. — С. 150-580.
4. Николай Прохоренок, Владимир Дронов. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. — СПб.: «БХВ-Петербург», 2015. — С. 20-348.
5. Дж. Уорсли, Дж. Дрейк. PostgreSQL . Для профессионалов — СПб.: «Питер», 2003. — С. 23-320.
6. Леон Шкляр, Рич Розен. Архитектура веб-приложений. — М.: «Эксмо», 2010. — С.35-150.
7. Ньюмен С. Создание микросервисов = Building Microservices. — СПб.: «Питер», 2016. — С. 56-246.
8. Марк Саммерфильд. Программирование на Go. Разработка приложений XXI века. — М.: «ДМК Пресс», 2016. — С. 34-251.
9. Марк Саммерфилд. Python на практике. — Перевод с английского. — М.: «ДМК Пресс», 2014. — С. 120-310.
10. Эрик Фримен, Элизабет Робсон. Изучаем программирование на JavaScript. — СПб.: «Питер», 2018. — С.125-340.

11. Шилдт Г. Java 8: руководство для начинающих. — М.: «Вильямс», 2018. — С. 149- 348.

12. Erik Wilde, Cesare Pautasso. REST: From Research to Practice. — «Springer Science & Business Media», 2011. — С. 52-340.

Приложение А. Листинги программы на Java

Файл «User.java»

```
package ru.psu.core.account;

import com.fasterXML.jackson.annotation.JSONIgnore;
import com.fasterXML.jackson.annotation.JSONProperty;
import com.fasterXML.jackson.databind.annotation.JSONSerialize;
import org.hibernate.annotations.*;
import org.hibernate.annotations.Parameter;
import ru.psu.core.common.AvatarObject;
import ru.psu.core.company.Company;
import ru.psu.core.contact.Contact;
import ru.psu.core.group.Group;

import javax.persistence.*;
import javax.persistence.Entity;
import javax.persistence.ForeignKey;
import javax.persistence.Table;
import java.io.Serializable;
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "ab_user")
@DynamicInsert
@DynamicUpdate
@NamedEntityGraph(name =
"user.company.userRole.groups.contacts.attributes",
        attributeNodes = {
                @NamedAttributeNode("company"),
                @NamedAttributeNode("userRole"),
                @NamedAttributeNode("groups"),
                @NamedAttributeNode("contacts"),
                @NamedAttributeNode(value = "attributes",
subgraph = "attributes.userAttributeType")},
        subgraphs = {
                @NamedSubgraph(
                        name = "attributes.userAttributeType",
                        attributeNodes = {

@NamedAttributeNode("userAttributeType")
                        }
                )
        })
@JSONSerialize(using = UserSerializer.class)
public class User implements Serializable, AvatarObject {
```

```
    @GenericGenerator(
            name = "hibernate_ab_company_group_id_seq",
            strategy =
"org.hibernate.id.enhanced.SequenceStyleGenerator",
            parameters = {
                    @Parameter(name = "sequence_name", value =
"hibernate_ab_company_group_id_seq"),
                    @Parameter(name = "INCREMENT", value = "1"),
                    @Parameter(name = "MINVALUE", value = "1"),
                    @Parameter(name = "MAXVALUE", value =
"9223372036854775807"),
                    @Parameter(name = "CACHE", value = "1")
            }
    )
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = "hibernate_ab_company_group_id_seq")
    @Column(name = "id", nullable = false, updatable = false)
    private Integer id;

    @JSONIgnore
    @OneToOne
    @JoinColumn(name = "company_id",
            insertable = false,
            updatable = false,
            foreignKey = @ForeignKey(name =
"fk_ab_user_ab_company"))
    private Company company;

    @JSONProperty("companyId")
    @Column(name = "company_id", nullable = false)
    private Integer companyIdFk;

    @JSONProperty("name")
    @Column(name = "name", nullable = false)
    private String userName;

    @Column(name = "login", nullable = false, unique = true)
    private String login;

    @JSONIgnore
    @Column(name = "password_hash", nullable = false)
    private String passwordHash;

    @Column(name = "avatar")
    private String avatarUri;

    @JSONProperty("avatarOriginUri")
    @Column(name = "avatar_origin")
    private String avatarOriginUri;
```

```
    @JSONProperty("avatarSmallUri")
    @Column(name = "avatar_small")
    private String avatarSmallUri;

    @JSONProperty("accountType")
    @OneToOne
    @JoinColumn(name = "role_id",
            foreignKey = @ForeignKey(name =
"fk_ab_user_ab_user_role"))
    private UserRole userRole;

    @JSONIgnore
    @Column(name = "role_id", insertable=false, updatable=false)
    private Integer roleIdFk;

    @Column(name = "disk_space_limit", nullable = false)
    private Integer diskSpaceLimit;

    @JSONProperty("canCreateConference")
    @Column(name = "can_create_conference", nullable = false)
    private Boolean isCanCreateConference;

    @ColumnDefault("1")
    @Column(name = "priority", nullable = false)
    private Integer priority;

    @JSONProperty("blocked")
    @ColumnDefault("false")
    @Column(name = "is_blocked", nullable = false)
    private Boolean isBlocked;

    @Column(name = "registered_at")
    private Date registeredAt;

    @JSONIgnore
    @Column(name = "last_login_at")
    private Date lastLoginDate;

    @JSONIgnore
    @Column(name = "updated_at")
    private Date lastUpdateDate;

    @JSONIgnore
    @Column(name = "synced_at")
    private Date lastSyncDate;

    @ManyToMany
    @JoinTable(name = "ab_user_group",
            joinColumns = {@JoinColumn(name = "user_id")},
```

```
            inverseJoinColumns = {@JoinColumn(name =
"group_id")}
    )
    private Set<Group> groups = new HashSet<>();

    @JSONIgnore
    @OneToMany(mappedBy = "user")
    private Set<Contact> contacts = new HashSet<>();

    @OneToMany(mappedBy = "user")
    private Set<UserAttribute> attributes = new HashSet<>();

    @Column(name = "description")
    private String description;

    @ColumnDefault("false")
    @Column(name = "is_active_directory", nullable = false)
    private Boolean isActiveDirectory;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Company getCompany() {
        return company;
    }

    public void setCompany(Company company) {
        this.company = company;
    }

    public Integer getCompanyIdFk() {
        return companyIdFk;
    }

    public void setCompanyIdFk(Integer companyIdFk) {
        this.companyIdFk = companyIdFk;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }
```

```
public String getLogin() {
    return login;
}

public void setLogin(String login) {
    this.login = login;
}

public String getPasswordHash() {
    return passwordHash;
}

public void setPasswordHash(String passwordHash) {
    this.passwordHash = passwordHash;
}

public String getAvatarUri() {
    if(avatarUri == null || avatarUri.isEmpty())
        return "";
    return avatarUri;
}

public void setAvatarUri(String avatarUri) {
    this.avatarUri = avatarUri;
}

public String getAvatarOriginUri() {
    if(avatarOriginUri == null || avatarOriginUri.isEmpty())
        return "";
    return avatarOriginUri;
}

public void setAvatarOriginUri(String avatarOriginUri) {
    this.avatarOriginUri = avatarOriginUri;
}

public String getAvatarSmallUri() {
    if(avatarSmallUri == null || avatarSmallUri.isEmpty())
        return "";
    return avatarSmallUri;
}

public void setAvatarSmallUri(String avatarSmallUri) {
    this.avatarSmallUri = avatarSmallUri;
}

public UserRole getUserRole() {
    return userRole;
}
```

```
    public void setUserRole(UserRole userRole) {
        this.userRole = userRole;
    }

    public Integer getRoleIdFk() {
        return roleIdFk;
    }

    public void setRoleIdFk(Integer roleIdFk) {
        this.roleIdFk = roleIdFk;
    }

    public Integer getDiskSpaceLimit() {
        return diskSpaceLimit;
    }

    public void setDiskSpaceLimit(Integer diskSpaceLimit) {
        this.diskSpaceLimit = diskSpaceLimit;
    }

    public Boolean getCanCreateConference() {
        return isCanCreateConference;
    }

    public void setCanCreateConference(Boolean
canCreateConference) {
        isCanCreateConference = canCreateConference;
    }

    public Integer getPriority() {
        return priority;
    }

    public void setPriority(Integer priority) {
        this.priority = priority;
    }

    public Boolean getBlocked() {
        return isBlocked;
    }

    public void setBlocked(Boolean blocked) {
        isBlocked = blocked;
    }

    public Date getRegisteredAt() {
        return registeredAt;
    }
```

```
public void setRegisteredAt(Date registeredAt) {
    this.registeredAt = registeredAt;
}

public Date getLastLoginDate() {
    return lastLoginDate;
}

public void setLastLoginDate(Date lastLoginDate) {
    this.lastLoginDate = lastLoginDate;
}

public Date getLastUpdateDate() {
    return lastUpdateDate;
}

public void setLastUpdateDate(Date lastUpdateDate) {
    this.lastUpdateDate = lastUpdateDate;
}

public Date getLastSyncDate() {
    return lastSyncDate;
}

public void setLastSyncDate(Date lastSyncDate) {
    this.lastSyncDate = lastSyncDate;
}

public Set<Group> getGroups() {
    return groups;
}

public void setGroups(Set<Group> groups) {
    this.groups = groups;
}

public Set<Contact> getContacts() {
    return contacts;
}

public void setContacts(Set<Contact> contacts) {
    this.contacts = contacts;
}

public Set<UserAttribute> getAttributes() {
    return attributes;
}

public void setAttributes(Set<UserAttribute> attributes) {
    this.attributes = attributes;
```

```
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public Boolean getActiveDirectory() {
        return isActiveDirectory;
    }

    public void setActiveDirectory(Boolean activeDirectory) {
        isActiveDirectory = activeDirectory;
    }

    public Integer getUserOrganization() {
        String hierarchy = this.company.getCompanyHierarchy();
        if(hierarchy != null) {
            String indexes[] = hierarchy.split("\\.");
            if (indexes.length > 0)
                return Integer.valueOf(indexes[0]);
            else
                return 0;
        }
        return 0;
    }
}
```

Файл «User*REST*Controller.java»

```
package ru.psu.core.account.web;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.MessageSource;
import org.springframework.context.i18n.LocaleContextHolder;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.data.web.PageableDefault;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import
org.springframework.security.core.authority.SimpleGrantedAuthori
ty;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
;
```

```
import
org.springframework.security.oauth2.provider.OAuth2Authenticatio
n;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.*;
import ru.psu.core.account.*;
import ru.psu.core.common.SearchHelper;
import ru.psu.core.company.Company;
import ru.psu.core.company.CompanyRepository;
import ru.psu.error.ApplicationException;

import javax.validation.Valid;
import java.security.Principal;
import java.util.Date;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

@RESTController
@RequestMapping(value = "/user")
public class UserRESTController {
    private final UserService userService;
    private final UserRepository userRepository;
    private final MessageSource messageSource;
    private final UserCreateRequestValidator
userCreateRequestValidator;
    private final UserUpdateRequestValidator
userUpdateRequestValidator;
    private final CompanyRepository companyRepository;

    @Autowired
    public UserRESTController(UserService userService,
                              UserRepository userRepository,
                              MessageSource messageSource,
                              UserCreateRequestValidator
userCreateRequestValidator,
                              UserUpdateRequestValidator
userUpdateRequestValidator,
                              CompanyRepository
companyRepository) {
        this.userService = userService;
        this.userRepository = userRepository;
        this.messageSource = messageSource;
        this.userCreateRequestValidator =
userCreateRequestValidator;
        this.userUpdateRequestValidator =
userUpdateRequestValidator;
        this.companyRepository = companyRepository;
    }
```

```
    @InitBinder("userCreateRequest")
    public void bindUserCreateRequest(WebDataBinder binder) {
        binder.addValidators(userCreateRequestValidator);
    }

    @InitBinder("userUpdateRequest")
    public void bindUserUpdateRequest(WebDataBinder binder) {
        binder.addValidators(userUpdateRequestValidator);
    }

    @PostMapping(value = {"/create"}, consumes =
MediaType.APPLICATION_JSON_UTF8_VALUE)
    @ResponseBody
    @ResponseStatus(HttpStatus.CREATED)
    public User createUser(@Valid @RequestBody UserCreateRequest
userCreateRequest)
            throws ApplicationException {
        Company company =
companyRepository.findOneById(userCreateRequest.getCompanyId());
        if (company == null) {
            String message = messageSource.getMessage(
                    "ru.psu.core.company.web.CompanyNotExists",
new Object[]{userCreateRequest.getCompanyId()},
LocaleContextHolder.getLocale());
            throw new ApplicationException(message);
        }
        User user = userService.createUser(userCreateRequest,
company);
        if(user == null) {
            throw new ApplicationException("Can't create user");
        }
        return user;
    }

    @PostMapping(value = {"/{userId}"}, consumes =
MediaType.APPLICATION_JSON_UTF8_VALUE)
    @ResponseBody
    public User updateUser(@PathVariable Integer userId,
                           @Valid @RequestBody UserUpdateRequest
userUpdateRequest)
            throws ApplicationException {
        User user = userRepository.findOneById(userId);
        if(userUpdateRequest.getPassword() != null &&
!userUpdateRequest.getPassword().isEmpty()) {
            if (new
BCryptPasswordEncoder().matches(userUpdateRequest.getPassword(),
user.getPasswordHash())) {
                String message = messageSource.getMessage(
                        "ru.psu.core.account.web.SamePassword",
new Object[]{}, LocaleContextHolder.getLocale());
```

```
                throw new ApplicationException(message);
            }
        }
        if (user == null) {
            String message = messageSource.getMessage(

"ru.psu.core.account.web.UserWithIdNotExists", new
Object[]{userId},
                    LocaleContextHolder.getLocale());
            throw new ApplicationException(message);
        }
        user = userService.updateUser(user, userUpdateRequest);
        if(user == null) {
            throw new ApplicationException("Can't update user");
        }
        return user;
    }

    @DeleteMapping(value = {"/{userId}"})
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteUser(@PathVariable Integer userId) throws
ApplicationException {
        User user = userRepository.findOneById(userId);
        if (user == null) {
            String message = messageSource.getMessage(

"ru.psu.core.account.web.UserWithIdNotExists", new
Object[]{userId}, LocaleContextHolder.getLocale());
            throw new ApplicationException(message);
        }
        userService.deleteUser(user);
    }

    @GetMapping(value = {"/{userId}"})
    @ResponseBody
    public User getUserInfo(@PathVariable Integer userId)
            throws ApplicationException {
        User user = userRepository.findOneById(userId);
        if(user == null) {
            String message = messageSource.getMessage(

"ru.psu.core.account.web.UserWithIdNotExists", new
Object[]{userId}, LocaleContextHolder.getLocale());
            throw new ApplicationException(message);
        }
        return user;
    }

    @GetMapping(value = {"", "/"})
    @ResponseBody
```

```
    public Page<User> getAllUsers(@RequestParam(value =
"search", required = false) String search,
                                  @PageableDefault(size = 10,
page = 0) Pageable pageable)
            throws ApplicationException {
        List<Specification<User>> spec =
SearchHelper.createSearchSpecificationList(search);
        return userRepository.findAllUsers(spec, pageable);
    }

    @PostMapping(value = {"/{userId}/changepass"}, consumes =
MediaType.APPLICATION_JSON_UTF8_VALUE)
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void changeUserPassword(@PathVariable Integer userId,
                                   @Valid @RequestBody
UserChangePasswordRequest userChangePasswordRequest,
                                   Principal principal)
            throws ApplicationException {
        User currentUser =
userRepository.findOneByLogin(principal.getName());
        User user = userRepository.findOneById(userId);
        if (user == null) {
            String message = messageSource.getMessage(

"ru.psu.core.account.web.UserWithIdNotExists", new
Object[]{userId}, LocaleContextHolder.getLocale());
            throw new ApplicationException(message);
        }
        // TODO: Временно не работает, исправить на фронте
        /*if(!currentUser.getId().equals(user.getId()) &&
!currentUser.getUserRole().getTypeName().equals("ADMIN")) {
            String message = messageSource.getMessage(
                    "ru.psu.core.account.web.NoPermissions", new
Object[]{}, LocaleContextHolder.getLocale());
            throw new NoPermissionException(message);
        }*/
        if(!(new
BCryptPasswordEncoder().matches(userChangePasswordRequest.getPas
sword(), user.getPasswordHash()))) {
            String message = messageSource.getMessage(
                    "ru.psu.core.account.web.WrongPassword", new
Object[]{}, LocaleContextHolder.getLocale());
            throw new ApplicationException(message);
        }
        if
(userChangePasswordRequest.getPassword().equals(userChangePasswo
rdRequest.getNewPassword())) {
            String message = messageSource.getMessage(
                    "ru.psu.core.account.web.SamePassword", new
Object[]{}, LocaleContextHolder.getLocale());
```

```
            throw new ApplicationException(message);
        }

        userService.changeUserPassword(user,
userChangePasswordRequest.getNewPassword());
    }

    @GetMapping(value = {"/me"})
    @ResponseBody
    public User getCurrentUserInfo(Principal principal)
            throws ApplicationException {
        User user;
        OAuth2Authentication authentication =
(OAuth2Authentication)principal;
        for(Object object:
authentication.getUserAuthentication().getAuthorities()) {
            SimpleGrantedAuthority authority =
(SimpleGrantedAuthority)object;
            if(authority.getAuthority().equals("ADMIN")) {
                user = new User();
                Company company = new Company();
                company.setId(0);
                company.setBlocked(false);
                company.setEntryName("adminCompany");
                company.setCompanyName("adminCompany");
                UserRole role = new UserRole();
                role.setTypeName("ADMIN");
                user.setId(0);
                user.setUserName(principal.getName());
                user.setPriority(0);
                user.setCanCreateConference(false);
                user.setDiskSpaceLimit(0);
                user.setLogin(principal.getName());
                user.setCompany(company);
                user.setCompanyIdFk(0);
                user.setUserRole(role);
                user.setRoleIdFk(3);
                user.setBlocked(false);
                user.setRegisteredAt(new Date());

                return user;
            }
        }
        user =
userRepository.findOneByLogin(principal.getName());
        if (user == null) {
            String message = messageSource.getMessage(
                    "ru.psu.core.account.web.UserNotExists",
                    new Object[]{principal.getName()},
LocaleContextHolder.getLocale());
```

```
            throw new ApplicationException(message);
        }

        return user;
    }

    @GetMapping(value = "/in")
    @ResponseBody
    public Page<User> getListUserByIds(@RequestParam(value =
"id") Integer[] ids,
                                       @RequestParam(value =
"search", required = false) String search,
                                       @PageableDefault(size =
Integer.MAX_VALUE, page = 0) Pageable pageable) {
        List<Specification<User>> spec =
SearchHelper.createSearchSpecificationList(search);
        return userRepository.findUsersByIdIn(ids, spec,
pageable);
    }

    @GetMapping(value = "/GroupsDepartments/in")
    @ResponseBody
    public Page<User>
getListUserOfGroupsAndDepartments(@RequestParam(value = "id")
Integer[] ids,

@RequestParam(value = "search", required = false) String search,

@PageableDefault(size = Integer.MAX_VALUE, page = 0) Pageable
pageable) {
        List<Specification<User>> spec =
SearchHelper.createSearchSpecificationList(search);
        return userRepository.getUsersOfGroupsOrCompanies(ids,
spec, pageable);
    }
}
```

Файл «UserRepository.java»

```
package ru.psu.core.account;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.EntityGraph;
import org.springframework.data.jpa.repository.JpaRepository;
import
org.springframework.data.jpa.repository.JpaSpecificationExecutor
;
import org.springframework.data.jpa.repository.Query;
```

```
import
org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import ru.psu.core.group.Group;

import java.util.ArrayList;

@Repository
public interface UserRepository extends JpaRepository<User,
Integer>, UserRepositoryCustom, JpaSpecificationExecutor<User> {
    @EntityGraph(value =
"user.company.userRole.groups.contacts.attributes", type =
EntityGraph.EntityGraphType.LOAD)
    User findOneById(Integer id);

    @EntityGraph(value =
"user.company.userRole.groups.contacts.attributes", type =
EntityGraph.EntityGraphType.LOAD)
    User findOneByLogin(String login);

    @EntityGraph(value =
"user.company.userRole.groups.contacts.attributes", type =
EntityGraph.EntityGraphType.LOAD)
    User findOneByLoginAndCompanyIdFk(String login, Integer
companyId);

    @EntityGraph(value =
"user.company.userRole.groups.contacts.attributes", type =
EntityGraph.EntityGraphType.LOAD)
    Page<User> findAll(Pageable pageable);

    @EntityGraph(value =
"user.company.userRole.groups.contacts.attributes", type =
EntityGraph.EntityGraphType.LOAD)
    Page<User> findAllByGroupsIsContaining(Group group, Pageable
pageable);

    @Query(value =
            "SELECT " +
                    "a.id, " +
                    "a.company_id, " +
                    "a.name, " +
                    "a.login, " +
                    "a.password_hash, " +
                    "a.avatar, " +
                    "a.avatar_origin, " +
                    "a.avatar_small, " +
                    "a.role_id, " +
                    "a.disk_space_limit, " +
```

```
                    "a.can_create_conference, " +
                    "a.priority, " +
                    "a.is_blocked, " +
                    "a.registered_at, " +
                    "a.last_login_at, " +
                    "a.updated_at, " +
                    "a.synced_at, " +
                    "a.description, " +
                    "a.is_active_directory " +
                    "FROM ab_user  a " +
                    "WHERE a.company_id IN (" +
                    "SELECT aa.id " +
                    "FROM ab_company aa " +
                    "WHERE aa.company_hierarchy <@
text2ltree(CONCAT(?#{#companyId}))" +
                    ") " +
                    "AND a.id NOT IN ( " +
                    "SELECT ua.id " +
                    "FROM ab_user ua INNER JOIN ab_user_group ug
" +
                    "ON ua.id = ug.user_id " +
                    "WHERE ug.group_id = ?#{#groupId} " +
                    ") " +
                    "ORDER BY ?#{#pageable} ",
            countQuery =
                    "SELECT COUNT(*) FROM ab_user a " +
                            "WHERE a.company_id IN (" +
                            "SELECT aa.id " +
                            "FROM ab_company aa " +
                            "WHERE aa.company_hierarchy <@
text2ltree(CONCAT(?#{#companyId}))" +
                            ") " +
                            "AND a.id NOT IN ( " +
                            "SELECT ua.id " +
                            "FROM ab_user ua INNER JOIN
ab_user_group ug " +
                            "ON ua.id = ug.user_id " +
                            "WHERE ug.group_id = ?#{#groupId} "
+
                            ") ",
            nativeQuery = true)
    Page<User> findAllOrganizationUsers(Pageable pageable,
                                        @Param("companyId")
Integer companyId,
                                        @Param("groupId")
Integer groupId);

    @Query(value ="SELECT " +
            "a.id, " +
            "a.company_id, " +
```

```
            "a.name, " +
            "a.login, " +
            "a.password_hash, " +
            "a.avatar, " +
            "a.avatar_origin, " +
            "a.avatar_small, " +
            "a.role_id, " +
            "a.disk_space_limit, " +
            "a.can_create_conference, " +
            "a.priority, " +
            "a.is_blocked, " +
            "a.registered_at, " +
            "a.last_login_at, " +
            "a.updated_at, " +
            "a.synced_at, " +
            "a.description, " +
            "a.is_active_directory " +
            "FROM ab_user  a " +
            "WHERE a.is_blocked = false AND (a.synced_at IS NULL
OR a.updated_at >= a.synced_at)",
            nativeQuery = true)
    ArrayList<User> findNotSyncedUsers();
}
```

Файл «UserServiceImpl.java»

```
package ru.psu.core.account;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
;
import org.springframework.stereotype.Service;
import org.springframework.Transaction.annotation.Transactional;
import ru.psu.client.PsuParticipantServiceClient;
import ru.psu.core.account.info.AttributeInfo;
import ru.psu.core.account.web.UserCreateRequest;
import ru.psu.core.account.web.UserFavoriteUpdateRequest;
import ru.psu.core.account.web.UserUpdateRequest;
import ru.psu.CommonStringFunctions;
import ru.psu.core.common.AvatarHelper;
import ru.psu.core.company.Company;
import ru.psu.core.group.Group;
import ru.psu.core.group.GroupRepository;
import ru.psu.core.threads.AsynchronousService;
import ru.psu.participant.ParticipantParametersView;
import ru.psu.property.AttachProperties;
import ru.psu.property.WebdavProperties;
```

```
import java.util.ArrayList;
import java.util.Date;

@Service
public class UserServiceImpl implements UserService {
    private final UserRepository userRepository;
    private final UserRoleRepository userRoleRepository;
    private final UserAttributeRepository
userAttributeRepository;
    private final UserAttributeTypeRepository
userAttributeTypeRepository;
    private final GroupRepository groupRepository;
    private final UserFavoriteRepository userFavoriteRepository;
    private final PsuParticipantServiceClient
PsuParticipantServiceClient;
    private final WebdavProperties webdavProperties;
    private final AttachProperties attachProperties;
    private final AvatarHelper avatarHelper;
    private final AsynchronousService asynchronousService;

    @Autowired
    public UserServiceImpl(UserRepository userRepository,
                           UserRoleRepository
userRoleRepository,
                           UserAttributeRepository
userAttributeRepository,
                           UserAttributeTypeRepository
userAttributeTypeRepository,
                           GroupRepository groupRepository,
                           UserFavoriteRepository
userFavoriteRepository,
                           PsuParticipantServiceClient
PsuParticipantServiceClient,
                           WebdavProperties webdavProperties,
                           AttachProperties attachProperties,
                           AvatarHelper avatarHelper,
                           AsynchronousService
asynchronousService) {
        this.userRepository = userRepository;
        this.userRoleRepository = userRoleRepository;
        this.userAttributeRepository = userAttributeRepository;
        this.userAttributeTypeRepository =
userAttributeTypeRepository;
        this.groupRepository = groupRepository;
        this.userFavoriteRepository = userFavoriteRepository;
        this.PsuParticipantServiceClient =
PsuParticipantServiceClient;
        this.webdavProperties = webdavProperties;
        this.attachProperties = attachProperties;
```

```
        this.avatarHelper = avatarHelper;
        this.asynchronousService = asynchronousService;
    }

    @Override
    @Transactional
    public User createUser(UserCreateRequest request, Company
company) {
        User user = new User();
        user.setCompanyIdFk(request.getCompanyId());
        user.setUserName(request.getName());
        user.setLogin(request.getLogin());
        user.setPasswordHash(new
BCryptPasswordEncoder().encode(request.getPassword()));

user.setUserRole(userRoleRepository.findOneByTypeName(request.ge
tAccountType()));
        user.setDiskSpaceLimit(request.getDiskSpaceLimit());

user.setCanCreateConference(request.getGrants().isCanCreateConfe
rence());
        if(request.getPriority() == null)
            user.setPriority(1);
        else
            user.setPriority(request.getPriority());
        user.setBlocked(request.isBlocked());
        if(request.getAvatarUri() != null &&
!request.getAvatarUri().isEmpty()){

if(avatarHelper.CheckAvatarNotExists(request.getAvatarUri()))
                return null;
            user.setAvatarOriginUri(request.getAvatarUri());
        }
        else
            user.setAvatarOriginUri("");
        avatarHelper.CreateThumbnails(user,
request.getAvatarUri());
        Date now = new Date();
        user.setLastUpdateDate(now);
        user.setRegisteredAt(now);
        user.setDescription(request.getDescription());
        user.setActiveDirectory(request.isActiveDirectory());

        // Если нужно, добавляем пользователя в группы
        if (!addUserToGroups(request.getGroups(), user))
            return null;
        userRepository.save(user);
        // Добавляем атрибуты пользователя
        ArrayList<UserAttribute> attrList =
createAttributes(request.getAttributes(), user.getId());
```

```
        if(attrList != null && attrList.size() != 0) {
            userAttributeRepository.save(attrList);
            user.getAttributes().addAll(attrList);
        }
        user.setCompany(company);

        userRepository.flush();
        asynchronousService.executeAsynchronously();
        return user;
    }

    @Override
    @Transactional
    public User updateUser(User user, UserUpdateRequest request)
{
        user.setCompanyIdFk(request.getCompanyId());
        user.setUserName(request.getName());
        if
(!user.getUserRole().getTypeName().equals(request.getAccountType
())) {

user.setUserRole(userRoleRepository.findOneByTypeName(request.ge
tAccountType()));
        }
        user.setDiskSpaceLimit(request.getDiskSpaceLimit());

user.setCanCreateConference(request.getGrants().isCanCreateConfe
rence());
        if(request.getPriority() == null)
            user.setPriority(1);
        else
            user.setPriority(request.getPriority());
        user.setBlocked(request.isBlocked());
        if(request.getAvatarSmallUri() == null ||
request.getAvatarSmallUri().isEmpty() ||
                request.getAvatarOriginUri() == null ||
request.getAvatarOriginUri().isEmpty()) {
            if (request.getAvatarUri() != null &&
!request.getAvatarUri().isEmpty()) {

if(avatarHelper.CheckAvatarNotExists(request.getAvatarUri()))
                    return null;
                user.setAvatarOriginUri(request.getAvatarUri());
            }
            else
                user.setAvatarOriginUri("");
            avatarHelper.CreateThumbnails(user,
request.getAvatarUri());
        }
        Date now = new Date();
```

```
        user.setLastUpdateDate(now);
        user.setDescription(request.getDescription());
        if(request.getPassword() != null &&
!request.getPassword().isEmpty())
            user.setPasswordHash(new
BCryptPasswordEncoder().encode(request.getPassword()));
        user.setActiveDirectory(request.isActiveDirectory());
        // Отчищаем список групп пользователя
        user.getGroups().clear();
        // Если нужно, добавляем пользователся в группы
        if (!addUserToGroups(request.getGroups(), user))
            return null;
        userRepository.save(user);

userAttributeRepository.deleteAllByUserIdFk(user.getId());
        // Добавляем атрибуты пользователя
        ArrayList<UserAttribute> newAttributeList =
createAttributes(request.getAttributes(), user.getId());
        if(newAttributeList != null && newAttributeList.size()
!= 0) {
            userAttributeRepository.save(newAttributeList);
            user.getAttributes().clear();
            user.getAttributes().addAll(newAttributeList);
        }
        userRepository.flush();
        asynchronousService.executeAsynchronously();
        return user;
    }

    @Override
    @Transactional
    public void deleteUser(User user) {

PsuParticipantServiceClient.deleteParticipant(CommonStringFuncti
ons.format(user.getId()));
        userRepository.delete(user);
    }

    @Override
    public Page<User> findAllOrganizationUsers(Pageable
pageable, Integer companyId, Integer groupId) {
        return userRepository.findAllOrganizationUsers(pageable,
companyId, groupId);
    }

    @Override
    public ArrayList<AttributeInfo> getUserAttributes(Integer
userId) {
        ArrayList<AttributeInfo> contacts = new ArrayList<>();
```

```
        ArrayList<UserAttribute> attributes =
userAttributeRepository.findAllByUserIdFk(userId);
        for(UserAttribute attribute : attributes){
            AttributeInfo attributeInfo = new
AttributeInfo(attribute.getUserAttributeType().getCode(),
                    attribute.getValue(),
attribute.getUserAttributeType().getName());
            contacts.add(attributeInfo);
        }

        return contacts;
    }

    @Override
    public ArrayList<Integer> getUserGroups(Integer userId) {
        User user = userRepository.findOneById(userId);
        ArrayList<Integer> groups = new ArrayList<>();

        for (Group group : user.getGroups()) {
            groups.add(group.getId());
        }

        return groups;
    }

    private ArrayList<UserAttribute>
createAttributes(ArrayList<AttributeInfo> attributes, Integer
userId) {
        ArrayList<UserAttribute> attributeList = new
ArrayList<>();
        if(attributes != null && !attributes.isEmpty()) {
            UserAttribute userAttribute;
            ArrayList<UserAttributeType> attributeTypes =
userAttributeTypeRepository.findAll();
            ArrayList<String> attributeTypeCodes = new
ArrayList<>();
            for (UserAttributeType attributeType :
attributeTypes) {
                attributeTypeCodes.add(attributeType.getCode());
            }
            for (AttributeInfo attribute : attributes) {
                if(attribute != null) {
                    int index =
attributeTypeCodes.indexOf(attribute.getCode());
                    //TODO: добавить проверку регулярок
                    if (index != -1) {
                        userAttribute = new UserAttribute();

userAttribute.setUserAttributeTypeIdFk(attributeTypes.get(index)
.getId());
```

```
userAttribute.setUserAttributeType(attributeTypes.get(index));

userAttribute.setValue(attribute.getValue());
                        userAttribute.setUserIdFk(userId);
                        attributeList.add(userAttribute);
                    }
                }
            }
        }
        return attributeList;
    }

    private boolean addUserToGroups(ArrayList<Integer> groups,
User user) {
        if (groups != null && groups.size() != 0) {
            Group group;
            for (Integer groupId : groups) {
                group = groupRepository.findOneById(groupId);
                if (group == null)
                    return false;
                user.getGroups().add(group);
            }
        }
        return true;
    }

    @Override
    public User updateUserFavorite(Integer userId, Integer
favoriteUserId,

UserFavoriteUpdateRequest request) {
        UserFavorite favorite = userFavoriteRepository
                .findOneByUserIdFkAndUserFavoriteIdFk(userId,
favoriteUserId);
        if (favorite == null)
            return null;

        if (request.getName() != null)
            favorite.setRewrittenName(request.getName());
        favorite.setPriority(request.getPriority());
        userFavoriteRepository.save(favorite);
        return favorite.getUserFavorite();
    }

    @Override
    public boolean deleteUserFavorite(Integer userId, Integer
favoriteUserId) {
        UserFavorite favorite = userFavoriteRepository
```

```
                .findOneByUserIdFkAndUserFavoriteIdFk(userId,
favoriteUserId);

        if (favorite == null)
            return false;

        userFavoriteRepository.delete(favorite);
        return true;
    }

    @Override
    public Page<UserFavorite> findAllUserFavorites(Pageable
pageable, Integer userId) {
        return
userFavoriteRepository.findAllByUserIdFk(pageable, userId);
    }

    @Override
    public boolean createUserFavorite(Integer userId, Integer
favoriteUserId) {
        if (userRepository.findOneById(userId) == null)
            return false;

        if
(userFavoriteRepository.findOneByUserIdFkAndUserFavoriteIdFk(use
rId, favoriteUserId) != null)
            return true;

        UserFavorite favorite = new UserFavorite();
        favorite.setPriority(1);
        favorite.setUserIdFk(userId);
        favorite.setUserFavoriteIdFk(favoriteUserId);
        userFavoriteRepository.save(favorite);
        return true;
    }

    @Override
    public Page<User> getAllUsers(Pageable pageable){
        return userRepository.findAll(pageable);
    }

    @Override
    public void changeUserPassword(User user, String password) {
        user.setPasswordHash(new
BCryptPasswordEncoder().encode(password));
        userRepository.save(user);
    }
```

```
    /*private void CreateUpdatePsuUser(String login, String
name, String avatarUri, Integer diskSpaceLimit, Integer userId)
{
        ParticipantParametersView participantParameters = new
ParticipantParametersView();
        if(avatarUri == null || avatarUri.isEmpty()) {
            Integer colorNumber =
CommonStringFunctions.getColorNumber(login,
                    attachProperties.getColorsArray().length);
            String color =
attachProperties.getColorsArray()[colorNumber];

participantParameters.setAvatar(attachProperties.getHost() +
CommonStringFunctions.getDefaultAvatarUrl(color, name,
attachProperties.getUrl(), 512));
        }
        else

participantParameters.setAvatar(webdavProperties.getExternalUrl(
) + avatarUri);
        participantParameters.setName(name);
        participantParameters.setDiskQuotaMb(diskSpaceLimit);
        participantParameters.setType("Psu");
        String GUID = CommonStringFunctions.format(userId);
        PsuParticipantServiceClient.createParticipant(GUID,
participantParameters);
    }*/
}
```

Приложение В. Листинг SQL-файла

```
CREATE EXTENSION IF NOT EXISTS ltree;

CREATE TABLE ab_company (
     id INTEGER ,
     name VARCHAR(255) NOT NULL,
     entry_name VARCHAR(255),
     description VARCHAR(512),
     company_hierarchy LTREE,
     is_blocked BOOLEAN DEFAULT false NOT NULL,
     updated_at TIMESTAMP,
     synced_at TIMESTAMP,
     registered_at TIMESTAMP NOT NULL,
     is_active_directory BOOLEAN DEFAULT false NOT NULL,
     user_count INTEGER DEFAULT 0 NOT NULL,
     avatar character varying(255),
     avatar_origin character varying(255),
     avatar_small character varying(255),

     CONSTRAINT pk_ab_company PRIMARY KEY (id)
);
COMMENT ON TABLE ab_company IS 'Организации';
COMMENT ON COLUMN ab_company.id IS 'Идентификатор организации';
COMMENT ON COLUMN ab_company.name IS 'Наименование организации';
COMMENT ON COLUMN ab_company.entry_name IS 'Котроткое
наименование организации для входа';
COMMENT ON COLUMN ab_company.company_hierarchy IS 'Иерархия
организации';
COMMENT ON COLUMN ab_company.description IS 'Описание';
COMMENT ON COLUMN ab_company.is_blocked IS 'Признак блокировки
организации';
COMMENT ON COLUMN ab_company.updated_at IS 'Дата последнего
обновления';
COMMENT ON COLUMN ab_company.synced_at IS 'Дата последней
синхронизации';
COMMENT ON COLUMN ab_company.registered_at IS 'Дата создания
организации';
COMMENT ON COLUMN ab_company.is_active_directory IS 'Признак
выгрузки данных организации из Active Directory';
COMMENT ON COLUMN ab_company.user_count IS 'Количество
пользователей организации/департамента';
COMMENT ON COLUMN ab_company.avatar IS 'Аватар';
COMMENT ON COLUMN ab_company.avatar_origin IS 'Изначальный
файл';
COMMENT ON COLUMN ab_company.avatar_small IS 'Аватар (маленький
файл)';
```

```
CREATE TABLE ab_company_attribute_type (
     id INTEGER,
     name VARCHAR(255) NOT NULL,
     code VARCHAR(255),
     description VARCHAR(512),
     reg_exp VARCHAR(4096),

     CONSTRAINT pk_ab_company_attribute_type PRIMARY KEY (id),
     CONSTRAINT unq_ab_company_attribute_type_code UNIQUE (code)
);
COMMENT ON TABLE ab_company_attribute_type IS 'Перечень
атрибутов для оргпнизации';
COMMENT ON COLUMN ab_company_attribute_type.id IS 'Идентификатр
атрибута';
COMMENT ON COLUMN ab_company_attribute_type.name IS
'Наименование атрибта';
COMMENT ON COLUMN ab_company_attribute_type.code IS
'Наименование поля в JSON';
COMMENT ON COLUMN ab_company_attribute_type.description IS
'Описание поля';
COMMENT ON COLUMN ab_company_attribute_type.reg_exp IS
'Регулярное выражение для проверки валидности';


CREATE TABLE ab_company_attribute (
     id INTEGER,
     company_id INTEGER NOT NULL,
     attribute_type_id INTEGER NOT NULL,
     value VARCHAR NOT NULL,
     parent_company_attribute_id INTEGER,

     CONSTRAINT pk_ab_company_attribute PRIMARY KEY (id),

     CONSTRAINT fk_ab_company_attribute_ab_company_attribute_type
FOREIGN KEY (attribute_type_id)
          REFERENCES ab_company_attribute_type (id)
          ON UPDATE NO ACTION ON DELETE CASCADE,

     CONSTRAINT fk_ab_company_attribute_ab_company FOREIGN KEY
(company_id)
          REFERENCES ab_company (id)
          ON UPDATE NO ACTION ON DELETE CASCADE,

     CONSTRAINT fk_ab_company_attribute_ab_company_attribute
FOREIGN KEY (parent_company_attribute_id)
          REFERENCES ab_company_attribute (id)
          ON UPDATE NO ACTION ON DELETE NO ACTION
);

COMMENT ON TABLE ab_company_attribute IS 'Атрибуты организации';
```

```
COMMENT ON COLUMN ab_company_attribute.id IS 'Идентификатор
атрибута организации';
COMMENT ON COLUMN ab_company_attribute.company_id IS
'Идентификатор организации';
COMMENT ON COLUMN ab_company_attribute.attribute_type_id IS
'Идентификатор атрибута';
COMMENT ON COLUMN ab_company_attribute.value IS 'Значение
атрибута';
COMMENT ON COLUMN
ab_company_attribute.parent_company_attribute_id IS
'Идентификатор родительского атрибута компании';


CREATE TABLE ab_user_role (
     id SERIAL,
     name VARCHAR(255) NOT NULL,

     CONSTRAINT pk_ab_user_role PRIMARY KEY (id),
     CONSTRAINT unq_ab_user_role UNIQUE (name)
);
COMMENT ON TABLE ab_user_role IS 'Роли пользователей в системе';
COMMENT ON COLUMN ab_user_role.id IS 'Идентификатор роли';
COMMENT ON COLUMN ab_user_role.name IS 'Наименование роли
пользователя';


CREATE TABLE ab_user_grant (
     id SERIAL,
     name VARCHAR(255) NOT NULL,

     CONSTRAINT pk_ab_user_grant PRIMARY KEY (id),
     CONSTRAINT unq_ab_user_grant UNIQUE (name)
);
COMMENT ON TABLE ab_user_grant IS 'Права пользователей в
системе';
COMMENT ON COLUMN ab_user_grant.id IS 'Идентификатор прав';
COMMENT ON COLUMN ab_user_grant.name IS 'Наименование прав
пользователя';


CREATE TABLE ab_user_role_grant (
     id SERIAL,
     role_id INTEGER NOT NULL,
     grant_id INTEGER NOT NULL,

     CONSTRAINT pk_ab_user_role_grant PRIMARY KEY (id),

     CONSTRAINT fk_ab_user_role_grant_ab_user_role FOREIGN KEY
(role_id)
          REFERENCES ab_user_role (id)
```

```
          ON UPDATE NO ACTION ON DELETE NO ACTION,

     CONSTRAINT fk_ab_user_role_grant_ab_user_grant FOREIGN KEY
(grant_id)
          REFERENCES ab_user_grant (id)
          ON UPDATE NO ACTION ON DELETE NO ACTION,

     CONSTRAINT unq_ab_user_role_grant UNIQUE (role_id, grant_id)
);
COMMENT ON TABLE ab_user_role_grant IS 'Права роли пользователя
в системе';
COMMENT ON COLUMN ab_user_role_grant.id IS 'Идентификатор';
COMMENT ON COLUMN ab_user_role_grant.role_id IS 'Идентификатор
роли';
COMMENT ON COLUMN ab_user_role_grant.grant_id IS 'Идентификатор
прав';


CREATE TABLE ab_group (
     id INTEGER,
     name VARCHAR(255) NOT NULL,
     company_id INTEGER NOT NULL,
     description VARCHAR(512),
     avatar character varying(255),
     avatar_origin character varying(255),
     avatar_small character varying(255),
     created_at TIMESTAMP NOT NULL,
     user_count INTEGER DEFAULT 0 NOT NULL,

     CONSTRAINT pk_ab_group PRIMARY KEY (id),

     CONSTRAINT fk_ab_group_ab_company FOREIGN KEY (company_id)
          REFERENCES ab_company (id)
          ON UPDATE NO ACTION ON DELETE CASCADE,

     CONSTRAINT unq_ab_group UNIQUE (company_id, name)
);
COMMENT ON TABLE ab_group IS 'Группы рассылки';
COMMENT ON COLUMN ab_group.id IS 'Идентификатор группы';
COMMENT ON COLUMN ab_group.name IS 'Наименование группы';
COMMENT ON COLUMN ab_group.company_id IS 'Идентификатор
компании';
COMMENT ON COLUMN ab_group.description IS 'Описание';
COMMENT ON COLUMN ab_group.avatar IS 'Аватар';
COMMENT ON COLUMN ab_group.avatar_origin IS 'Изначальный файл';
COMMENT ON COLUMN ab_group.avatar_small IS 'Аватар (маленький
файл)';
COMMENT ON COLUMN ab_group.created_at IS 'Дата создания группы';
COMMENT ON COLUMN ab_group.user_count IS 'Количество
пользователей группы';
```

```
CREATE TABLE ab_user (
     id INTEGER,
     company_id INTEGER NOT NULL,
     name VARCHAR(255) NOT NULL,
     login VARCHAR(255) NOT NULL,
     password_hash VARCHAR(255) NOT NULL,
     avatar VARCHAR(255) ,
     avatar_origin VARCHAR(255),
     avatar_small VARCHAR(255),
     role_id INTEGER NOT NULL,
     disk_space_limit INTEGER NOT NULL,
     can_create_conference BOOLEAN NOT NULL,
     priority INTEGER DEFAULT 1 NOT NULL,
     is_blocked BOOLEAN DEFAULT false NOT NULL,
     registered_at TIMESTAMP NOT NULL,
     last_login_at TIMESTAMP,
     updated_at TIMESTAMP,
     synced_at TIMESTAMP,

     CONSTRAINT pk_ab_user PRIMARY KEY (id),

     CONSTRAINT fk_ab_user_ab_company FOREIGN KEY (company_id)
          REFERENCES ab_company (id)
          ON UPDATE NO ACTION ON DELETE CASCADE,

     CONSTRAINT fk_ab_user_ab_user_role FOREIGN KEY (role_id)
          REFERENCES ab_user_role (id)
          ON UPDATE NO ACTION ON DELETE NO ACTION,

     CONSTRAINT unq_ab_user UNIQUE (login, company_id)
);

COMMENT ON TABLE ab_user IS 'Пользователи и контакты системы';
COMMENT ON COLUMN ab_user.id IS 'Идентификатор пользователя';
COMMENT ON COLUMN ab_user.company_id IS 'Идентификатор
организации';
COMMENT ON COLUMN ab_user.name IS 'Имя пользователя';
COMMENT ON COLUMN ab_user.login IS 'Логин для входа в систему
(login)';
COMMENT ON COLUMN ab_user.password_hash IS 'Пароль для входа в
систему';
COMMENT ON COLUMN ab_user.avatar IS 'Аватар';
COMMENT ON COLUMN ab_user.avatar_origin IS 'Изначальный файл';
COMMENT ON COLUMN ab_user.avatar_small IS 'Аватар (маленький
файл)';
COMMENT ON COLUMN ab_user.role_id IS 'Роль пользователя';
COMMENT ON COLUMN ab_user.disk_space_limit IS 'Доступное
дисковое пространство';
```

```
COMMENT ON COLUMN ab_user.can_create_conference IS 'Возможность
создания конференции';
COMMENT ON COLUMN ab_user.priority IS 'Приоритет пользователей
(в рамках организации)';
COMMENT ON COLUMN ab_user.is_blocked IS 'Состояние
заблокированности пользователя';
COMMENT ON COLUMN ab_user.registered_at IS 'Дата регистрации
пользователя';
COMMENT ON COLUMN ab_user.last_login_at IS 'Дата последнего
входа';
COMMENT ON COLUMN ab_user.updated_at IS 'Дата последнего
изменения данных пользователя';
COMMENT ON COLUMN ab_user.synced_at IS 'Дата последней
синхронизации данных';


CREATE TABLE ab_user_group (
     id SERIAL,
     user_id INTEGER NOT NULL,
     group_id INTEGER NOT NULL,

     CONSTRAINT pk_ab_user_group PRIMARY KEY (id),

     CONSTRAINT fk_ab_user_group_ab_group FOREIGN KEY (group_id)
          REFERENCES ab_group (id)
          ON UPDATE NO ACTION ON DELETE CASCADE,

     CONSTRAINT fk_ab_user_group_ab_user FOREIGN KEY (user_id)
          REFERENCES ab_user (id)
          ON UPDATE NO ACTION ON DELETE CASCADE,

     CONSTRAINT unique_ab_user_group UNIQUE(user_id, group_id)
);
COMMENT ON TABLE ab_user_group IS 'Группы пользователя';
COMMENT ON COLUMN ab_user_group.user_id IS 'Идентификатор группы
пользователя';
COMMENT ON COLUMN ab_user_group.user_id IS 'Идентификатор
пользователя';
COMMENT ON COLUMN ab_user_group.group_id IS 'Идентификатор
группы';


CREATE TABLE ab_user_attribute_type (
     id INTEGER,
     name VARCHAR(255) NOT NULL,
     code VARCHAR(255),
     description VARCHAR(512),
     reg_exp VARCHAR(4096),

     CONSTRAINT pk_ab_user_attribute_type PRIMARY KEY (id)
```

```
);
COMMENT ON TABLE ab_user_attribute_type IS 'Перечень атрибутов
для пользователей';
COMMENT ON COLUMN ab_user_attribute_type.id IS 'Идентификатр
атрибута';
COMMENT ON COLUMN ab_user_attribute_type.name IS 'Наименование
атрибта';
COMMENT ON COLUMN ab_user_attribute_type.code IS 'Наименование
поля в JSON';
COMMENT ON COLUMN ab_user_attribute_type.description IS
'Описание поля';
COMMENT ON COLUMN ab_user_attribute_type.reg_exp IS 'Регулярное
выражение для проверки валидности';


CREATE TABLE ab_user_attribute (
     id INTEGER,
     user_id INTEGER NOT NULL,
     attribute_type_id INTEGER NOT NULL,
     value VARCHAR NOT NULL,
     parent_user_attribute_id INTEGER,

     CONSTRAINT pk_ab_user_attribute PRIMARY KEY (id),

     CONSTRAINT fk_ab_user_attribute_ab_user FOREIGN KEY
(user_id)
          REFERENCES ab_user (id)
          ON UPDATE NO ACTION ON DELETE CASCADE,

     CONSTRAINT fk_ab_user_attribute_ab_user_attribute_type
FOREIGN KEY (attribute_type_id)
          REFERENCES ab_user_attribute_type (id)
          ON UPDATE NO ACTION ON DELETE CASCADE,

     CONSTRAINT fk_ab_user_attribute_self FOREIGN KEY
(parent_user_attribute_id)
          REFERENCES ab_user_attribute (id)
          ON UPDATE NO ACTION ON DELETE NO ACTION
);
COMMENT ON TABLE ab_user_attribute IS 'Атрибуты пользователя';
COMMENT ON COLUMN ab_user_attribute.id IS 'Идентификатор
атрибута пользователя';
COMMENT ON COLUMN ab_user_attribute.user_id IS 'Идентификатор
пользователя';
COMMENT ON COLUMN ab_user_attribute.attribute_type_id IS
'Идентификатор атрибута';
COMMENT ON COLUMN ab_user_attribute.value IS 'Значение
атрибута';
COMMENT ON COLUMN ab_user_attribute.parent_user_attribute_id IS
'Идентификатор родительского атрибута (для сложных объектов)';
```

```
CREATE TABLE ab_user_favorite (
     id INTEGER,
     user_id INTEGER NOT NULL,
     user_favorite_id INTEGER NOT NULL,
     rewritten_name VARCHAR(255),
     priority INTEGER,

     CONSTRAINT pk_ab_user_favorite PRIMARY KEY (id),

     CONSTRAINT fk_ab_user_favorite_ab_user FOREIGN KEY (user_id)
          REFERENCES ab_user (id)
          ON UPDATE NO ACTION ON DELETE CASCADE,

     CONSTRAINT fk_ab_user_favorite_ab_user_favorite FOREIGN KEY
(user_favorite_id)
          REFERENCES ab_user (id)
          ON UPDATE NO ACTION ON DELETE CASCADE
);
COMMENT ON TABLE ab_user_favorite IS 'Адресеая книга
пользователя';
COMMENT ON COLUMN ab_user_favorite.id IS 'Идентификатор
контакта';
COMMENT ON COLUMN ab_user_favorite.user_id IS 'Идентификатор
пользователя';
COMMENT ON COLUMN ab_user_favorite.user_favorite_id IS
'Идентификатор контакта';
COMMENT ON COLUMN ab_user_favorite.rewritten_name IS
'Переопределенное пользователем имя контакта';
COMMENT ON COLUMN ab_user_favorite.priority IS 'Приоритет
контакта';



CREATE TABLE ab_contact (
     id INTEGER,
     name VARCHAR(255) NOT NULL,
     avatar VARCHAR(255) ,
     avatar_origin VARCHAR(255),
     avatar_small VARCHAR(255),
     priority INTEGER DEFAULT 1 NOT NULL,
     registered_at TIMESTAMP NOT NULL,
     updated_at TIMESTAMP,
     user_id INTEGER,

     CONSTRAINT pk_ab_contact PRIMARY KEY (id),

     CONSTRAINT fk_ab_contact_ab_user FOREIGN KEY (user_id)
          REFERENCES ab_user (id)
```

```
          ON UPDATE NO ACTION ON DELETE CASCADE
);
COMMENT ON TABLE ab_contact IS 'Контакты пользователя';
COMMENT ON COLUMN ab_contact.id IS 'Идентификатор контакта
пользователя';
COMMENT ON COLUMN ab_contact.avatar IS 'Аватар';
COMMENT ON COLUMN ab_contact.avatar_origin IS 'Изначальный
файл';
COMMENT ON COLUMN ab_contact.avatar_small IS 'Аватар (маленький
файл)';
COMMENT ON COLUMN ab_contact.priority IS 'Приоритет отображения
контакта';;
COMMENT ON COLUMN ab_contact.registered_at IS 'Дата регистрации
контакта';
COMMENT ON COLUMN ab_contact.updated_at IS 'ата последнего
изменения данных пользователя';
COMMENT ON COLUMN ab_contact.user_id IS 'Идентификатор
пользователя - владельца контакта';


CREATE TABLE ab_contact_attribute (
     id INTEGER,
     contact_id INTEGER NOT NULL,
     attribute_type_id INTEGER NOT NULL,
     value VARCHAR NOT NULL,
     parent_contact_attribute_id INTEGER,

     CONSTRAINT pk_ab_contact_attribute PRIMARY KEY (id),

     CONSTRAINT fk_ab_contact_attribute_ab_user_attribute_type
FOREIGN KEY (attribute_type_id)
          REFERENCES ab_user_attribute_type (id)
          ON UPDATE NO ACTION ON DELETE CASCADE,

     CONSTRAINT fk_ab_contact_attribute_ab_contact FOREIGN KEY
(contact_id)
          REFERENCES ab_contact (id)
          ON UPDATE NO ACTION ON DELETE CASCADE,

     CONSTRAINT fk_ab_contact_attribute_self FOREIGN KEY
(parent_contact_attribute_id)
          REFERENCES ab_contact_attribute (id)
          ON UPDATE NO ACTION ON DELETE NO ACTION
);
COMMENT ON TABLE ab_contact_attribute IS 'Атрибуты контактов';
COMMENT ON COLUMN ab_contact_attribute.id IS 'Идентификатор
атрибута контакта';
COMMENT ON COLUMN ab_contact_attribute.contact_id IS
'Идентификатор контакта';
```

```
COMMENT ON COLUMN ab_contact_attribute.attribute_type_id IS
'Идентификатор атрибута';
COMMENT ON COLUMN ab_contact_attribute.value IS 'Значение
атрибута';
COMMENT ON COLUMN
ab_contact_attribute.parent_contact_attribute_id IS
'Идентификатор родительского атрибута контакта';


CREATE SEQUENCE hibernate_ab_contact_attribute_id_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;

 CREATE SEQUENCE hibernate_ab_contact_id_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;

CREATE SEQUENCE hibernate_ab_user_attribute_id_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;

 CREATE SEQUENCE hibernate_ab_user_id_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;

 CREATE SEQUENCE hibernate_ab_company_group_id_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;

 CREATE SEQUENCE hibernate_ab_user_favorite_id_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;
```

```
 CREATE SEQUENCE hibernate_ab_user_attribute_type_id_seq
 INCREMENT 1
 MINVALUE 1
 MAXVALUE 9223372036854775807
 START 1
 CACHE 1;

--CREATE SEQUENCE hibernate_ab_group_id_seq
--  INCREMENT 1
--  MINVALUE 1
--  MAXVALUE 9223372036854775807
--  START 1
--  CACHE 1;

 CREATE SEQUENCE hibernate_ab_company_attribute_id_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;

 CREATE SEQUENCE hibernate_ab_company_attribute_type_id_seq
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;
```

Приложение С. Описание REST API для работы с пользователями

Создание пользователя.

Метод: *POST*
URI:*/addressbook/user/create*
Тело запроса:

```
{
  "name": " string",
  "login": " string",
  "password": " string",
  "companyId": int ,
  "accountType": "string",
  "diskSpaceLimit": int,
  "priority": int,
  "blocked": "boolean",
  "attributes": [
           {"#attributeType#": "string"}
         ],
  "grants": {
    "canCreateConference": "boolean"
       },
  "avatarUri": "string"
}
```

Ответ:

200 OK

```
{
 "id": "string",
 "name": " string",
 "login": " string"
 "companyId": " string" ,
 "companyName": " string" ,
 "accountType": "string",
 "blocked": boolean,
 "priority": int,
 "diskSpaceLimit": int,
 "avatarUri": "string",
 "avatarOriginUri": "string",
 "avatarSmallUri": "string",
 "registeredAt":"int",
 "groups": [ int ],
```

```
            "attributes": [
              {"#attributeType#": "string"}
            ],
            "grants": {
                    "canCreateConference": boolean
                  }
          }
    400   Bad Request
          { "errorMessage": "string" }
    401   Not authorized
          { "errorMessage": "string" }
    403   Forbidden
          { "errorMessage": "string" }
    500   Server error
          { "errorMessage": "string" }
```

Обновление пользователя

```
Метод: POST
URI:/addressbook/user/{userId}
Тело запроса:
         {
          "name": " string",
          "companyId": int ,
          "accountType": "string",
          "diskSpaceLimit": int,
          "priority": int,
          "blocked": "boolean",
          "attributes": [
                   {"#attributeType#": "string"}
                ],
          "grants": {
            "canCreateConference": "boolean"
                },
           "avatarUri": "string"
         }
Ответ:
    200   OK
            {
             "id": "string",
             "name": " string",
             "login": " string"
```

```
            "companyId": " string" ,
            "companyName": " string" ,
            "accountType": "string",
            "blocked": boolean,
            "priority": int,
            "diskSpaceLimit": int,
            "avatarUri": "string",
            "avatarOriginUri": "string",
            "avatarSmallUri": "string",
            "registeredAt":"int",
            "groups": [ int ],
            "attributes": [
              {"#attributeType#": "string"}
            ],
            "grants": {
                   "canCreateConference": boolean
                 }
          }
```

400 Bad Request

```
{ "errorMessage": "string" }
```

401 Not authorized

```
{ "errorMessage": "string" }
```

403 Forbidden

```
{ "errorMessage": "string" }
```

500 Server error

```
{ "errorMessage": "string" }
```

Смена пароля пользователя

Метод: POST

URI: */addressbook*/user/{userId}/changepass

Тело запроса:

```
{
  "password": " string",
  "newPassword": " string"
}
```

Ответ:

200 OK

400 Bad Request

```
{ "errorMessage": "string" }
```

401 Not authorized

```
{ "errorMessage": "string" }
```

403 Forbidden

```
          { "errorMessage": "string" }
     500  Server error
          { "errorMessage": "string" }
```

Удаление пользователя

```
Метод: DELETE
URI:/addressbook/user/{userId}
Тело запроса:        -
Ответ:
     200  OK
     400  Bad Request
          { "errorMessage": "string" }
     401  Not authorized
          { "errorMessage": "string" }
     403  Forbidden
          { "errorMessage": "string" }
     500  Server error
          { "errorMessage": "string" }
```

Получение пользователя по ID

```
Метод: GET
URI:/addressbook/user/{userId}
Тело запроса:
Ответ:
     200  OK
            {
             "id": "string",
             "name": " string",
             "login": " string"
             "companyId": " string" ,
             "companyName": " string" ,
             "accountType": "string",
             "blocked": boolean,
             "priority": int,
             "diskSpaceLimit": int,
             "avatarUri": "string",
             "avatarOriginUri": "string",
             "avatarSmallUri": "string",
             "registeredAt":"int",
             "groups": [ int ],
             "attributes": [
               {"#attributeType#": "string"}
```

```
            ],
            "grants": {
                    "canCreateConference": boolean
                  }
           }
     400   Bad Request
           { "errorMessage": "string" }
     401   Not authorized
           { "errorMessage": "string" }
     403   Forbidden
           { "errorMessage": "string" }
     500   Server error
           { "errorMessage": "string" }
```

Получение авторизованного пользователя

```
Метод: DELETE
URI:/addressbook/user/me
Тело запроса:        -
Ответ:
     200   OK
           {
            "id": "string",
            "name": " string",
            "login": " string"
            "companyId": " string" ,
            "companyName": " string" ,
            "accountType": "string",
            "blocked": boolean,
            "priority": int,
            "diskSpaceLimit": int,
            "avatarUri": "string",
            "avatarOriginUri": "string",
            "avatarSmallUri": "string",
            "registeredAt":"int",
            "groups": [ int ],
            "attributes": [
              {"#attributeType#": "string"}
            ],
            "grants": {
                    "canCreateConference": boolean
                  }
           }
```

```
400   Bad Request
      { "errorMessage": "string" }
401   Not authorized
      { "errorMessage": "string" }
403   Forbidden
      { "errorMessage": "string" }
500   Server error
      { "errorMessage": "string" }
```

Приложение D. Презентация

Микросервис «Адресная книга» для интернет-мессенджера

Автор работы: студентка группы 14ВВ1 Пучкова У.Н.
Научный руководитель: д.т.н., профессор Пащенко Д.В.

Цель и объект работы

Цель работы: разработать микросервис «Адресная книга».

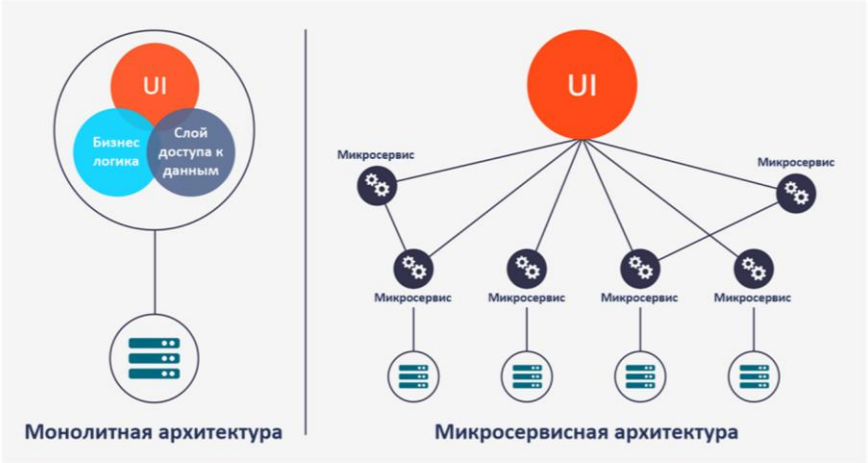
Объектом разработки является приложение, обеспечивающее выполнение следующих функций:

- создание;
- удаление;
- редактирование пользователей, организаций, групп.

Хранение информации о пользователях, организациях и группах должно осуществляться в БД. Также необходимо реализовать API для взаимодействия с остальными микросервисами.

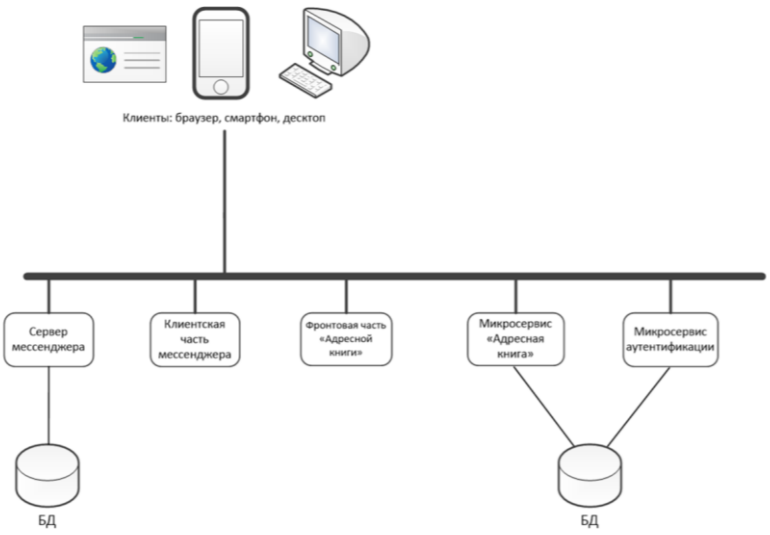
2

Сравнение монолитной и микросервисной архитектур



3

Структура мессенджера



4

Стек технологий

4

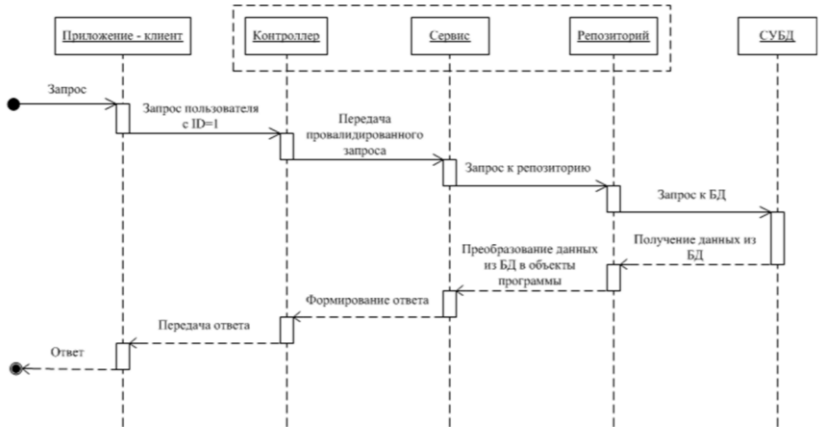
Структура БД «Адресной книги»

6

Диаграмма класса-репозитория «Адресной книги»

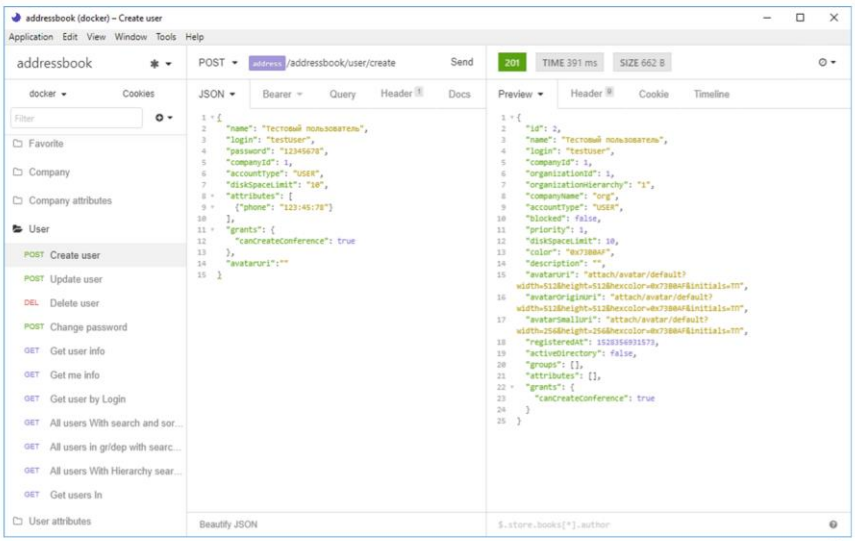
7

Последовательность взаимодействия классов при обработке запроса «Получение пользователя по ID»



8

Результат выполнения функционального тестирования разработанного микросервиса



9

Результаты работы

- разработана структура базы данных для СУБД *PostgreSQL*
- спроектирована архитектура микросервиса «Адресная книга»
- разработан прикладной интерфейс для взаимодействия с микросервисами, входящими в состав интернет-мессенджера, согласно архитектурному стилю *REST API*
- осуществлено функциональное тестирование микросервиса «Адресная книга»

10

Благодарю за внимание!