

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Вычислительные машины, комплексы, системы и сети»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
на тему
ПРЕПРОЦЕССОР ВАЛИДАЦИИ И
КОРРЕКТИРОВКИ ВХОДНЫХ ДАННЫХ
ДЛЯ ВИРТУАЛЬНОГО АССИСТЕНТА

Студент ДВ - 31.05.2023 Нешко Дарья Олеговна
(подпись, дата) (ФИО полностью)

Руководитель 31.05.2023 Синев М.П.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр 31.05.2023 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент зав. каф. Программ. Инж. 02.06.23 Мартышкин А.И.
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 02.06.2023 г № 14)

Заведующий кафедрой М. Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отметка протокол заседания ГЭК от

9.06.2023 № 2)

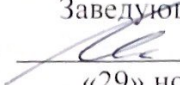
Секретарь ГЭК Е. Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«29» ноября 2021г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Нешко Дарья Олеговна гр.21BVBm1 факультета ВТ
2. Тема работы «Препроцессор валидации и корректировки входных данных для виртуального ассистента»

Тема утверждена приказом ПГУ №924/о от "29" ноября 2021г.
3. Руководитель магистерской диссертации работы к.т.н, доцент кафедры «ВТ» Синев М.П.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Проектирование и разработка препроцессора для валидации и корректировки входных данных для виртуального ассистента для дистанционной подачи заявлений на зачисление в образовательные организации, позволяющего автоматизировать процесс приема заявлений на зачисление. Приложение должно предоставлять следующие возможности: предоставлять сведения об образовательных организациях и специальностях, реализовать быстрый поиск данных, доступных для подачи заявления, функционал подачи заявления, функционал поиска заявлений по данным абитуриента. Необходимо реализовать валидацию пользовательских данных и их корректировку для исключения ошибок, связанных с человеческим фактором.

5. Перечень подлежащих разработке вопросов

Требуется провести анализ предметной области: изучить способы реализации быстрого поиска, рассмотреть основные системы управления базами данных и типы архитектур для построения приложения. Изучить вспомогательные технологии, используемые для разработки, такие как Spring Framework, Hibernate, Flyway, Elasticsearch, Swagger и т.д.

6. Календарный график выполнения работы

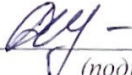
№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	11%	29.11.2021- 29.01.2022	
2	Анализ требований	11%	30.01.2022- 30.03.2022	
3	Выбор технологий, применяемых при проектировании и разработке препроцессора валидации	11%	31.03.2022- 31.05.2022	
4	Проектирование и реализация базы данных	16%	01.06.2022- 01.09.2022	
5	Оптимизация поисковых запросов	6%	02.09.2022- 02.10.2022	
6	Разработка функционала приложения	33%	03.10.2022- 31.03.2023	
7	Тестирование и отладка	6%	01.04.2023- 29.04.2023	
8	Оформление пояснительной записки	6%	30.04.2023- 31.05.2023	

Дата выдачи задания "29" ноября 2021г.

Руководитель магистерской диссертации

 31.05.2023
(подпись, дата)

Задание к исполнению принял студент

 -
(подпись, дата)

Работу к защите допустить
Декан факультета

 И.Р. Рязанов 05.06.2023
(подпись, дата)

РЕФЕРАТ

Пояснительная записка содержит 152 листа, 59 рисунка, 11 таблиц, 10 источников, 2 приложения.

ПРЕПРОЦЕССОР ВАЛИДАЦИИ И КОРРЕКТИРОВКИ ВХОДНЫХ ДАННЫХ ДЛЯ ВИРТУАЛЬНОГО АССИСТЕНТА, SPRING FRAMEWORK, JACKSON FRAMEWORK, INTELLIJ IDEA, HIBERNATE, FLYWAY, SPRING DOC OPENAPI, POSTGRESQL, PGADMIN4, JAVA, ELASTICSEARCH.

Целью проекта является проектирование, разработка и анализ работы сервиса виртуального ассистента для дистанционной подачи заявлений на зачисление в образовательные организации с препроцессором валидации и корректировки входных данных. Система должна обеспечивать высокопроизводительный процесс получения, обработки и корректировки входных данных в режиме реального времени. Система должна иметь задокументированный программный интерфейс для дальнейшей возможной интеграции с ней.

В результате проделанной работы разработана система, удовлетворяющая поставленной задаче: было реализовано и протестировано сервис-ассистент для дистанционной подачи заявлений на зачисление в образовательные организации с валидацией и корректировкой входных данных. Также был задокументирован программный интерфейс приложения.

Для разработки использовался широкий стек технологий, в списке которых язык программирования Java, Spring Framework, Jackson Framework, Hibernate, Flyway, Spring Doc OpenAPI, ElasticSearch. Для хранения данных выбрана система управления базами данных PostgreSQL, а также инструмент администрирования pgAdmin4.

					ПГУ 1.090401.08.001 ПЗ			
Изм.	Лист	№ докум	Подпись	Дата	Препроцессор валидации и корректировки входных данных для виртуального ассистента Пояснительная записка	Лит.	Лист	Листов
Разраб.		Нешко Д.О.		31.05.23				
Провер.		Синев М.П.		31.05.23			2	152
Н. Контр.		Бычков А.С.				ФВТ, каф.ВТ гр. 21ВВВм1		
Утв.		Митрохин М.А.						

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ	6
1 Анализ предметной области	8
1.1 Архитектура приложения.....	8
1.2 Способы быстрого поиска данных	10
1.3 Функциональные блоки сервиса.....	13
1.4 Документирование API.....	14
1.5 Выводы.....	15
2 Реализация.....	17
2.1 стек технологий, используемый для разработки	17
2.2 Требования к разработке	18
2.3 Алгоритм работы сервиса	19
2.4 Описание структуры и таблиц БД	22
2.5 Описание API сервиса	27
2.6 Elasticsearch API.....	28
2.7 Описание схемы работы препроцессора.....	30
2.8 Выводы.....	40
3 Методы испытаний	41
3.1 Проверка метода получения всех доступных специальностей	41
3.2 Проверка метода поиска специальности по названию.....	44
3.3 Проверка метода поиска ПОО	47
3.4 Проверка метода получения списка доступных ПОО.....	49
3.5 Проверка метода поиска ПОО по названию	53
3.6 Проверка метода поиска специальностей в ПОО	55
3.7 Проверка метода получения справочника «Форма обучения»	58
3.8 Проверка метода получения справочника «Пол»	59
3.9 Проверка метода получения справочника «Финансирование».....	60
3.10 Проверка метода поиска адреса.....	61
3.11 Проверка метода подачи заявления	66

3.12 Проверка метода поиска заявлений студента по СНИЛС	75
3.13 Модульное тестирование.....	76
3.14 Выводы	76
ЗАКЛЮЧЕНИЕ	78
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	80
ПРИЛОЖЕНИЕ А Листинг программы	81
ПРИЛОЖЕНИЕ Б Презентация.....	144

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей работе применяют следующие сокращения и обозначения

БД – база данных

ВУЗ – высшее учебное заведение

Госуслуги – портал государственных услуг Российской Федерации

ОС – операционная система

ПО – программное обеспечение

ПОО – профессиональная образовательная организация

СУБД – система управления базой данных

СУЗ – среднее учебное заведение

СНИЛС – страховой номер индивидуального лицевого счёта

ЯП – язык программирования

API (Application programming interface) – программный интерфейс приложения

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных

JSON (JavaScript Object Notation) – формат обмена данными, который основан на языке программирования JavaScript

NoSQL (not only SQL) – это семейство нереляционных баз данных

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределённого приложения в сети

SQL (Structured Query Language) – язык структурированных запросов

YAML (Yet Another Markup Language) – формат сериализации данных

ВВЕДЕНИЕ

В настоящее время можно наблюдать не только российскую, но и общемировую тенденцию на цифровизацию документооборота. Этот процесс затронул многие сферы нашей жизни от экономики до образования. С 2020 года для абитуриентов всех регионов России стала доступна возможность подать заявление на зачисление в ВУЗ через портал Госуслуги. А с 2023 года во многих регионах была открыта новая услуга – подача заявления в школу на запись в 1 класс. Цифровизация процессов – это приоритетное направление ближайших лет.

Разрабатываемые системы должны обладать высокой отказоустойчивостью, быстродействием и обеспечивать целостность и корректность данных.

Объект исследования: сервис-виртуальный ассистент, включающий в себя препроцессор корректировки и валидации входных данных пользователя в режиме реального времени, обеспечивающий целостность и корректность данных, а также предоставляющий справочную информацию для абитуриентов СУЗов, для которых в данный момент отсутствует возможность дистанционной подачи заявлений.

Предмет исследования: препроцессор валидации и корректировки входных данных для виртуального ассистента для дистанционной подачи заявлений на зачисление в образовательные организации

Цель исследования: выявление и анализ требуемого функционала приложения, проектирование архитектуры и реализация приложения, выполняющего функцию виртуального ассистента для дистанционной подачи заявлений на зачисление в образовательные организации, включающего в себя препроцессор обработки и валидации входных данных. Документирование API разработанного приложения.

Задачи исследования:

- сбор и хранение данных для их дальнейшей передачи в образовательную организацию в корректном виде,
- валидация и корректировка введенных пользователем данных,
- реализация быстрого поиска данных по заданным параметрам.

1 Анализ предметной области

При разработке любого программного продукта необходимо провести комплексный анализ существующих технологий и вариантов их использования для решения поставленной задачи, выбрать оптимальную архитектуру приложения, а также определить необходимые функциональные блоки приложения. В данном разделе рассмотрены общие теоретические вопросы необходимые для реализации программного продукта, включающего в себя функции валидации входных данных и их корректировки.

1.1 Архитектура приложения

Архитектура приложения — это план взаимодействия между компонентами системы, такими как база данных, пользовательский интерфейс и внешние системы. Архитектуру можно также описать как набор значимых решений по поводу организации системы программного обеспечения, набор структурных элементов и их интерфейсов, при помощи которых составляется система, вместе с их поведением, определяемым во взаимодействии между этими элементами, а также стиль архитектуры, который направляет эту организацию - элементы и их интерфейсы, взаимодействия и компоновку [1].

Выбор архитектуры приложения одно из важнейших решений при проектировании приложения, от выбранной архитектуры будет зависеть способность системы к адаптации к новым бизнес-требованиям, ее производительность и возможность справляться с возрастающими нагрузками.

В настоящее время в разработке серверных приложений наибольшее распространение имеют две архитектуры: микросервисная и монолитная. Сравнение данных архитектур приведено в таблице 1.

Таблица 1 – Сравнение монолитной и микросервисной архитектур приложения

	Монолит	Микросервис
Развёртывание	Вся система разворачивается с	Развертывание функционала, предполагающего работу нескольких

	Монолит	Микросервис
	минимальными усилиями и трудозатратами	сервисов, требуют тщательной координации
Масштабируемость	Вывод нового функционала требует переразвертывание всей системы	Каждый из элементов системы может быть масштабирован независимо от других элементов.
Гибкость	Не гибко. Сложно сменить используемые технологии, язык программирования и фреймворк	Каждый из сервисов может быть написан на различных языках программирования, которые бы лучше решали поставленные задачи. Простота перехода на новые технологии
Простота модификации	Сбой в одном из функциональных блоков системы может привести к сбою всей системы в целом	Ошибка в одном из сервисов не затрагивает или влияет незначительно на работу других сервисов
Тестирование	Большой объем тестирования. Перед выводом новой сборки приложения необходимо тестирование всех компонентов системы. Из-за сложности тестирования вероятность критических ошибок выше	При изменении внутренней логики микросервиса достаточно провести тестирование только измененного микросервиса. Микросервисы предполагают согласование контрактов, а именно какие данные предполагаются на вход и выход для конкретных методов сервиса. Изменение контракта делает необходимым тестирование всех компонентов, которые его использовали
Безопасность	Обработка всех данных внутри одного сервиса обеспечивает высокую безопасность	Межсервисное взаимодействие предполагает использование API, что не всегда безопасно
Разработка	Быстрая разработка на старте приложения, но долгая при дальнейшем расширении функционала. Команда разработки не может работать независимо в рамках одного большого приложения	Команды разработки разных микросервисов работают независимо. На старте приложения необходимо много времени на разработку контрактов взаимодействия между командами

У каждой из рассмотренных выше архитектур есть свои положительные и отрицательные особенности. Исходя из поставленной задачи, для проектирования сервиса дистанционной подачи заявлений на зачисление с препроцессором обработки и корректировки входных данных лучшим

решением будет выбор микросервисной архитектуры за счет ее гибкости, легкости масштабирования и тестирования, и высокой производительности.

1.2 Способы быстрого поиска данных

Современные информационные системы, предполагающие сбор, хранение, а также обработку данных, невозможно реализовать без БД, а также систем управления БД. В реляционных базах данных информация хранится структурированно в определенных таблицах с заранее определенными атрибутами таблицы, в это время в нереляционных, так называемых «NoSQL», базах данных не используется табличная схема строк и столбцов. В этих базах данных применяется модель хранения, оптимизированная под конкретные требования типа хранимых данных. Например, данные могут храниться как простые пары "ключ — значение", документы JSON или граф, состоящий из ребер и вершин. Но независимо от БД, операция поиска должна выполняться максимально быстро.

Поиск – один из основных механизмов в работе с базами данных. Поиск в базах данных представляет собой процесс отбора множества объектов, которые удовлетворяют условиям запроса. Проблема поиска состоит не только в больших объемах данных, но и в ограничении на время обработки данных, их анализ, а также в росте сложности информационных моделей [2]. Для сокращения времени поиска за счет предоставления ограниченного объема данных, наиболее релевантных входным данным используются поисковые подсказки. Поисковые подсказки широко используются в поисковых строках информационных систем.

Поисковая подсказка предоставляет более широкие возможности, чем поиск по полному слову, например, счетчик поисковой выдачи, сортировки. Первоочередная цель данных дополнительных механизмов — быстро и точно удовлетворить потребность пользователя в нахождении информации [3].

Поисковая подсказка имеет вид наиболее подходящих слов или фраз, которые начинаются или содержат введенные пользователем буквы,

поисковые подсказки реализованы в поисковой строке всех популярных поисковых машин, например, Яндекс, Google, Yahoo, пример поисковой подсказки приведен на рисунке 1.

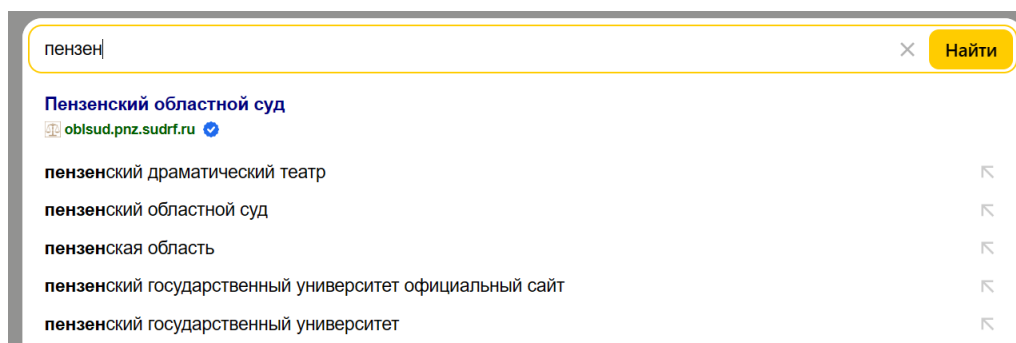


Рисунок 1 – Пример поисковой подсказки в поисковой машине Яндекс

Для ускорения поиска по таблицам в базах данных используются индексы. Индекс – это объект базы данных, цель создания которого заключается в повышении производительности поиска данных. Однако, поисковая подсказка затребует индексирования огромного объема данных, что означает большой объем используемой памяти, а также большие временные затраты на построение индекса. Также индекс ускоряет только поиск по полному совпадению, что не решает целей, поставленных перед подсказчиком в поиске по части слова [3].

В настоящее время, для предоставления подсказок разной сложности реализации активно используются продукты, в основе которых лежит библиотека Apache Lucene (Lucene — свободная библиотека для высокопроизводительного полнотекстового поиска фонда Apache), а именно Elasticsearch и Solr.

Сравнение поисковых серверов приведено в таблице 2.

Таблица 2 – Сравнение поисковых серверов Elasticsearch и Solr

	Solr	ElasticSearch
Возможности использования	Поисковая система с открытым исходным кодом, которая предназначена для корпоративного полнотекстового поиска. Может использоваться для поиска информации и аналитики	Поисковая система, предназначенная для обработки временных рядов, аналитики, масштабирования. Также может выполнять функции полнотекстового поиска

	Solr	ElasticSearch
Преимущества	Широкий функционал для полнотекстового поиска	Самый популярный поисковый сервер Обладает удобным API Имеет хорошую документацию Конструкции запросов просты Является бесплатным
Недостатки	Сложность запросов Ограниченная функциональность Ограниченные возможности аналитики Необходимо использование ZooKeeper для координирования работы Сложность балансировки	Закрытый исходный код

Исходя из анализа, приведенного в таблице 2, оптимальным инструментом для реализации поисковых подсказок является ElasticSearch, который обладает обновляемой и хорошо описанной документацией, удобным API и наиболее простой конструкцией запросов поиска, реализованной через REST [4].

Для организации поиска документов и вычисления релевантности ElasticSearch использует формулу практической функции оценки. Данная формула основывается на вероятностной модели, также в вычисления добавляются нормализация длины поля, коэффициент координации и вес слова, предложения, запроса. В новых версиях ElasticSearch используется функция ранжирования Okapi BM25. Механизм базируется на статистической мере TF-IDF. Okapi BM25 — это семейство функций, которые включают в себя различные параметры и компоненты.

Рассмотрим одну из форм функции BM25:

Пусть задан некоторый запрос Q_1 , при этом Q_1 включает в себя слова $q_1, q_2 \dots q_n$, тогда оценка релевантности документа D высчитывается следующим образом:

$$score(D, Q) = \sum_{i=1}^n IDF(q_i) * \frac{f(q_i, D) * (k_1 + 1)}{f(q_1, D) + k_1 * \left(1 - b + b * \frac{|D|}{avgdl}\right)}, \quad (1)$$

где $f(q_i, D)$ – это TF (частота слова) q_i в документе D ;

$|D|$ – это длина документа D ;

$avgdl$ – это средняя длина документа в коллекции;

k_1 и b – это свободные коэффициенты;

$IDF(q_i)$ – обратная документальная частота слова q_i .

Определение IDF представлено формулой (2).

$$IDF = \log \frac{N}{n(q_i)}, \quad (2)$$

где N – общее количество документов в коллекции;

$n(q_i)$ – это количество документов коллекции, содержащих слово q_i .

1.3 Функциональные блоки сервиса

Виртуальный ассистент для подачи заявлений в образовательные организации имеет несколько задач, решение которых является необходимым, а именно:

- сбор и передача данных в образовательную организацию в корректном виде,
- предоставление информации о доступных в системе образовательных организациях и направлениях подготовки, которые реализуют представленные образовательные организации;
- исключения возможных ошибок пользователя при вводе информации необходимо использовать справочные данные.

В рамках выполнения работ необходимо решить следующие задачи:

- составление справочников, на основании которых будет осуществляться проверка корректности введенных пользователем данных,
- заполнение заявлений на зачисление с подсказками и контролем вводимой пользователем информации,
- разработка метода получения списка всех ПОО, доступных для подачи заявлений,

- разработка метода получения списка доступных специальностей в конкретном ПОО,
- разработка метода заполнения заявления.

Результатом выполнения работы станет виртуальный ассистент для дистанционной подачи заявлений с препроцессором валидации и корректировки пользовательских данных. Данная система позволит исключить ошибки, связанные с человеческим фактором, сократить число некорректных заявлений, требующих проверки оператором образовательной организации.

1.4 Документирование API

Виртуальный ассистент для дистанционной подачи заявлений на зачисление в образовательные организации является серверным приложением и не имеет пользовательского интерфейса. Функционал сервиса может быть интегрирован в любое уже существующее программное обеспечение, например, веб-сайт, мобильное приложение или десктопное приложение. Однако, процесс интеграции – сложный процесс, требующий времени, от API-документации напрямую зависит как быстро пройдет этот процесс [5].

API-документация представляет собой коллекцию инструкций, которые описывают возможности предоставленного API. В зависимости от сферы деятельности, API-документация может иметь различную структуру, но можно однозначно выделить ее обязательные части:

- эндпоинт (англ. endpoint) – это точка приема запроса на стороне сервера,
- параметры эндпоинта – наборы пар «ключ-значение»,
- тип HTTP-запроса, указание типа необходимо для понимания, какой процесс будет выполнен в рамках бизнес-задачи. Например, HTTP-метод PUT выполнит изменение данных, а HTTP-метод DELETE – удаление указанных данных,
- тело запроса,

– ответ на запрос. Ответ состоит из HTTP-кода и тела ответа. HTTP-код позволяет понять разработчику успешность выполнения запроса. Тело ответа представляет собой информацию, которую сервер передает клиенту в ответ на запрос.

Наиболее популярной спецификацией для документирования Rest API на сегодняшний день является OpenAPI. Спецификация OpenAPI — это документ, описывающий возможности API. Документ базируется на YAML-формате и заметках атрибутов в контроллерах и моделях. Данная спецификация не зависит от языка программирования [6].

Также имеется инструмент, использующий спецификацию OpenAPI— Swagger. Swagger предоставляет возможность создания интерактивного API, важным преимуществом которого является его актуальность, изменение любого из объектов тел запроса или ответа, изменение эндпоинта или HTTP-метода запроса будет актуализировано в пользовательском интерфейсе при перезапуске приложения. Swagger гибок в персональных настройках и позволяет добавить описания как на модели запросов и ответов, так и на объекты в моделях, имеется возможность описания различных кодов ответа помимо успешного. Еще одно преимущество Swagger — поддержка инструментов генерации кода по имеющейся Swagger-документации. Помимо прочего, в интерфейсе Swagger пользователь может отправить запрос с необходимыми ему параметрами в теле запроса и получить ответ от сервера, что является полезной функцией при тестировании Rest API.

1.5 Выводы

По итогу проведения анализа предметной области и технологий для разработки виртуального ассистента для подачи заявлений на зачисление в ПОО с препроцессором обработки и корректировки входных данных были выбраны микросервисная архитектура, как наиболее гибкая, масштабируемая и простая в тестировании, поисковый сервис Elasticsearch для реализации функций быстрого поиска за счет его удобного API и простоты конструкций

составления запросов. Для документирования API был выбран наиболее популярная спецификация OpenAPI и инструмент Swagger, предоставляющий интерактивную, актуальную документацию по API с возможностью реализации персональных настроек.

2 Реализация

Перед началом разработки программного обеспечения необходимо определить, какие технологии будут применены для реализации приложения, какой функционал необходимо реализовывать, разработать схему базы данных: определить таблицы и атрибуты, принадлежащие таблицам. А также выяснить, какие сущности должны быть взаимосвязаны.

2.1 Стек технологий, используемый для разработки

В первую очередь необходимо определить язык программирования для разработки ПО. Наиболее подходящий ЯП является Java. Java – строго типизированный объектно-ориентированный язык программирования. Java постоянно развивается и добавляет разработчику все больше новых гибких и удобных инструментов для разработки приложения. Java популярен для разработки высоконагруженных серверных приложений, имеет актуальную и постоянно обновляемую документацию. На Java уже написаны открытые библиотеки, предлагающие решения по наиболее популярным задачам. ЯП используется вместе с наиболее популярными фреймворками Spring Boot и Spring Data JPA, Jackson Framework, Flyway и Hibernate.

Spring Boot – фреймворк для ЯП Java, предоставляет набор библиотек и инструментов для разработки ПО корпоративного уровня [7]. Для упрощения работы с БД используется спецификация JPA и библиотека для работы с БД Hibernate, которые включают в себя все базовые операции работы с базами данных. Для удобного отслеживания и версионирования базы данных используется библиотека, реализующая механизм миграций Flyway [8].

В качестве среды разработки используется IntelliJ IDEA. Наиболее современная интегрированная среда программного обеспечения, поддерживающая ЯП Java, создание приложений с использованием Spring, а также работу с базами данных.

Для реализации данного проекта в качестве СУБД была выбрана PostgreSQL. PostgreSQL – объектно-реляционная система управления базами данных, имеет свободную лицензию и обладает полной SQL-совместимостью. Поддерживает следующие операционные системы: Linux, Microsoft Windows, Oracle Solaris, IBM AIX, macOS, HP-UX, QNX. PostgreSQL поддерживает логические репликации, параллельные запросы к БД и декларативное разбиение таблиц. PostgreSQL работает со сложными структурами данных. Помимо этого, в PostgreSQL возможны вложенные и составные конструкции.

В настоящее время, при разработке программного обеспечения принято подготавливать автоматические тесты, запускаемые в этап сборки приложения. Автоматические тесты позволяют выявить ошибки в программе до перехода к ручному тестированию. Автоматические тесты можно разделить на интеграционные и модульные.

Модульные тесты (unit-тесты) – это тесты на наименьший логический функционал приложения – метод.

Интеграционные тесты – это тесты, проверяющие более крупные функциональности приложения, классы, цепочки методов или компоненты, отвечающие за внешнее взаимодействие.

Для реализации интеграционного и модульного тестирования используется фреймворк Junit и библиотека Mockito [9]. Они предоставляют набор инструментов для написания модульного и интеграционного тестирования, гибки в персонализации тестов.

2.2 Требования к разработке

Виртуальный ассистент с препроцессором корректировки и валидации входных данных представляет собой сервис, который может быть интегрирован в любое существующее ПО, поэтому должен обладать понятной и полной документацией API. Также должен включать в себя валидацию и корректировку пользовательских данных для их корректной записи. Система должна быть устойчива к высоким нагрузкам. Обладать базой данных,

способной хранить большие объемы данных и осуществлять быстрый поиск и выдачу информации.

Разрабатываемое приложение должно включать в себя следующие функции:

- работа с заявлениями:
 - 1) подача заявления;
 - 2) поиск заявления по СНИЛС;
 - 3) поиск адреса;
- работа со специальностями:
 - 1) поиск специальности по названию;
 - 2) получение списка всех доступных специальностей;
 - 3) поиск ПОО, реализующих данную специальность;
- работа с образовательными организациями:
 - 1) поиск образовательной организации по названию;
 - 2) поиск специальностей в ПОО по названию;
 - 3) получение списка всех доступных образовательных организаций;
- справочные данные:
 - 1) получение данных справочника «Форма обучения»;
 - 2) получение данных справочника «Пол»;
 - 3) получение данных справочника «Финансирование».

2.3 Алгоритм работы сервиса

Последовательность работы виртуального ассистента для зачисления в образовательные организации с препроцессором обработки и корректировки входных данных в общем виде можно представить следующим образом: клиентское приложение, используя REST API посылает запрос, переданные данные проходят валидацию, в случае выявления ошибки при валидации клиентскому приложению направляется HTTP-код ошибки и сообщение, описывающее ошибку, в случае успешной валидации сервис обращается к базе данных на запись или получение информации (в зависимости от входящего

запроса), при успехе выполнения запроса к БД клиентское приложение получает запрашиваемую информацию или записанную информацию, в случае ошибки выполнения запроса к БД, клиентское приложение в ответ получает HTTP-код и сообщение, соответствующее ошибке. Последовательность работы сервиса представлена на рисунке 2.

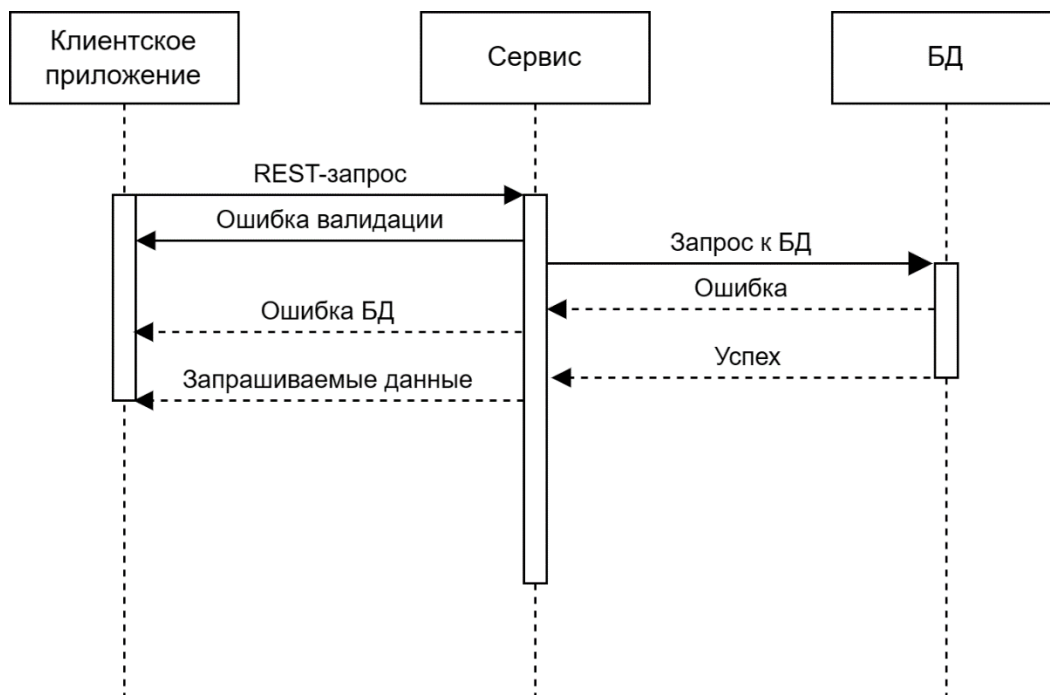


Рисунок 2 – UML-диаграмма последовательности работы сервиса

Валидация входящих запросов – одна из основных функциональных возможностей сервиса, валидация представляет собой множество проверок, таких как проверка на заполнение обязательных полей, проверка длины введенных строковых данных, проверка соответствия введенных кодов кодам справочных данных. Для иллюстрации механизма валидации приведена диаграмма последовательности заполнения заявления, представленная на рисунке 3. Диаграмма классов приложения представлена на рисунках 4-6.

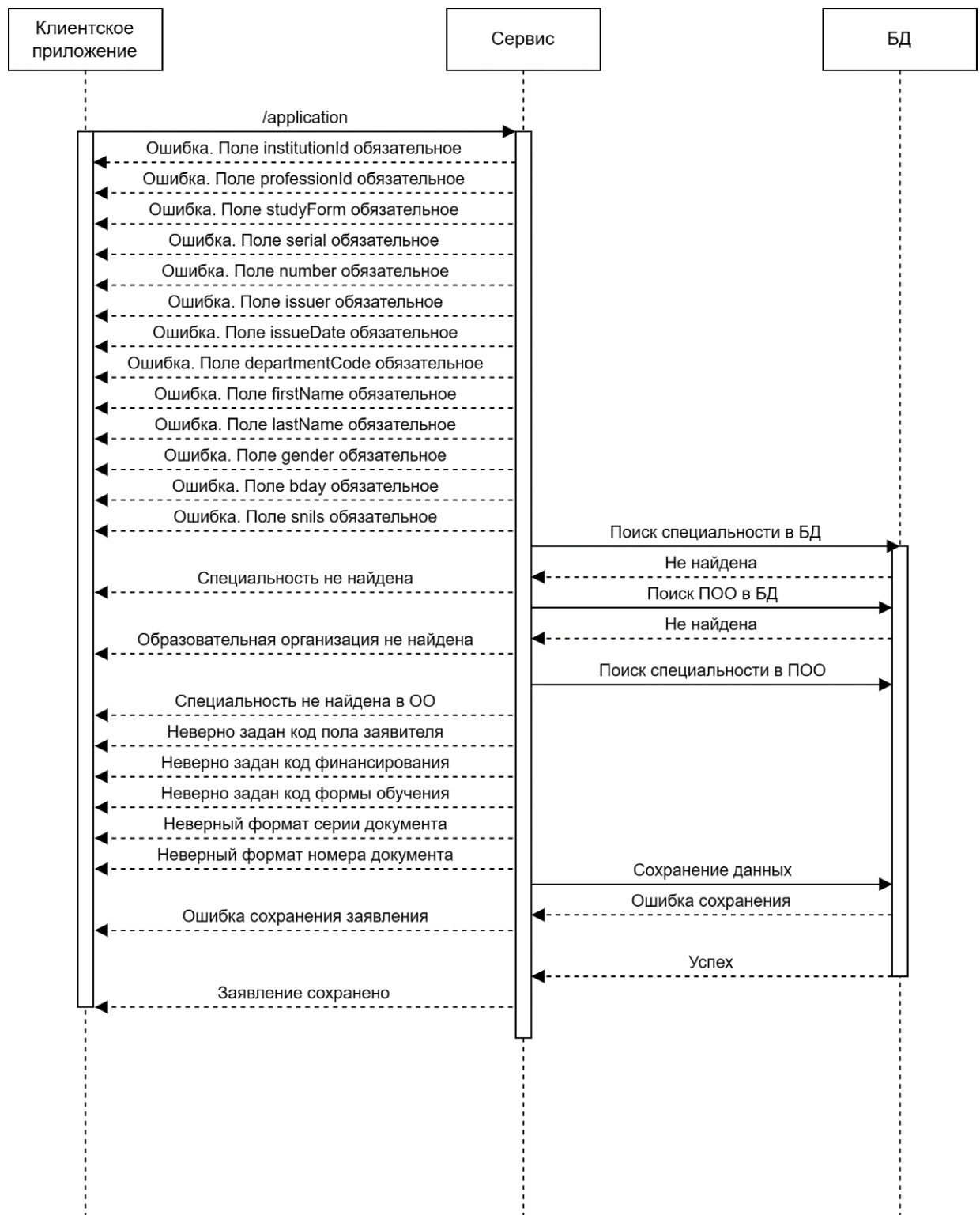


Рисунок 3 – UML-диаграмма последовательности подачи заявления



Рисунок 4 – UML-диаграмма классов сервиса

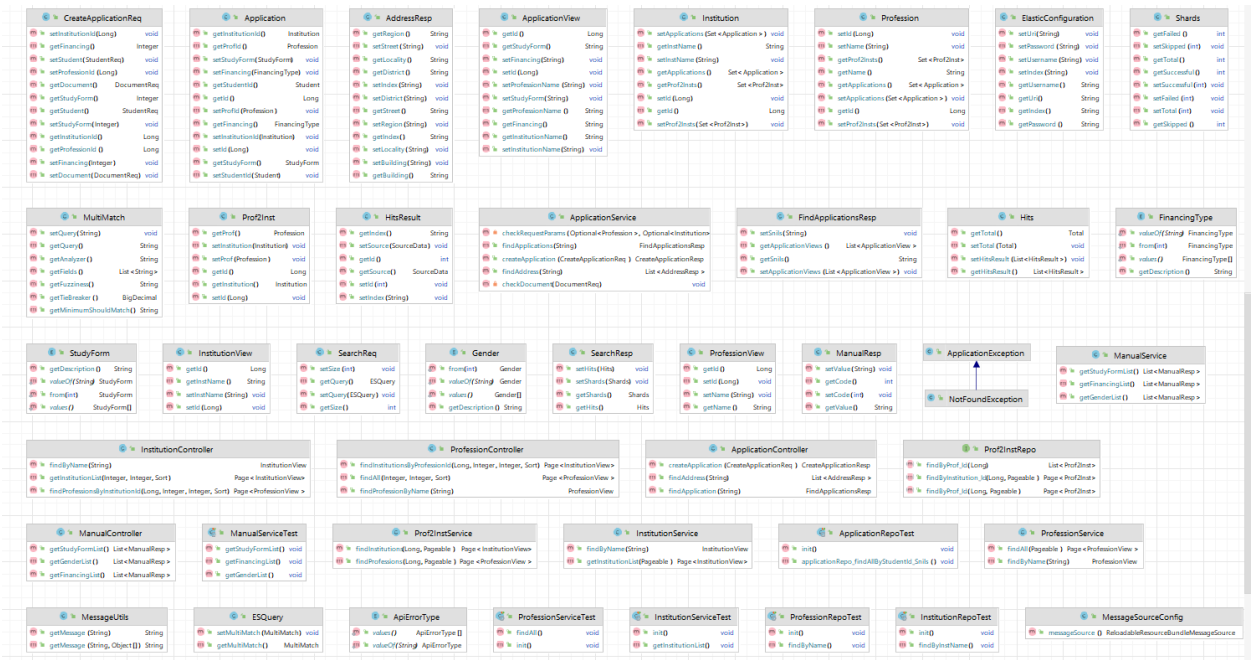


Рисунок 5 – UML-диаграмма классов сервиса

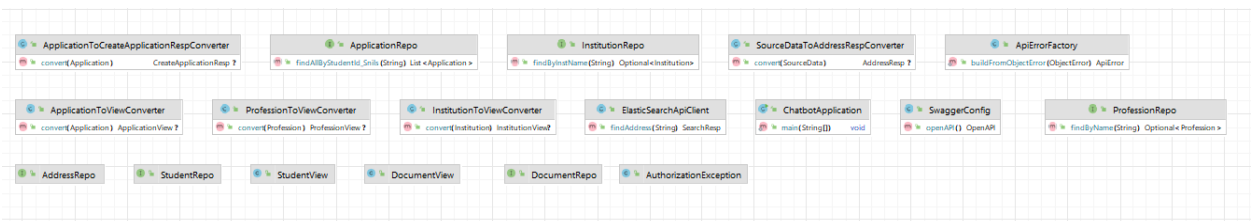


Рисунок 6 – UML-диаграмма классов сервиса

2.4 Описание структуры и таблиц БД

Для работы приложения были разработаны следующие таблицы:

- student – таблица, содержащая сведения об абитуриенте, представленная в таблице 4,
- document – таблица, содержащая данные документа удостоверяющего личность абитуриента, атрибуты представлены в таблице 5;
- institution – таблица, содержащая перечень образовательных организаций, доступных для подачи заявления на зачисление, представленная в таблице 6;
- profession – таблица, содержащая перечень доступных специальностей и необходимую информацию о них, представленная в таблице 7,
- prof2inst – таблица, содержащая соотношение образовательного учреждения и образовательных программ, представленная в таблице 8,
- application – таблица, содержащая информацию о заявлении, представленная в таблице 9,
- address – таблица, содержащая адрес, указанный абитуриентом, атрибуты представлены в таблице 10.

Таблица 4 - Атрибуты таблицы «student»

Имя столбца	Описание	Обязательность	Тип данных	Роль атрибута	Пример
id	Идентификатор абитуриента	Да	bigint	PK	1
first_name	Имя абитуриента	Да	character varying(255)		Петров
last_name	Фамилия абитуриента	Да	character varying(255)		Петр
middle_name	Отчество абитуриента	Нет	character varying(255)		Петрович
bday	Дата рождения	Да	date		19.02.1999
gender	Пол	Да	integer		1
document_id	Идентификатор документа абитуриента из таблицы document	Да	bigint	FK	1
address_id	Идентификатор адреса	Да	bigint	FK	2

Имя столбца	Описание	Обязательность	Тип данных	Роль атрибута	Пример
	абитуриента из таблицы address				

Поле gender может принимать следующие значения:

- 0 – мужской,
- 1 – женский.

Таблица 5 - Атрибуты таблицы «document»

Имя столбца	Описание	Обязательность	Тип данных	Роль атрибута	Пример
id	Идентификатор документа	Да	bigint	РК	1
doc_seria	Серия документа	Да	character varying(255)		5600
doc_number	Номер документа	Да	character varying(255)		000111
issuer text	Кем выдан	Да	character varying(255)		МВД России по Пензенской области
issue_date	Дата выдачи	Да	date		01.02.2005
dep_code	Код подразделения	Да	character varying(255)		580-001

Таблица 6 - Атрибуты таблицы «institution»

Имя столбца	Описание	Обязательность	Тип данных	Роль атрибута	Пример
id	Идентификатор	Да	bigint	РК	1
inst_name	Название образовательной организации	Да	character varying(255)		ПГУ

Таблица 7 - Атрибуты таблицы «profession»

Имя столбца	Описание	Обязательность	Тип данных	Роль атрибута	Пример
id	Идентификатор	Да	bigint	РК	1
name	Название специальности	Да	character varying(255)		Реклама

Таблица 8 - Атрибуты таблицы «prof2inst»

Имя столбца	Описание	Обязательность	Тип данных	Роль атрибута	Пример
id	Идентификатор	Да	bigint	PK	1
institution_id	Идентификатор ПОО	Да	bigint	FK	2
prof_id	Идентификатор направления подготовки	Да	bigint	FK	1

Таблица 9 - Атрибуты таблицы «application»

Имя столбца	Описание	Обязательность	Тип данных	Роль атрибута	Пример
id	Идентификатор	Да	bigint	PK	1
student_id	Название специальности	Да	bigint	FK	15
prof_id	Идентификатор направления подготовки	Да	bigint	FK	6
institution_id	Идентификатор ПОО	Да	bigint	FK	5
study_form	Форма обучения	Да	integer		1
financing	Форма финансирования	Да	integer		1
snils	СНИЛС	Да	character varying(255)		123-456- 789 00

Поле study_form может принимать следующие значения:

- 0 – заочно,
- 1 – очно.

Поле financing может принимать следующие значения:

- 0 – на договорной основе,
- 1 – бюджет.

Таблица 10 - Атрибуты таблицы «address»

Имя столбца	Описание	Обязательность	Тип данных	Роль атрибута	Пример
id	Идентификатор	Да	bigint	PK	1
region	Область	Да	character varying(255)		Пензенская область
district	Округ	Да	character varying(255)		город Пенза
Имя столбца	Описание	Обязательность	Тип данных	Роль атрибута	Пример
locality	Населенный пункт	Да	character varying(255)		город Пенза

Имя столбца	Описание	Обязательность	Тип данных	Роль атрибута	Пример
street	Улица	Да	character varying(255)		1-й проезд 40 лет Октября
building	Строение	Да	character varying(255)		10
index	Индекс	Да	character varying(255)		440031
flat	Квартира	Нет	character varying(255)		1

Схема базы данных, определяющая таблицы, атрибуты таблиц с указанием типов и ограничений целостности приведена на рисунке 7.

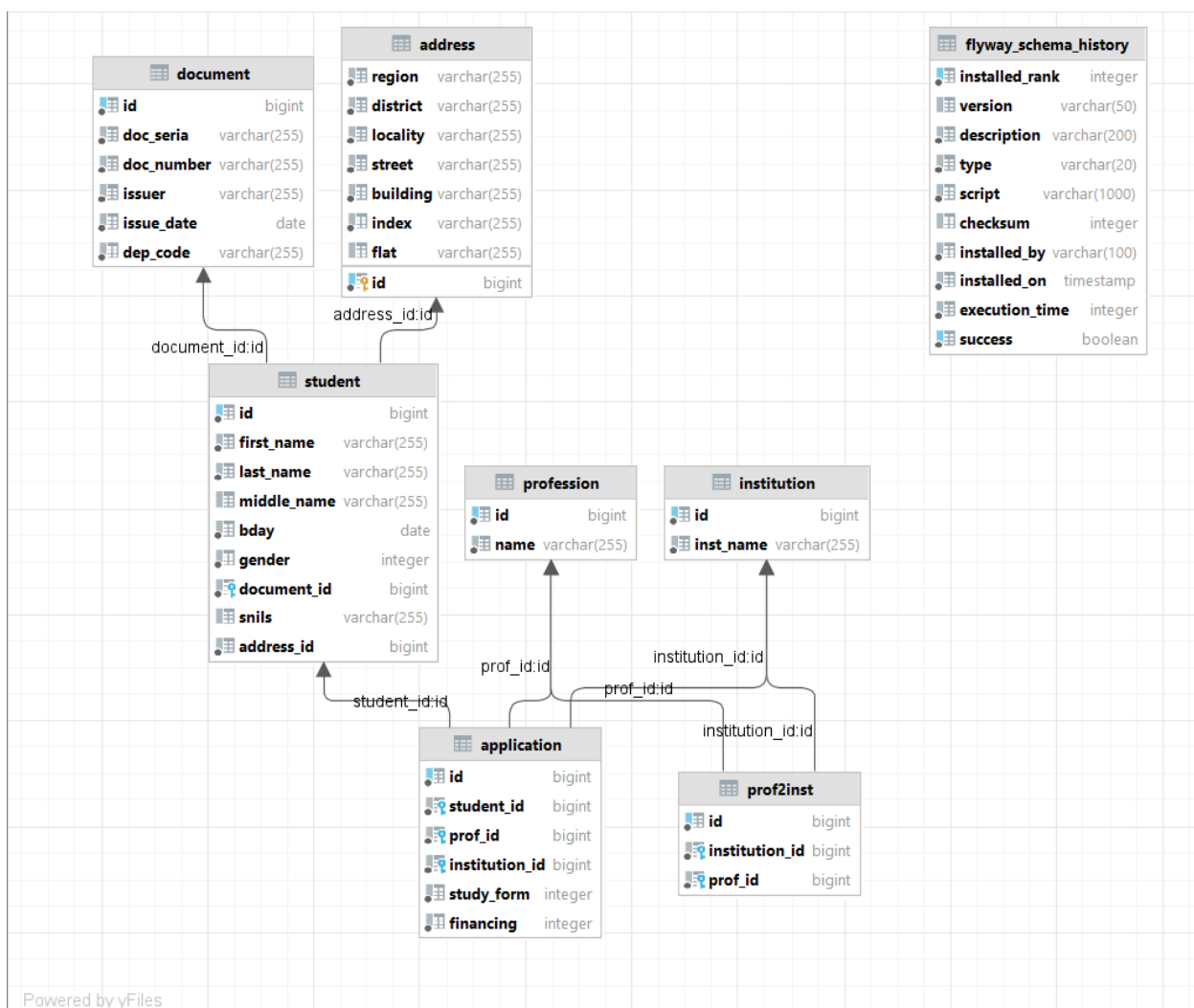


Рисунок 7 – Схема таблиц и атрибутов БД

2.5 Описание API сервиса

В ходе проведенного анализа механизма подачи заявлений на зачисление в образовательные организации были выделены основные функциональные возможности приложения.

API сервиса включает в себя следующие методы:

- метод подачи заявления,
- метод поиска заявления по СНИЛС,
- метод поиска адреса,
- метод поиска специальности по названию,
- метод получения списка всех доступных специальностей,
- метод поиска ПОО, реализующих конкретную специальность,
- метод поиска образовательной организации по названию,
- метод поиска специальностей в ПОО,
- метод получения списка всех доступных организаций для подачи заявления на зачисление,
- метод получения данных справочника «Форма обучения»,
- метод получения данных справочника «Пол»,
- метод получения справочника «Финансирование».

Все методы представлены в виде REST API и задокументированных с помощью Swagger, страница документации REST API представлена на рисунке 8.

Заявления		^
POST	/application	Подать заявление
GET	/application/student/{snils}	Поиск заявлений студента по СНИЛС
GET	/application/findAddress	Поиск адреса
Специальности		^
GET	/profession/search	Поиск специальности по названию
GET	/profession/list	Получение списка всех доступных специальностей
GET	/profession/institution/list	Поиск образовательных организаций, реализующих конкретную специальность
Справочные данные		^
GET	/manual/studyForm	Получение справочника Форма обучения
GET	/manual/gender	Получение справочника Пол
GET	/manual/financing	Получение справочника Финансирование
Образовательные организации		^
GET	/institution/search	Поиск образовательной организации по названию
GET	/institution/profession/list	Поиск специальностей в образовательной организации
GET	/institution/list	Получение списка доступных образовательных организаций

Рисунок 8 – Документация REST API сервиса

2.6 Elasticsearch API

Для реализации поиска адресов используется поисковый сервис и нереляционная БД Elasticsearch. Для предоставления поисковых подсказок в БД должен быть создан индекс и документ, заполненный данными, по которым будет осуществляться поиск. Индекс – это аналог понятия «база данных» в реляционных БД. Документ представляет собой набор полей: пару ключ-значение, является аналогом таблицы в реляционных БД.

Был создан индекс assistant, в данном индексе был создан документ, по которому проводится поиск для реализации поисковых подсказок.

Набор полей и их тип данных в созданном документе приведены в таблице 11.

Таблица 11- Набор полей для документа

Имя столбца	Описание	Тип данных	Пример
region	Область	string	Пензенская область
district	Округ	string	городской округ город Пенза
locality	Населенный пункт	string	город Пенза
street	Улица	string	1-й проезд 40 лет Октября
building	Строение	string	10
index	Индекс	string	440031

Взаимодействие с API Elasticsearch реализовано через REST-запросы, для создания индекса использовался запрос:

PUT /assistant, где assistant – название индекса.

Для добавления документа в индекс assistant использован запрос вида: PUT assistant /_doc/1 с телом запроса:

```
{  
  "region": "Пензенская область",  
  "district": "городской округ город Пенза",  
  "locality": "город Пенза",  
  "street": "1-й проезд 40 лет Октября",  
  "building": "10",  
  "index": "440031"  
}
```

Механизм корректировки применяется для поиска адреса абитуриента и позволяет привести входные данные к нормализованному виду для последующего сохранения в БД, а также для исключения ошибок при заполнении данных.

Для поиска по документу индекса Elasticsearch API используется метод `_search`, который имеет возможность гибкой настройки. Для корректировки введенных пользователем данных путем поисковых подсказок был выбран механизм поиска – `multi_match` [10]. `multi_match` реализует поиск по нескольким полям, что является оптимальным решением при поиске адреса объекта, так как пользователь вводит не только название улицы, но и населенный пункт, номер строения, область. Стоит учитывать, что пользователь вводит свой адрес в произвольном порядке его элементов, например, сначала улицу и строение, затем город, либо город, улицу и строение, для данной ситуации имеет смысл предусмотреть нечеткий поиск. Возможность нечеткого поиска предусмотрена параметром `fuzziness`, который принимает значение максимальное расстояние между словами. Также

fuzziness используется для автодополнения и корректировки возможных ошибок при вводе. По умолчанию, поиск по multi_match использует стратегию лучших полей – best_fields, которая предназначена для поиска по нескольким полям и выбора оценки с наивысшей степенью соответствия определенного поля. Однако, для получения релевантного ответа этого недостаточно, так как несколько элементов могут будут иметь одинаковую оценку. Для выставления окончательной оценки совпадения необходимо учитывать результаты сопоставления элементов, имеющих одинаковую степень соответствия, для этого зададим параметр tie_breaker, который также будет суммировать оценки и других полей для нормализации оценки.

При условии большого объема данных, которые могут подходить под пользовательский запрос, необходимо ограничить выборку наиболее релевантными значениями. Для ограничения выборки наиболее подходящими значениями установлен минимальный процент сходства поля документа, это возможно с помощью параметра minimum_should_match, также ограничен и размер выдачи первыми пятью наиболее релевантными значениями для удобства пользователя, за ограничение размера выдачи отвечает параметр size.

2.7 Описание схемы работы препроцессора

Работу препроцессора валидации и корректировки входных данных можно проиллюстрировать на примере процесса заполнения заявления, данный процесс является основным и принимает данные, возвращаемые другими методами системы.

Тело запроса на подачу заявления на зачисление имеет следующий вид:

```
{  
  "institutionId": 0,  
  "professionId": 0,  
  "financing": 0,  
  "studyForm": 0,  
  "document": {
```

```
"serial": "string",
"number": "string",
"issuer": "string",
"issueDate": "2023-05-13",
"departmentCode": "string"
},
"student": {
  "firstName": "string",
  "middleName": "string",
  "lastName": "string",
  "gender": 0,
  "bday": "2023-05-13",
  "snils": "string",
  "address": {
    "region": "string",
    "district": "string",
    "locality": "string",
    "street": "string",
    "building": "string",
    "index": "string",
    "flat": "string"
  }
}
}
```

Помимо персональных данных во вложенных объектах `document` и `student` присутствуют также поля, содержащие себе идентификаторы из БД и справочные значения, такие как: `institutionId`, `professionId`, `financing`, `studyForm`, `student.gender` и `student.address`. Это необходимо для исключения

ошибок, связанных с человеческим фактором, а также для обеспечения быстродействия системы.

Рассмотрим один из возможных сценариев заполнения заявления, когда пользователь выбирает образовательную организацию, а затем специальность, которую реализует данное ПОО.

Клиентское приложение отправляет запрос на получение списка всех доступных образовательных организация – `institution/list`, данные об образовательных организациях отдаются в следующем виде:

```
{
  "id": 1,
  "instName": "КАДР №26"
},
{
  "id": 2,
  "instName": "ДЗМ №1"
}, ...
```

У выбранной образовательной организации необходимо значение поля `id` использовать для заполнения поля `institutionId` в теле запроса для подачи заявления.

Далее необходимо получить список специальностей, доступных в ПОО, для этого используется запрос `institution/profession/list?institutionId=1`, где в параметр `institutionId` передается идентификатор ПОО, полученный в предыдущем запросе. Данные о специальностях отдаются клиентскому приложению в следующем виде:

```
{
  "id": 5,
  "name": "Реклама"
},
{
```

```

    "id": 13,
    "name": "Анимация"
  },
  {
    "id": 19,
    "name": "Дизайн"
  }, ...

```

Значение поля `id` используется для заполнения `professionId` в теле запроса на подачу заявления.

Для заполнения полей `financing`, `studyForm`, `student.gender` необходимо обратиться к методам получения справочной информации: `manual/financing`, `manual/studyForm` и `manual/gender` соответственно. Запросы имеют следующие форматы ответа:

Ответ на запрос `manual/studyForm`

```

[
  {
    "code": 0,
    "value": "Заочно"
  },
  {
    "code": 1,
    "value": "Очно"
  }
]

```

Ответ на запрос `manual/financing`

```

[
  {
    "code": 0,
    "value": "На договорной основе"
  }
]

```

```
},  
{  
  "code": 1,  
  "value": "Бюджет"  
}  
]
```

Ответ на запрос *manual/gender*

```
[  
  {  
    "code": 0,  
    "value": "Мужской"  
  },  
  {  
    "code": 1,  
    "value": "Женский"  
  }  
]
```

Запросы имеют одинаковую структуру ответа, значения полей `code` используются для заполнения соответствующих им полей в запросе на подачу заявления на зачисление.

Для заполнения объекта `student.address` необходимо обратиться к методу `application/findAddress`. В параметр `address` передается адрес или часть адреса для поиска. Например, на запрос с параметром `address = «проезд 2 малиновый»`, будет получен следующий ответ:

```
[  
  {  
    "region": "Пензенская область",  
    "district": "городской округ город Пенза",  
    "locality": "город Пенза",
```

```
"street": "10-й Малиновый проезд",
"building": "2",
"index": "440015"
},
{
  "region": "Пензенская область",
  "district": "городской округ город Пенза",
  "locality": "город Пенза",
  "street": "10-й Малиновый проезд",
  "building": "5",
  "index": "440015"
},
{
  "region": "Пензенская область",
  "district": "городской округ город Пенза",
  "locality": "город Пенза",
  "street": "10-й Малиновый проезд",
  "building": "6",
  "index": "440015"
},
{
  "region": "Пензенская область",
  "district": "городской округ город Пенза",
  "locality": "город Пенза",
  "street": "10-й Малиновый проезд",
  "building": "7",
  "index": "440015"
},
{
```

```
"region": "Пензенская область",  
"district": "городской округ город Пенза",  
"locality": "город Пенза",  
"street": "10-й Малиновый проезд",  
"building": "10",  
"index": "440015"  
}  
]
```

Необходимо выбрать подходящий вариант и передать выбранный объект в тело запроса на подачу заявления в объект `student.address`, добавив при необходимости также параметр `flat` для указания номера квартиры при наличии.

При получении запроса `/application` сервис проводит первый этап валидации введенных данных, которая включает в себя проверку на наличие всех обязательных полей и объектов в теле запроса.

Второй этап валидации включает в себя поиск образовательной организации и специальности по заданным идентификаторам, при их наличии в БД проверяется, реализует ли данное ПОО эту специальность, в случае, когда в БД не были найдены соответствующие записи, клиентское приложение получит ошибку `NotFoundException` с HTTP-кодом 404 и поясняющим сообщением. Далее проводится поиск соответствия справочных значений и введенных кодов. Если не было найдено соответствие, то клиентское приложение получит ошибку `NotFoundException` с HTTP-кодом 404 и поясняющим сообщением.

На втором этапе проводится также проверка длины паспортных данных, для нумерации бланка паспорта гражданина Российской Федерации установлено 3 группы цифр, первые две группы цифр являются серией бланка паспорта и имеют длину в 4 цифры, третья группа цифр содержит в себе 6 цифровых значений и является номером бланка паспорта. При несоответствии

длин полей serial и number объекта document клиентское приложение получит ошибку ApplicationException с HTTP-кодом 400 и сообщение, содержащее описание, в каком из полей была допущена ошибка.

В случае успешного прохождения двух этапов валидации заявление будет сохранено.

Диаграмма деятельности процессов валидации и корректировки на примере подачи заявления на зачисление представлена на рисунках 9-12.

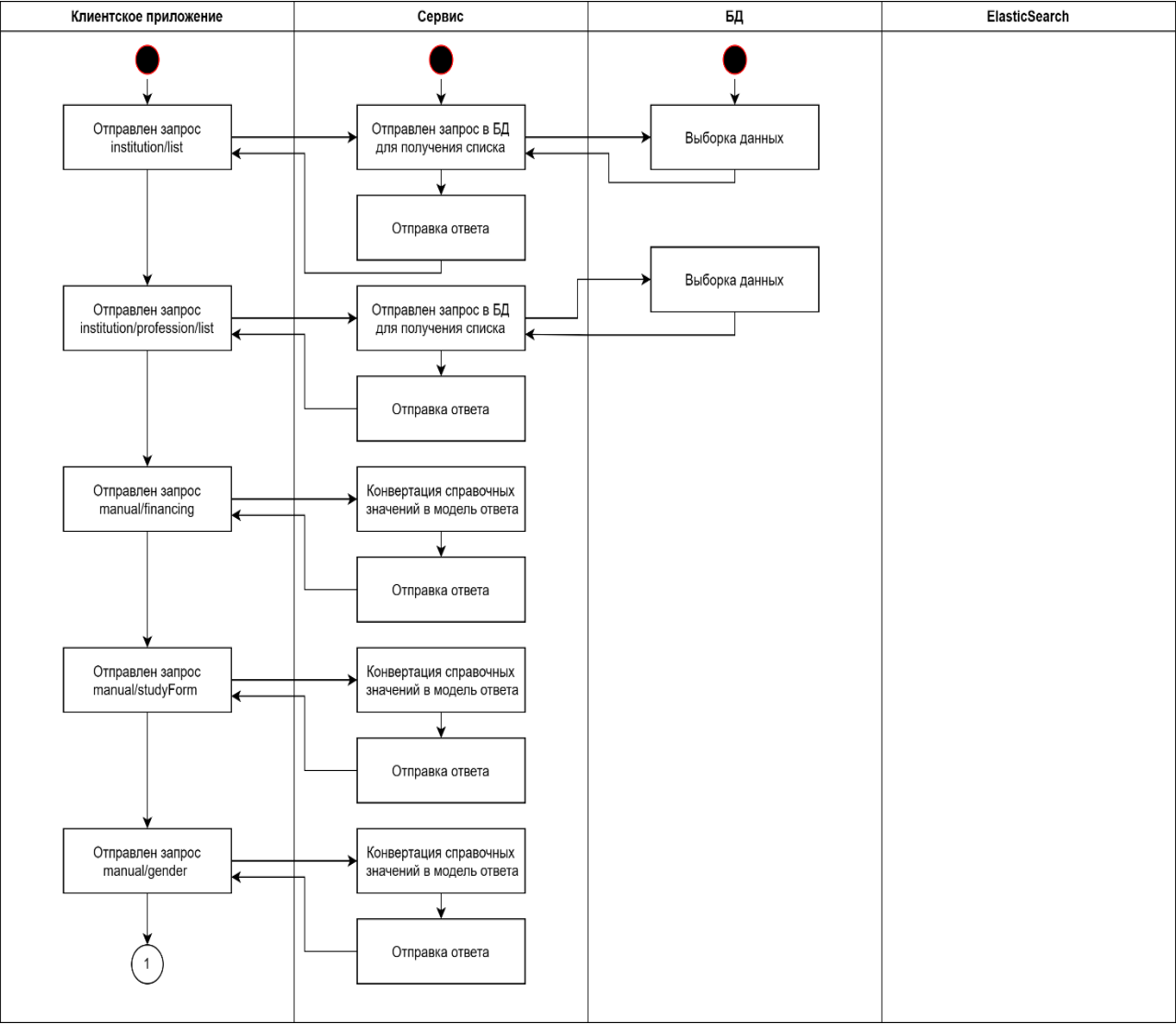


Рисунок 9 - Диаграмма деятельности процессов валидации и корректировки 1 часть

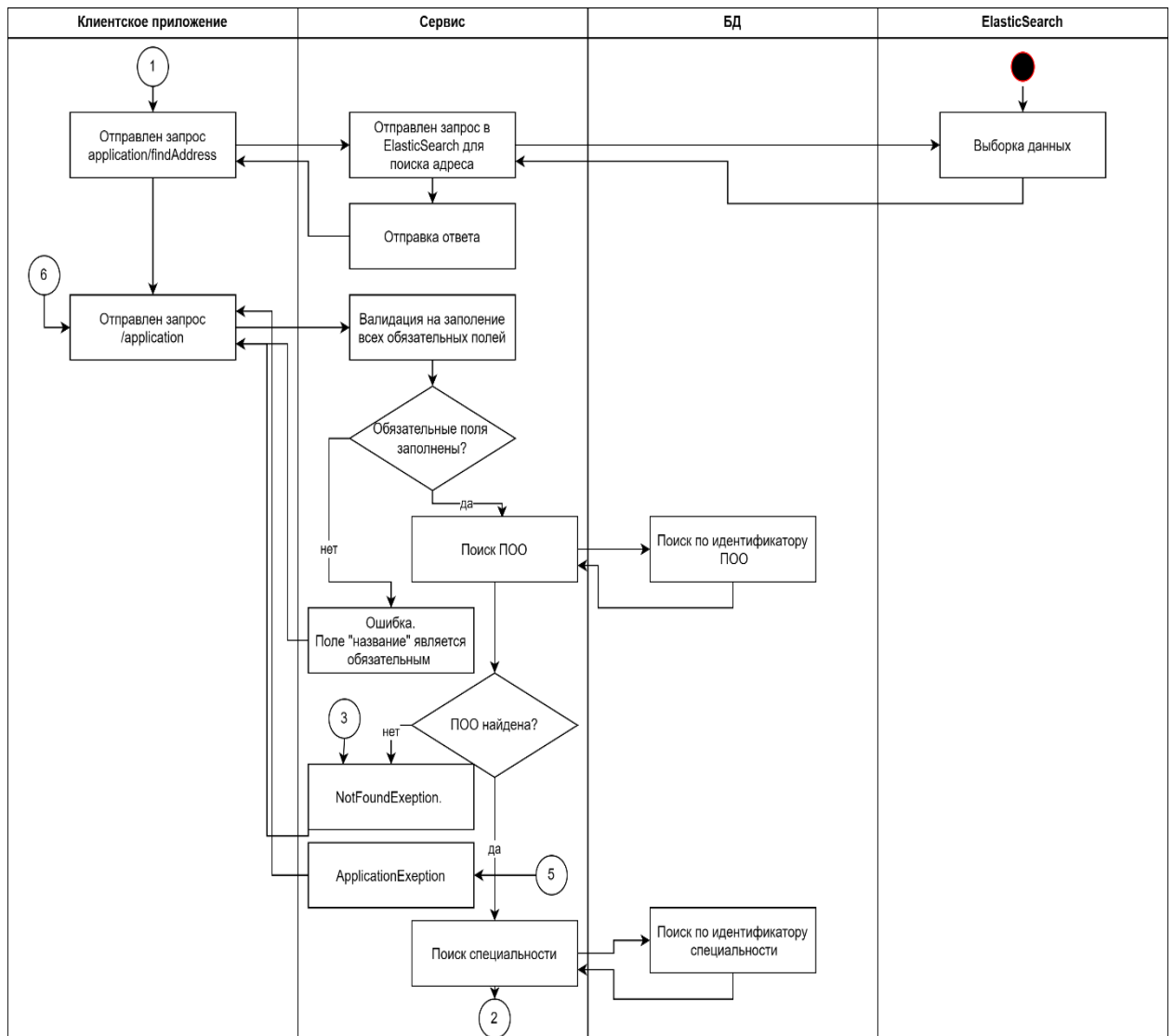


Рисунок 10 - Диаграмма деятельности процессов валидации и корректировки
2 часть

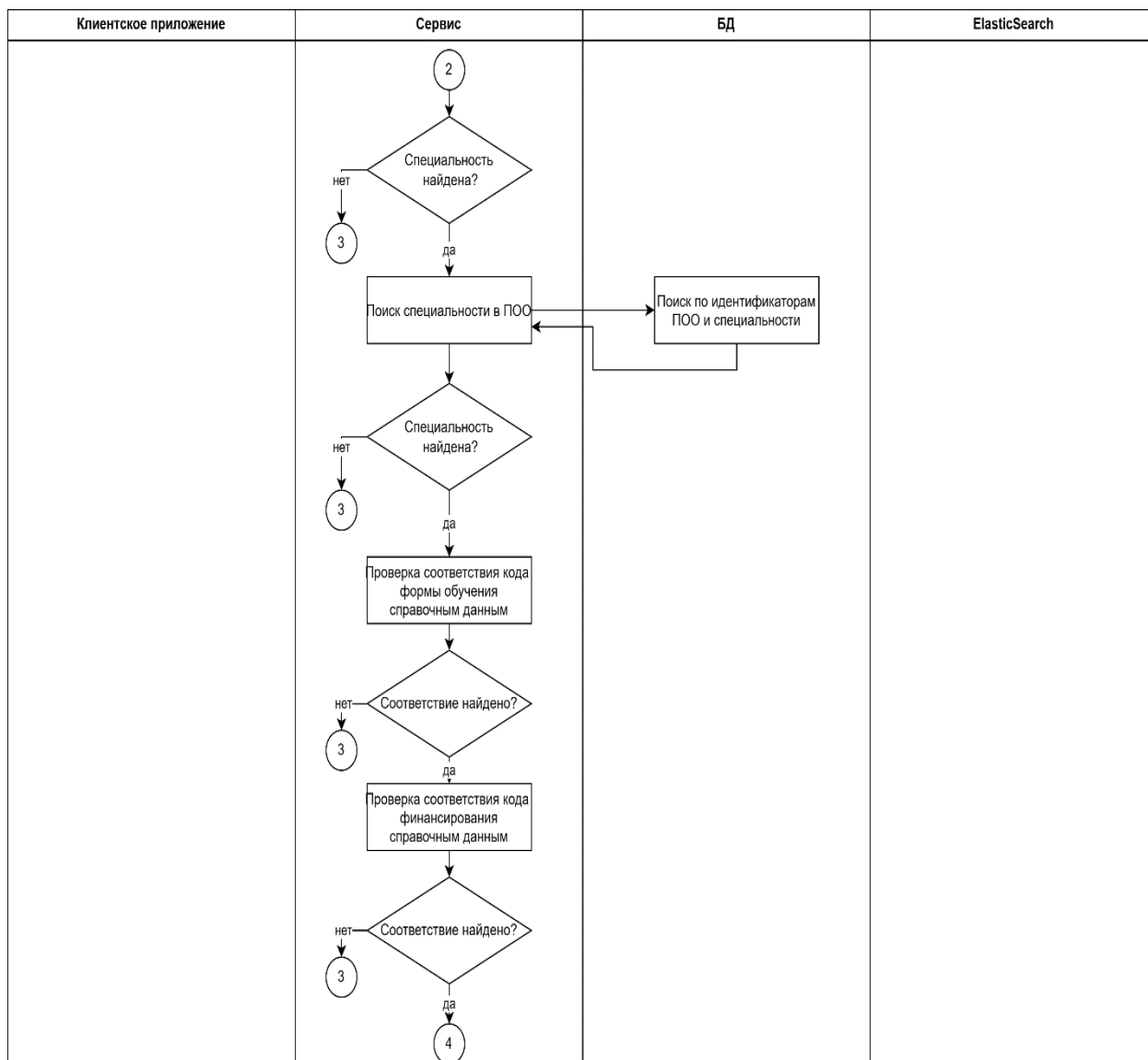


Рисунок 11 - Диаграмма деятельности процессов валидации и корректировки
3 часть

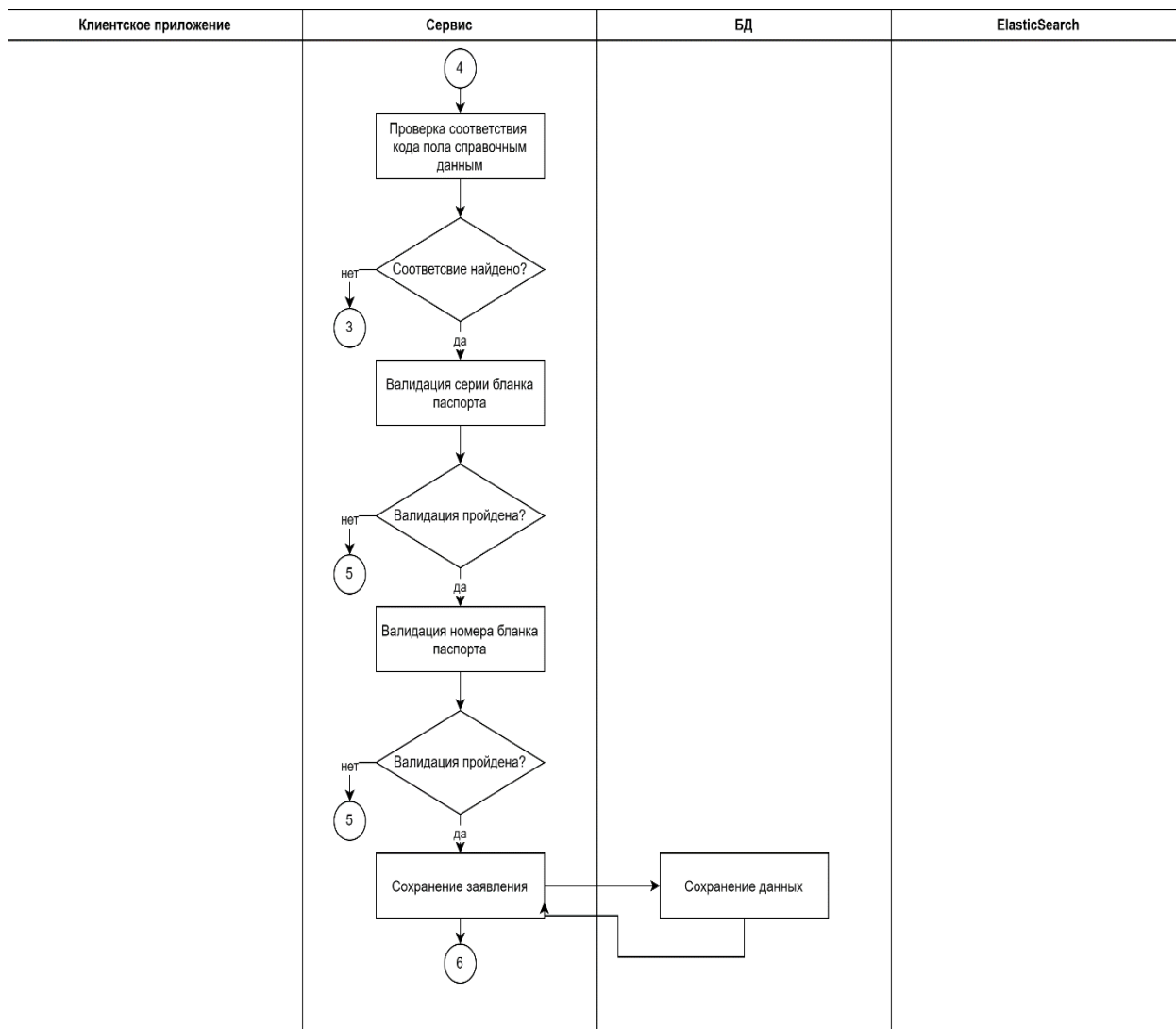


Рисунок 12 - Диаграмма деятельности процессов валидации и корректировки
4 часть

2.8 Выводы

При разработке препроцессора валидации и корректировки входных данных для виртуального ассистента был выбран стек технологий, позволяющий с максимальной производительностью реализовать поставленные перед приложением задачи: предоставление полной документации с примерами запросов и ответов для дальнейшей интеграции в клиентскую систему, валидацию и корректировку входящих данных для исключения ошибок, связанных с человеческим фактором, а также ошибок при сопоставлении пользовательских данных и справочных данных.

3 Методы испытаний

Тестирования приложения включает в себя как ручную проверку (функциональное тестирование) каждого из методов API с помощью пользовательского интерфейса Swagger (Swagger UI), так и модульное тестирование программы посредством фреймворка Junit и библиотеки Mockito.

3.1 Проверка метода получения всех доступных специальностей

Убедимся, что данные, предоставляемые в ответе метода соответствуют данным, хранящимся в базе данных. Также проверит корректность работы постраничной выдачи данных.

Для проверки выполним следующие действия:

- в Swagger UI выбрать вкладку «Специальности», метод «Получение списка всех доступных специальностей», которые представлены на рисунке 13,
- задать размер страницы выдачи равным 5,
- задать номер страницы выдачи 0,
- выполнить запрос,
- получить список специальностей.

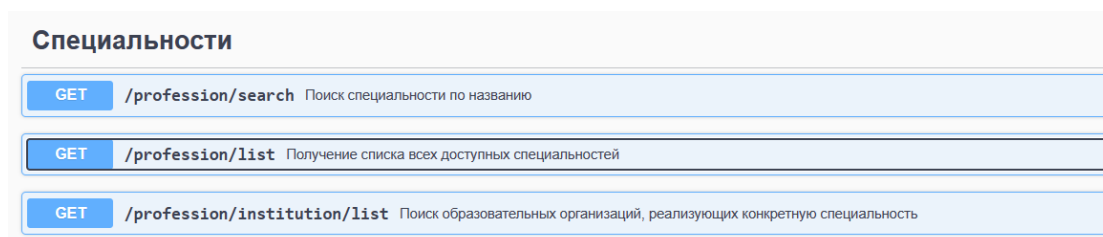


Рисунок 13 – Метод «Получение списка всех доступных специальностей»

Ответ, содержащий список специальностей:

```
{
  "content": [
    {
      "id": 1,
      "name": "Сестринское дело"
```

```
    },  
    {  
      "id": 2,  
      "name": "Лечебное дело"  
    },  
    {  
      "id": 3,  
      "name": "Гостиничное дело"  
    },  
    {  
      "id": 4,  
      "name": "Туризм"  
    },  
    {  
      "id": 5,  
      "name": "Реклама"  
    }  
  ],  
  "pageable": {  
    "sort": {  
      "sorted": true,  
      "unsorted": false,  
      "empty": false  
    },  
    "offset": 0,  
    "pageSize": 5,  
    "pageNumber": 0,  
    "paged": true,  
    "unpaged": false
```

```
},  
  "last": false,  
  "totalElements": 30,  
  "totalPages": 6,  
  "size": 5,  
  "number": 0,  
  "sort": {  
    "sorted": true,  
    "unsorted": false,  
    "empty": false  
  },  
  "first": true,  
  "numberOfElements": 5,  
  "empty": false  
}
```

Сравним полученные данные и данные, хранящиеся в БД:

- выполнить запрос в СУБД, приведенный ниже, убедиться в соответствии данных в ответе метода получения всех доступных специальностей и данных, хранящихся в БД, которые представлены на рисунке 14,

- выполнить запрос в СУБД, приведенный ниже, для получения общего количества записей в таблице,

- сравнить результаты, результат выполнения запроса представлен на рисунке 15.

Запрос к БД:

```
select * from profession  
group by id order by id limit 5
```

	id	name
1	1	Сестринское дело
2	2	Лечебное дело
3	3	Гостиничное дело
4	4	Туризм
5	5	Реклама

Рисунок 14 – Результат выполнения запроса к БД для получения первых 5 специальностей

1 row	
	count
1	30

Рисунок 15 – Результат выполнения запроса к БД для получения общего количества записей в таблице

Данные рисунка 14 совпадают с данными, содержащимися в ответе метода получения списка всех специальностей с учетом постраничной выдачи. Количество элементов, полученных в параметре ответа `totalElements`, равное 30, совпадает с результатом запроса общего количества записей в таблице на рисунке 15.

3.2 Проверка метода поиска специальности по названию

- в Swagger UI выбрать вкладку «Специальности», метод «Поиск специальности по названию», которые представлены на рисунке 16,
- в параметр `name` ввести название специальности «Реклама»,
- выполнить запрос, результат выполнения представлен на рисунке 17,
- получить ответ с идентификатором и названием искомой специальности.

Специальности	
GET	/profession/search Поиск специальности по названию
GET	/profession/list Получение списка всех доступных специальностей
GET	/profession/institution/list Поиск образовательных организаций, реализующих конкретную специальность

Рисунок 16 – Метод «Поиск специальности по названию»

Code	Details
200	<p>Response body</p> <pre>{ "id": 5, "name": "Реклама" }</pre>

Рисунок 17 – Результат работы метода «Поиск специальности по названию»

Сравним полученные данные с данными, хранящимися в БД. Для этого выполним следующие действия:

- выполнить в СУБД запрос, приведенный ниже. Результат запроса поиска представлен на рисунке 18,
- сравнить результаты.

Запрос к БД:

```
select * from profession where name like 'Реклама'
```

Данные при выполнении метода поиска специальности по названию совпадают с данными из БД.

1 row		
	id	name
1	5	Реклама

Рисунок 18 – Результат запроса поиска специальности «Реклама»

Выполним поиск специальности, которой не должно быть в системе:

- в параметр name запроса поиска специальности по названию ввести название специальности, которого в БД быть не должно, например, «Тест», результат работы представлен на рисунке 19,
- получить ошибку с HTTP-кодом 404 – данные не найдены.

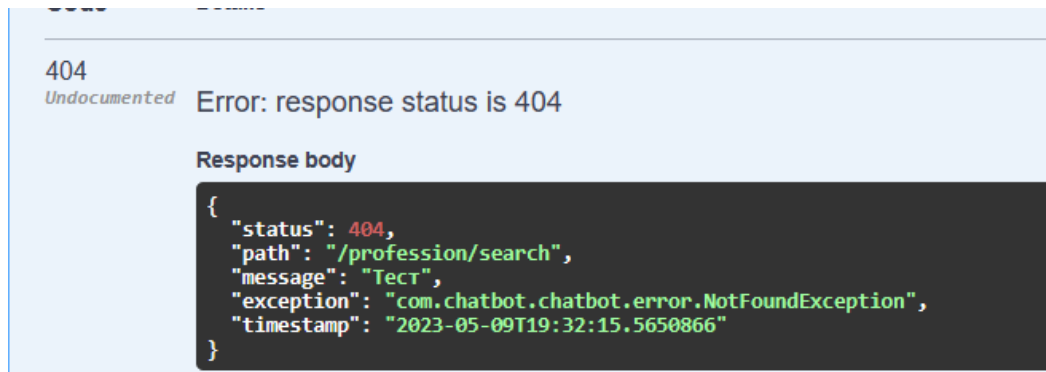


Рисунок 19 – Результат работы метода «Поиск специальности по названию»

Проверим отсутствие данных в БД следующими шагами:

- выполнить в СУБД запрос, приведенный ниже. Результат запроса поиска представлен на рисунке 20,
- сравнить результаты.

Запрос к БД:

```
select * from profession where name like 'Тест'
```

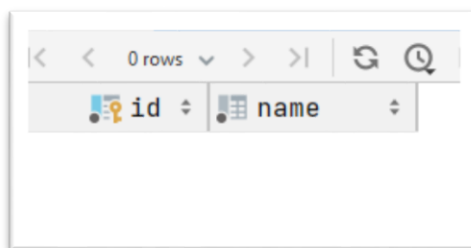


Рисунок 20 – Результат запроса поиска специальности «Тест»

По запросу в БД данных, соответствующих названию специальности «Тест» не найдено. Метод поиска специальности по названию вернул ошибку NotFoundException с HTTP-кодом 404, что означает отсутствие таких записей в БД. Метод работает корректно.

3.3 Проверка метода поиска ПОО

- в Swagger UI выбрать вкладку «Специальности», метод «Поиск образовательных организаций, реализующих конкретную специальность», которые представлены на рисунке 21,
- в параметр `professionId` ввести идентификатор искомой специальности, например, специальности «Реклама», которая имеет идентификатор 5,
- задать размер страницы равный 10 для проверки постраничного отображения,
- задать номер страницы выдачи 0,
- получить ответ с идентификаторами и названиями ПОО, которые реализуют данные специальности.

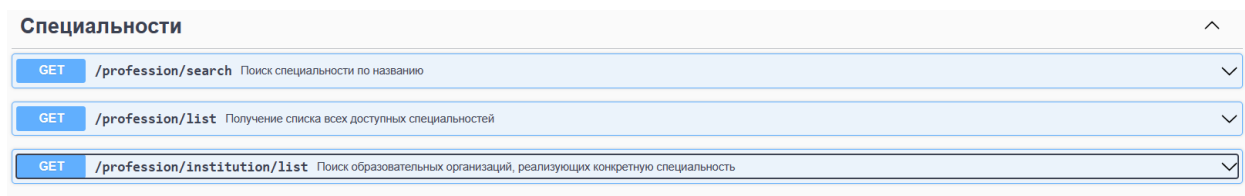


Рисунок 21 – Метод «Поиск образовательных организаций, реализующих конкретную специальность»

Ответ, полученный в результате выполнения запроса:

```
{
  "content": [
    {
      "id": 1,
      "instName": "КАДР №26"
    },
    {
      "id": 14,
      "instName": "ТК №34"
    }
  ],
}
```

```
"pageable": {  
  "sort": {  
    "sorted": true,  
    "unsorted": false,  
    "empty": false  
  },  
  "offset": 0,  
  "pageSize": 10,  
  "pageNumber": 0,  
  "paged": true,  
  "unpaged": false  
},  
"last": true,  
"totalElements": 2,  
"totalPages": 1,  
"size": 10,  
"number": 0,  
"sort": {  
  "sorted": true,  
  "unsorted": false,  
  "empty": false  
},  
"first": true,  
"numberOfElements": 2,  
"empty": false  
}
```

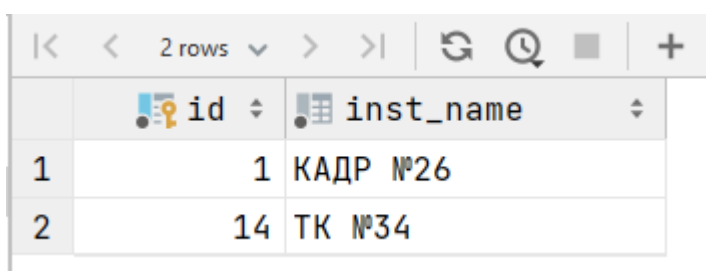
Проверим соответствие данных ответа данным, хранящимся в БД, выполнив следующие шаги:

– выполним запрос к БД, приведенный ниже, для получения идентификаторов и названий всех ПОО, которые реализуют данную специальность. Результат выполнения запроса представлен на рисунке 22,

– сравнить результаты.

Запрос к БД:

```
select institution.id, institution.inst_name
from institution
      left join prof2inst p2i on institution.id =
p2i.institution_id
where prof_id = 5;
```



	id	inst_name
1	1	КАДР №26
2	14	ТК №34

Рисунок 22 – Результат выполнения запроса поиска всех ПОО, реализующих специальность

Данные рисунка 22 совпадают с данными, содержащимися в ответе метода получения всех ПОО, реализующих конкретную специальность с учетом постраничной выдачи. Количество элементов, полученных в параметре ответа `totalElements`, равное 2, совпадает с количеством строк результата запроса к БД.

3.4 Проверка метода получения списка доступных ПОО

Убедимся, что данные, предоставляемые в ответе метода соответствуют данным, хранящимся в базе данных. Также проверит корректность работы постраничной выдачи данных.

Для проверки выполним следующие действия:

– в Swagger UI выбрать вкладку «Образовательные организации», метод «Получение списка доступных образовательных организаций», которые представлены на рисунке 23,

– задать размер страницы выдачи равным 5,

- задать номер страницы выдачи 0,
- выполнить запрос,
- получить список образовательных организаций.

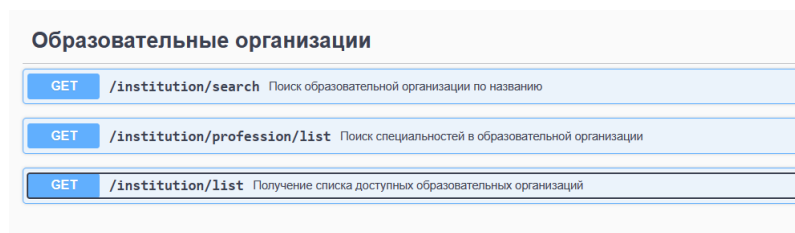


Рисунок 23 – Метод «Получение списка доступных образовательных организаций»

Ответ на запрос имеет следующий вид:

```
{
  "content": [
    {
      "id": 1,
      "instName": "КАДР №26"
    },
    {
      "id": 2,
      "instName": "ДЗМ №1"
    },
    {
      "id": 3,
      "instName": "КП №11"
    },
    {
      "id": 4,
      "instName": "МКАГ"
    },
    {
      "id": 5,
```

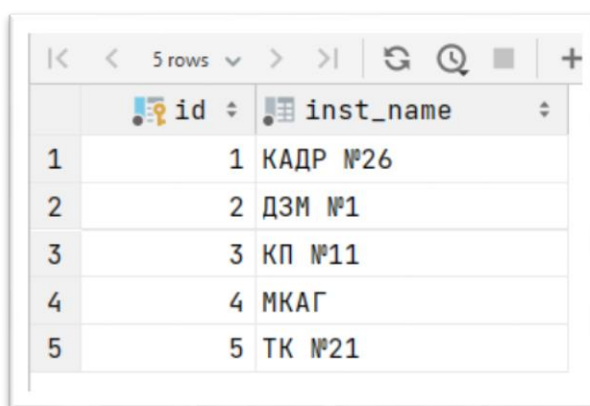
```
    "instName": "TK №21"
  }
],
"pageable": {
  "sort": {
    "sorted": true,
    "unsorted": false,
    "empty": false
  },
  "offset": 0,
  "pageSize": 5,
  "pageNumber": 0,
  "paged": true,
  "unpaged": false
},
"last": false,
"totalElements": 16,
"totalPages": 4,
"size": 5,
"number": 0,
"sort": {
  "sorted": true,
  "unsorted": false,
  "empty": false
},
"first": true,
"numberOfElements": 5,
"empty": false
}
```

Выполним проверку соответствия данных в ответе и данных, хранящихся в БД, выполнив следующие шаги:

- выполнить запрос в СУБД для получения списка первых 5 ПОО, приведенный ниже. Результат выполнения запроса представлен на рисунке 24,
- выполнить запрос в СУБД для получения общего количества ПОО в БД. Результат выполнения запроса представлен на рисунке 25,
- сравнить результаты работы метода и результаты запросов в БД.

Запрос к базе данных:

```
select * from institution limit 5;
```

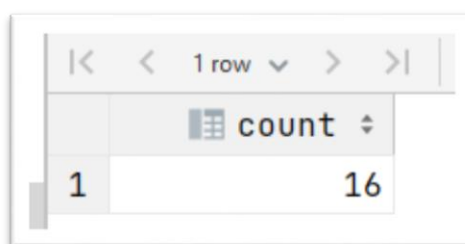


	id	inst_name
1	1	КАДР №26
2	2	ДЗМ №1
3	3	КП №11
4	4	МКАГ
5	5	ТК №21

Рисунок 24 – Результат выполнения запроса получения списка ПОО

Запрос к БД:

```
select count(*) from institution;
```



	count
1	16

Рисунок 25 – Результат выполнения запроса к БД на получение общего количества ПОО

Данные рисунка 24 совпадают с данными, содержащимися в ответе метода получения всех ПОО с учетом постраничной выдачи. Количество элементов, полученных в параметре ответа `totalElements`, равное 16, совпадает с результатов выполнения SQL-запроса на получение количества записей.

3.5 Проверка метода поиска ПОО по названию

- в Swagger UI выбрать вкладку «Образовательные организации», метод «Поиск образовательной организации по названию», которые представлены на рисунке 26,
- в параметр name ввести название ПОО «МКАГ»,
- выполнить запрос, результат выполнения представлен на рисунке 27,
- получить ответ с идентификатором и названием искомой специальности.

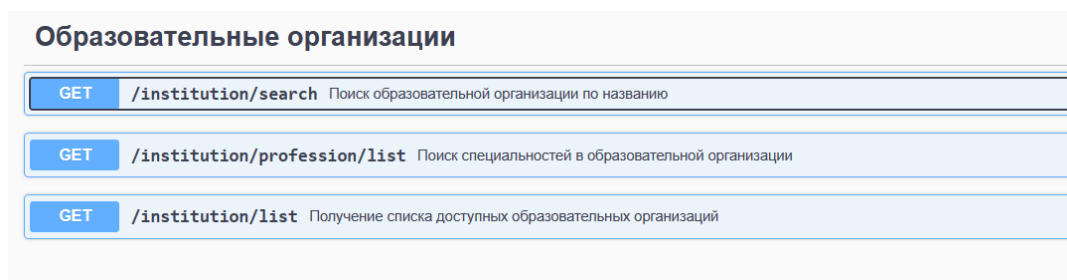


Рисунок 26 – Метод «Поиск образовательной организации по названию»

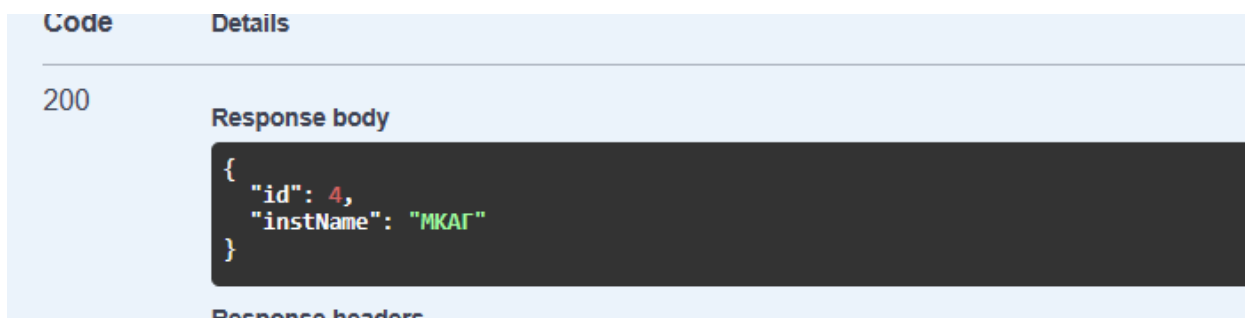


Рисунок 27 – Результат работы метода «Поиск образовательной организации по названию»

Выполним сравнение данных, полученных в ответе метода и данных, хранящихся в БД:

- выполнить в СУБД запрос, приведенный ниже. Результат запроса поиска представлен на рисунке 28,
- сравнить результаты.

Запрос к БД:

```
select * from institution where inst_name like 'МКАГ';
```

	id	inst_name
1	4	МКАГ

Рисунок 28 – Результат запроса поиска образовательной организации «МКАГ»

Данные при выполнении метода поиска специальности по названию совпадают с данными из БД.

Проведем поиск специальности, которой не должно быть в системе:

- в параметр name запроса поиска образовательной организации по названию ввести название ПОО, которого в БД быть не должно, например, «Тест», результат работы представлен на рисунке 29,
- получить ошибку с HTTP-кодом 404 – данные не найдены.

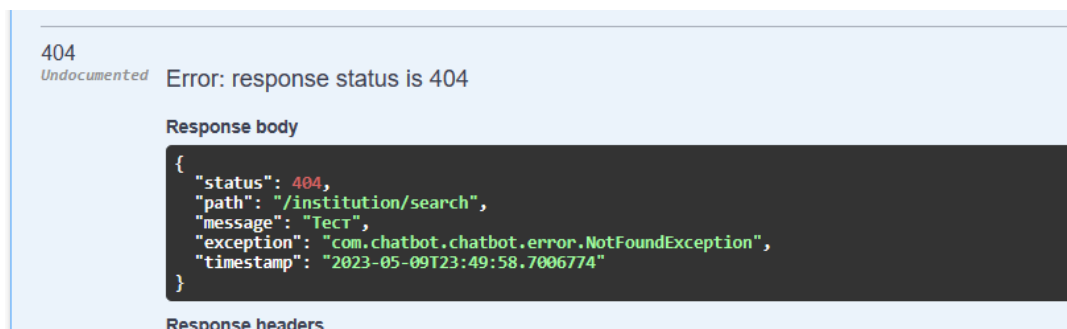


Рисунок 29 – Результат работы метода «Поиск образовательной организации по названию»

- выполнить в СУБД запрос, приведенный ниже. Результат запроса поиска представлен на рисунке 30,
- сравнить результаты.

Запрос к БД:

```
select * from institution where inst_name 'Тест';
```

	id	inst_name
0 rows		

Рисунок 30 – Результат запроса поиска образовательной организации «Тест»

По запросу в БД данных, соответствующих названию ПОО «Тест» не найдено. Метод поиска образовательной организации по названию вернул ошибку NotFoundException с HTTP-кодом 404, что означает отсутствие таких записей в БД. Метод работает корректно.

3.6 Проверка метода поиска специальностей в ПОО

- в Swagger UI выбрать вкладку «Образовательные организации», метод «Поиск специальностей в образовательной организации», которые представлены на рисунке 31,

- в параметр `institutionId` ввести идентификатор ПОО, например, ПОО «МКАГ», которая имеет идентификатор 4,

- задать размер страницы равный 10 для проверки постраничного отображения,

- задать номер страницы выдачи 0,

- получить ответ с идентификаторами и названиями специальностей, которые реализуются в данном ПОО.

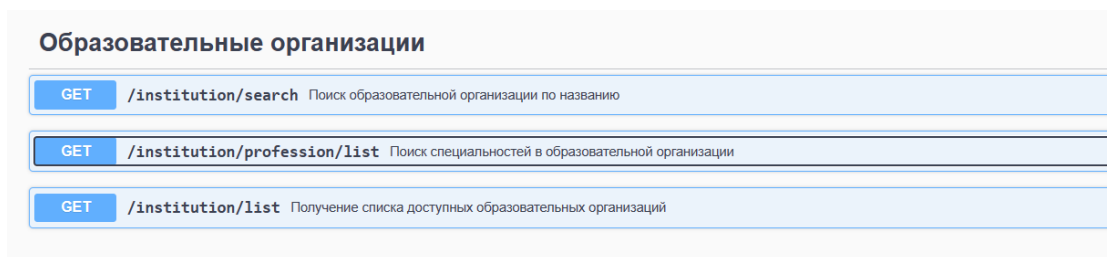


Рисунок 31 – Метод «Поиск специальностей в образовательной организации»

Ответ, полученный в ходе выполнения запроса:

```
{
  "content": [
    {
      "id": 12,
      "name": "Программирование"
    },
    {
```

```
"id": 14,  
  "name": "Ювелир"  
},  
{  
  "id": 15,  
  "name": "Архитектура"  
},  
{  
  "id": 16,  
  "name": "Сварщик"  
},  
{  
  "id": 19,  
  "name": "Дизайн"  
}  
],  
"pageable": {  
  "sort": {  
    "sorted": true,  
    "unsorted": false,  
    "empty": false  
  },  
  "offset": 0,  
  "pageSize": 10,  
  "pageNumber": 0,  
  "paged": true,  
  "unpaged": false  
},  
"last": true,
```

```

"totalElements": 5,
"totalPages": 1,
"size": 10,
"number": 0,
"sort": {
  "sorted": true,
  "unsorted": false,
  "empty": false
},
"first": true,
"numberOfElements": 5,
"empty": false
}

```

Выполним сравнение данных, полученных в результате выполнения запроса и данных, хранящихся в БД:

– выполним запрос к БД, приведенный ниже, для получения идентификаторов и названий всех специальностей, которые реализуются в данном ПОО:

Результат выполнения запроса представлен на рисунке 32,

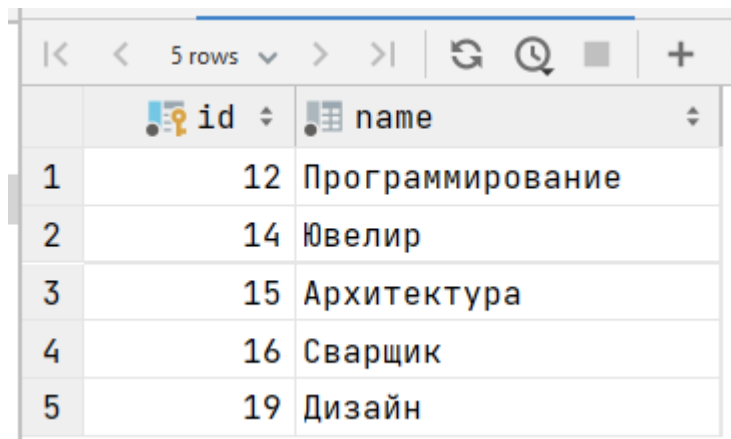
– сравнить результаты.

Запрос к БД:

```

select profession.id, profession.name
from profession
        join  prof2inst  p2i  on  profession.id  =
p2i.prof_id
where institution_id = 4;

```



	id	name
1	12	Программирование
2	14	Ювелир
3	15	Архитектура
4	16	Сварщик
5	19	Дизайн

Рисунок 32 – Результат выполнения запроса поиска всех специальностей МКАГ

Данные рисунка 32 совпадают с данными, содержащимися в ответе метода получения всех специальностей, которые реализует ПОО с учетом постраничной выдачи. Количество элементов, полученных в параметре ответа `totalElements`, равное 5, совпадает с количеством строк результата запроса к БД.

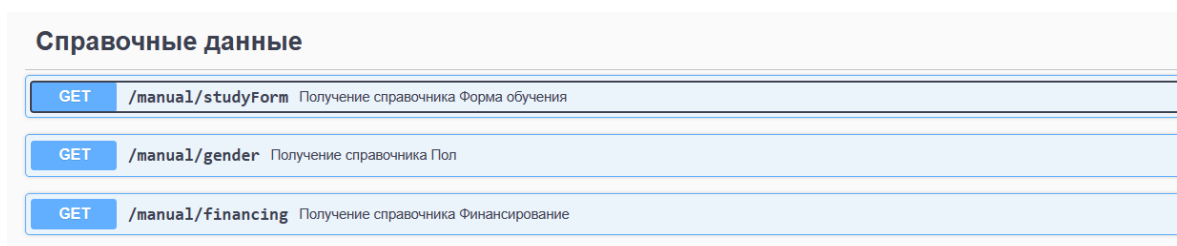
3.7 Проверка метода получения справочника «Форма обучения»

Справочные данные «Форма обучения» включает в себя следующие значения:

- 0 – заочно,
- 1 – очно.

Для проверки выполним следующие действия:

- в Swagger UI выбрать вкладку «Справочные данные», метод «Получение справочника «Форма обучения»», которые представлены на рисунке 33,
- выполнить запрос, результат выполнения представлен на рисунке 34,
- сравнить полученные данные с ожидаемыми справочными данными.



Справочные данные	
GET	/manual/studyForm Получение справочника Форма обучения
GET	/manual/gender Получение справочника Пол
GET	/manual/financing Получение справочника Финансирование

Рисунок 33 – Метод «Получение справочника «Форма обучения»»

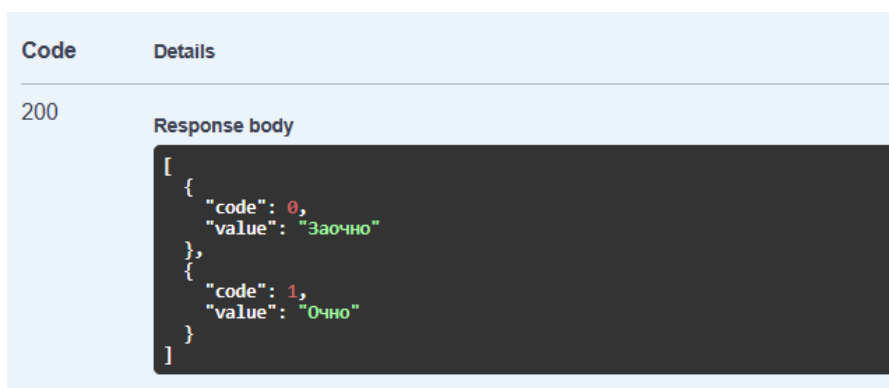


Рисунок 34 – Результат выполнения метода «Получение справочника «Форма обучения»»

Результат работы запроса совпадает с ожидаемыми справочными данными, метод работает корректно.

3.8 Проверка метода получения справочника «Пол»

Справочные данные «Пол» включает в себя следующие значения:

- 0 – мужской,
- 1 – женский.

Для проверки выполним следующие действия:

- в Swagger UI выбрать вкладку «Справочные данные», метод «Получение справочника «Пол»», которые представлены на рисунке 35,
- выполнить запрос, результат выполнения представлен на рисунке 36,
- сравнить полученные данные с ожидаемыми справочными данными.

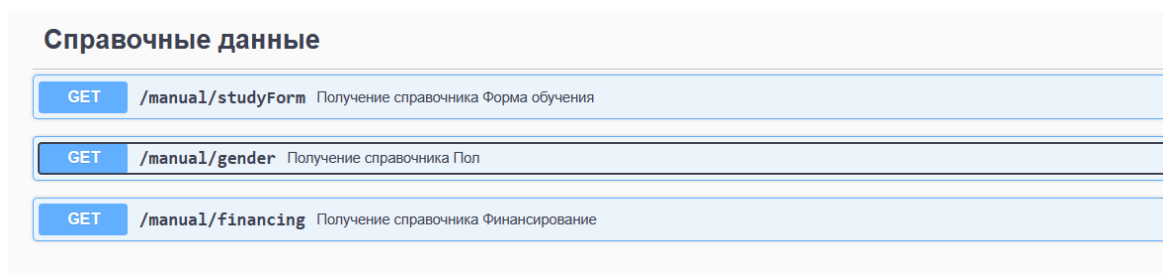


Рисунок 35 – Метод «Получение справочника «Пол»»



Рисунок 36 – Результат выполнения метода «Получение справочника «Пол»»

Результат работы запроса совпадает с ожидаемыми справочными данными, метод работает корректно.

3.9 Проверка метода получения справочника «Финансирование»

Справочные данные «Финансирование» включает в себя следующие значения:

- 0 – на договорной основе,
- 1 – бюджет.

Для проверки выполним следующие действия:

– в Swagger UI выбрать вкладку «Справочные данные», метод «Получение справочника «Финансирование»», которые представлены на рисунке 37,

- выполнить запрос, результат выполнения представлен на рисунке 38,
- сравнить полученные данные с ожидаемыми справочными данными.

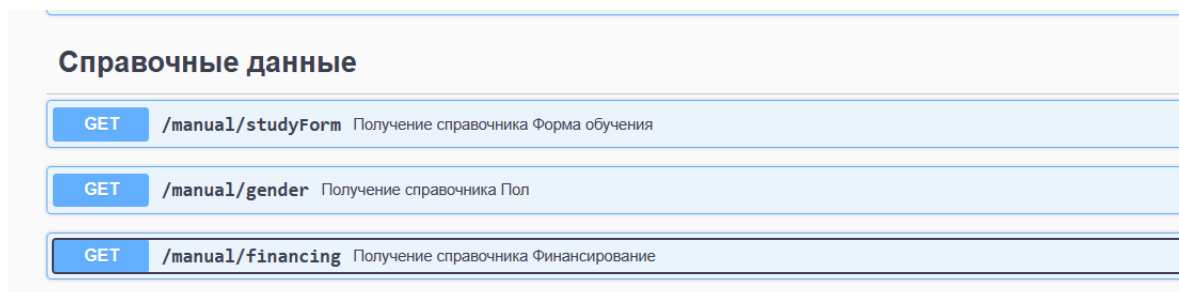


Рисунок 37 – Метод «Получение справочника «Финансирование»»

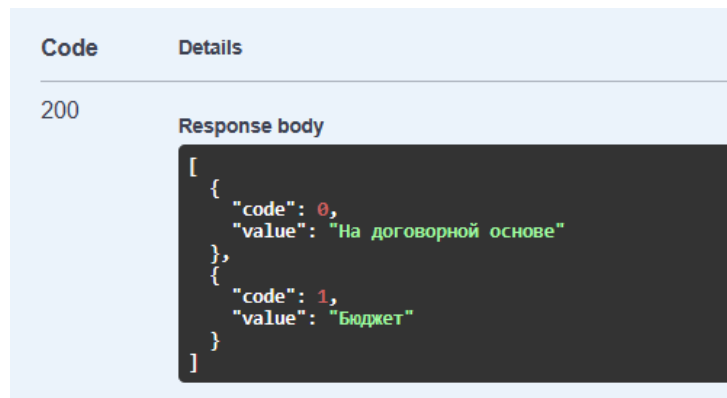


Рисунок 38 – Результат выполнения метода «Получение справочника «Финансирование»»

Результат работы запроса совпадает с ожидаемыми справочными данными, метод работает корректно.

3.10 Проверка метода поиска адреса

- в Swagger UI выбрать вкладку «Заявления», метод «Поиск адреса», которые представлены на рисунке 39,
- в параметр address ввести следующий адрес «пушкина»,
- выполнить запрос,
- убедиться в соответствии данных ответа введенной строке.

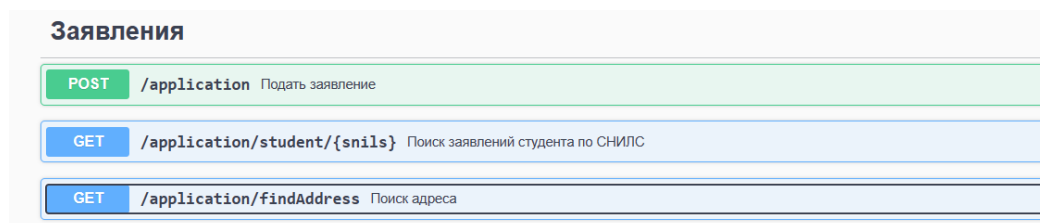


Рисунок 39 – Метод «Поиск адреса»

Ответ, полученный в результате вызова метода:

```

[
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "Проезд Пушкина",
    "building": "2",

```

```
"index": "440000"
},
{
  "region": "Пензенская область",
  "district": "городской округ город Пенза",
  "locality": "город Пенза",
  "street": "Проезд Пушкина",
  "building": "3",
  "index": "440000"
},
{
  "region": "Пензенская область",
  "district": "городской округ город Пенза",
  "locality": "город Пенза",
  "street": "Улица Пушкина",
  "building": "1",
  "index": "440008"
},
{
  "region": "Пензенская область",
  "district": "городской округ город Пенза",
  "locality": "город Пенза",
  "street": "Улица Пушкина",
  "building": "7",
  "index": "440008"
},
{
  "region": "Пензенская область",
  "district": "городской округ город Пенза",
```

```
"locality": "город Пенза",  
"street": "Улица Пушкина",  
"building": "8",  
"index": "440008"  
}  
]
```

В параметре ответа `street` содержатся данные, содержащие значения «Улица Пушкина» и «Проезд Пушкина», что удовлетворяет условиям запроса.

Введем новый адрес для поиска и убедимся в корректности работы:

- в параметр `address` ввести следующий адрес «пушкина 3»,
- выполнить запрос,
- убедиться в соответствии данных ответа введенной строке.

Ответ, полученный в результате вызова метода:

```
[  
{  
  "region": "Пензенская область",  
  "district": "городской округ город Пенза",  
  "locality": "город Пенза",  
  "street": "Проезд Пушкина",  
  "building": "3",  
  "index": "440000"  
},  
{  
  "region": "Пензенская область",  
  "district": "городской округ город Пенза",  
  "locality": "город Пенза",  
  "street": "Улица Бадигина",  
  "building": "3",  
  "index": "440067"
```

```
},  
{  
  "region": "Пензенская область",  
  "district": "городской округ город Пенза",  
  "locality": "город Пенза",  
  "street": "Улица Бакунина",  
  "building": "3",  
  "index": "440000"  
},  
{  
  "region": "Пензенская область",  
  "district": "городской округ город Пенза",  
  "locality": "город Пенза",  
  "street": "Улица Пушкина",  
  "building": "3",  
  "index": "440056"  
},  
{  
  "region": "Пензенская область",  
  "district": "городской округ город Пенза",  
  "locality": "город Пенза",  
  "street": "Проезд Пушкина",  
  "building": "2",  
  "index": "440000"  
}  
]
```

Ответ метода содержит 5 наиболее релевантных ответов, первый из которых в параметре `street` содержит значение «Проезд Пушкина», а в параметре `building` – значение «3», что соответствует строке запроса.

Введем новый адрес для поиска и убедимся в корректности работы:

- в параметр address ввести следующий адрес «проезд 2 малиновый»,
- выполнить запрос,
- убедиться в соответствии данных ответа введенной строке.

Ответ, полученный в результате вызова запроса:

```
[
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "10-й Малиновый проезд",
    "building": "2",
    "index": "440015"
  },
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "10-й Малиновый проезд",
    "building": "5",
    "index": "440015"
  },
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "10-й Малиновый проезд",
    "building": "6",
    "index": "440015"
  }
]
```

```

},
{
  "region": "Пензенская область",
  "district": "городской округ город Пенза",
  "locality": "город Пенза",
  "street": "10-й Малиновый проезд",
  "building": "7",
  "index": "440015"
},
{
  "region": "Пензенская область",
  "district": "городской округ город Пенза",
  "locality": "город Пенза",
  "street": "10-й Малиновый проезд",
  "building": "10",
  "index": "440015"
}
]

```

Ответ метода содержит 5 наиболее релевантных ответов, первый из которых в параметре `street` содержит значение «10-й Малиновый проезд», а в параметре `building` – значение «2», что соответствует строке запроса.

3.11 Проверка метода подачи заявления

Проверки на заполнение обязательных полей:

- в Swagger UI выбрать вкладку «Заявления», метод «Подать заявление», которые представлены на рисунке 40,
- не заполнять поле `institutionId`,
- выполнить запрос, результат выполнения представлен на рисунке 41,
- получить ошибку,
- не заполнять поле `professionId`,

- выполнить запрос, результат выполнения представлен на рисунке 42,
- получить ошибку,
- не заполнять поле financing,
- выполнить запрос, результат выполнения представлен на рисунке 43,
- получить ошибку,
- не заполнять поле studyForm,
- выполнить запрос, результат выполнения представлен на рисунке 44,
- получить ошибку,
- не заполнять объект document,
- выполнить запрос, результат выполнения представлен на рисунке 45,
- получить ошибку,
- не заполнять объект student,
- выполнить запрос, результат выполнения представлен на рисунке 46,
- получить ошибку.

Заявления		
POST	/application	Подать заявление
GET	/application/student/{snils}	Поиск заявлений студента по СНИЛС
GET	/application/findAddress	Поиск адреса

Рисунок 40 – Метод «Подать заявление»

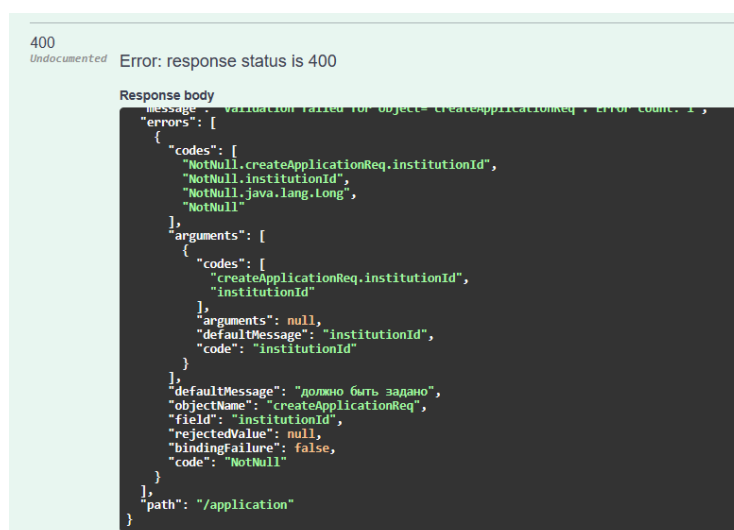


Рисунок 41 – Ошибка при пустом параметре institutionId

Получена ошибка с HTTP-кодом 400 и сообщение об обязательности поля institutionId.

```
400
Undocumented Error: response status is 400

Response body
{
  "message": "Validation failed for object= createApplicationReq",
  "errors": [
    {
      "codes": [
        "NotNull.createApplicationReq.professionId",
        "NotNull.professionId",
        "NotNull.java.lang.Long",
        "NotNull"
      ],
      "arguments": [
        {
          "codes": [
            "createApplicationReq.professionId",
            "professionId"
          ],
          "arguments": null,
          "defaultMessage": "professionId",
          "code": "professionId"
        }
      ],
      "defaultMessage": "должно быть задано",
      "objectName": "createApplicationReq",
      "field": "professionId",
      "rejectedValue": null,
      "bindingFailure": false,
      "code": "NotNull"
    }
  ],
  "path": "/application"
}
```

Рисунок 42 – Ошибка при пустом параметре professionId

Получена ошибка с HTTP-кодом 400 и сообщение об обязательности поля professionId.

```
400
Undocumented Error: response status is 400

Response body
{
  "message": "Validation failed for object= createApplicationReq",
  "errors": [
    {
      "codes": [
        "NotNull.createApplicationReq.financing",
        "NotNull.financing",
        "NotNull.java.lang.Integer",
        "NotNull"
      ],
      "arguments": [
        {
          "codes": [
            "createApplicationReq.financing",
            "financing"
          ],
          "arguments": null,
          "defaultMessage": "financing",
          "code": "financing"
        }
      ],
      "defaultMessage": "должно быть задано",
      "objectName": "createApplicationReq",
      "field": "financing",
      "rejectedValue": null,
      "bindingFailure": false,
      "code": "NotNull"
    }
  ],
  "path": "/application"
}
```

Рисунок 43 – Ошибка при пустом параметре financing

Получена ошибка с HTTP-кодом 400 и сообщение об обязательности поля financing.

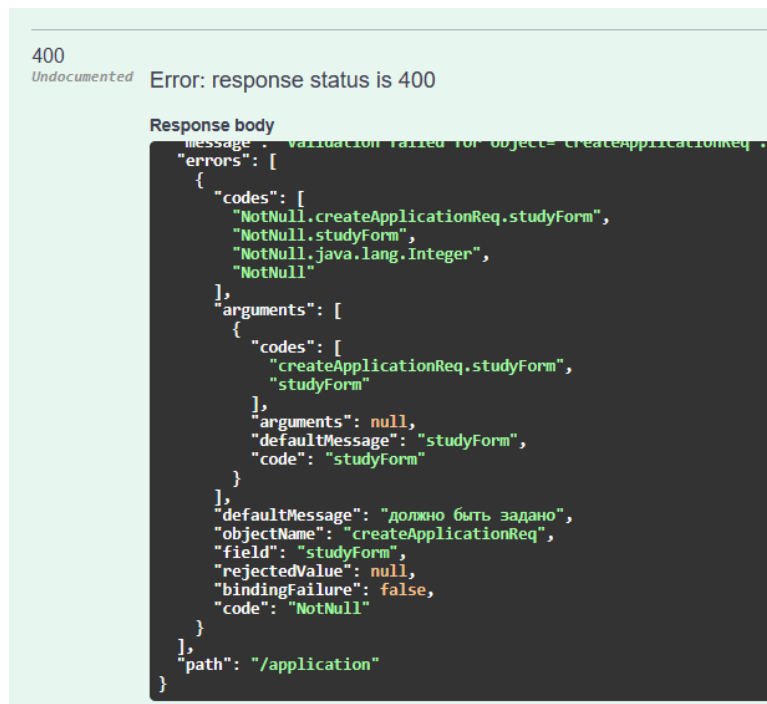


Рисунок 44 – Ошибка при пустом параметре studyForm

Получена ошибка с HTTP-кодом 400 и сообщение об обязательности поля studyForm.

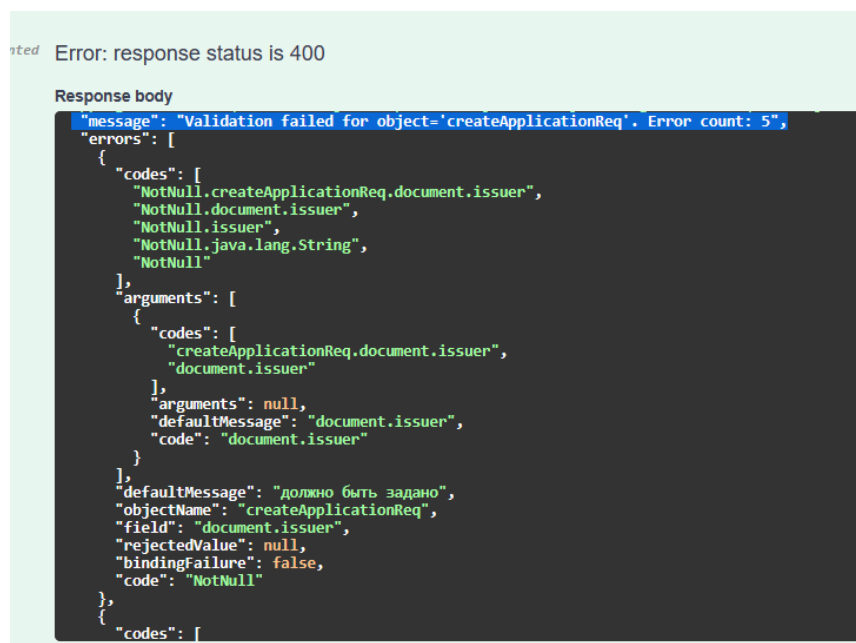


Рисунок 45 – Ошибка при пустом объекте document

При валидации найдено 5 ошибок, выделено на рисунке 45, так как все поля объекта переданы пустыми и являются обязательными к заполнению.

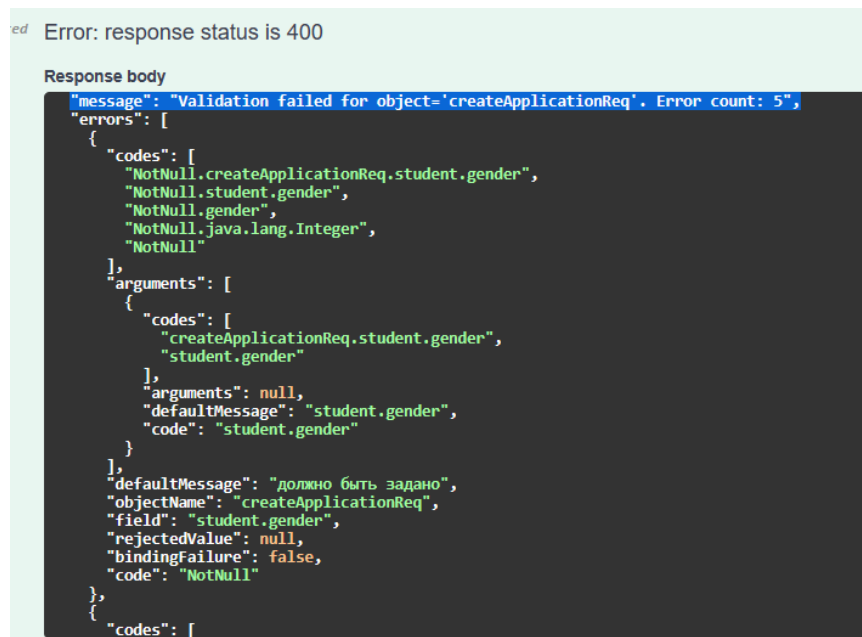


Рисунок 46 – Ошибка при пустом объекте student

При валидации найдено 5 ошибок, выделено на рисунке 46, так как все поля объекта переданы пустыми и являются обязательными к заполнению.

Проверим на соответствие переданных значений БД и справочным данным:

- заполнить поле `institutionId` значением, которого нет в БД, например, - 1,
- выполнить запрос, результат выполнения представлен на рисунке 47,
- получить ошибку,
- заполнить поле `professionId` значением, которого нет в БД, например, - 1,
- выполнить запрос, результат выполнения представлен на рисунке 48,
- получить ошибку,
- заполнить поля `professionId` и `institutionId` так, чтобы ПОО со значением идентификатора `institutionId` не реализовывала специальность со значением идентификатора `professionId`, например, `professionId = 1` и `institutionId = 1`,
- выполнить запрос, результат выполнения представлен на рисунке 49,
- получить ошибку,

- заполнить поле `financing` значением не из справочных данных, например, 2,
- выполнить запрос, результат выполнения представлен на рисунке 50,
- получить ошибку,
- заполнить поле `studyForm` значением не из справочных данных, например, 2,
- выполнить запрос, результат выполнения представлен на рисунке 51,
- получить ошибку,
- заполнить поле `student.gender` значением не из справочных данных, например, 2,
- выполнить запрос, результат выполнения представлен на рисунке 52,
- получить ошибку.

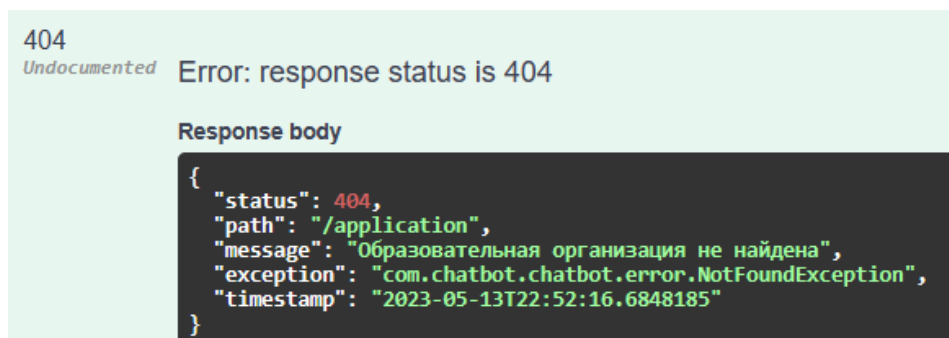


Рисунок 47 – Образовательная организация не найдена

В ответ на запрос получена ошибка с HTTP-кодом 404 и сообщением «Образовательная организация не найдена».

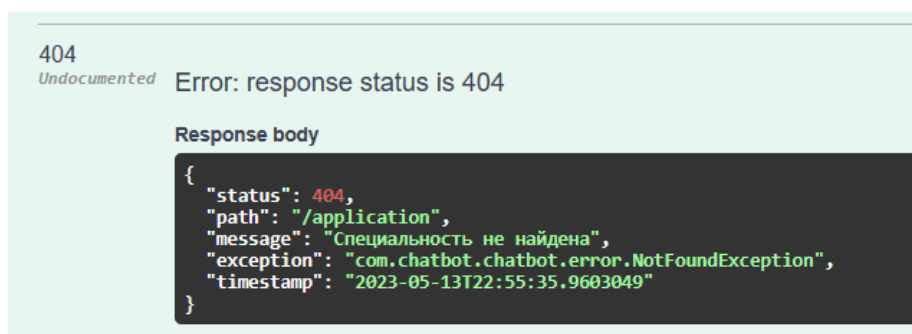


Рисунок 48 – Специальность не найдена

В ответ на запрос получена ошибка с HTTP-кодом 404 и сообщением «Специальность не найдена».

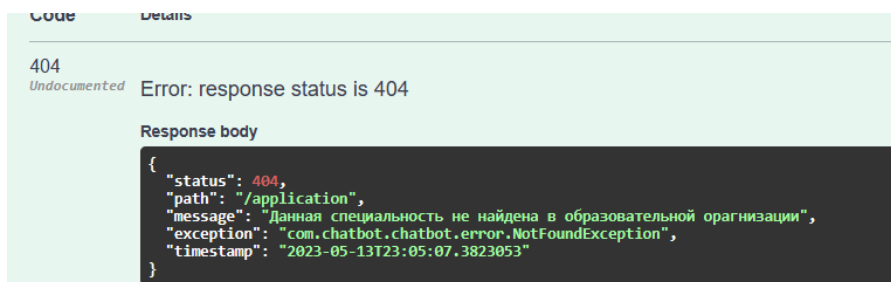


Рисунок 49 – Данная специальность не найдена в образовательной организации

В ответ на запрос получена ошибка с HTTP-кодом 404 и сообщением «Данная специальность не найдена в образовательной организации».

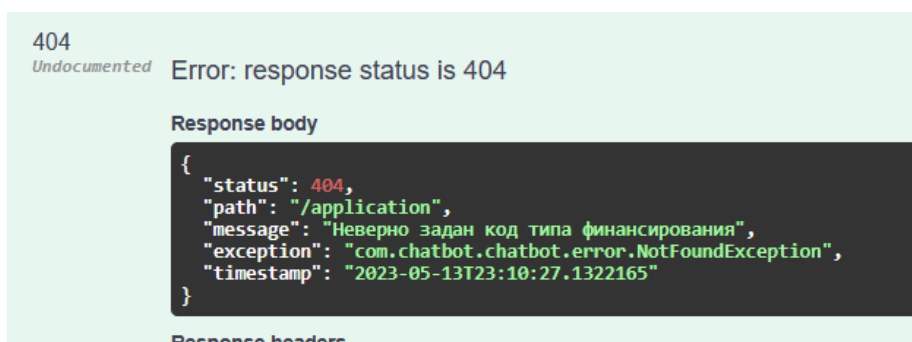


Рисунок 50 – Неверно задан код типа финансирования

В ответ на запрос получена ошибка с HTTP-кодом 404 и сообщением «Неверно задан код типа финансирования».

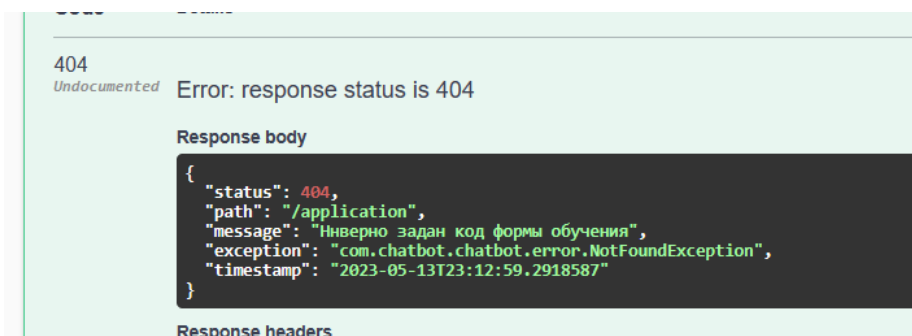


Рисунок 51 – Неверно задан код формы обучения

В ответ на запрос получена ошибка с HTTP-кодом 404 и сообщением «Неверно задан код формы обучения».

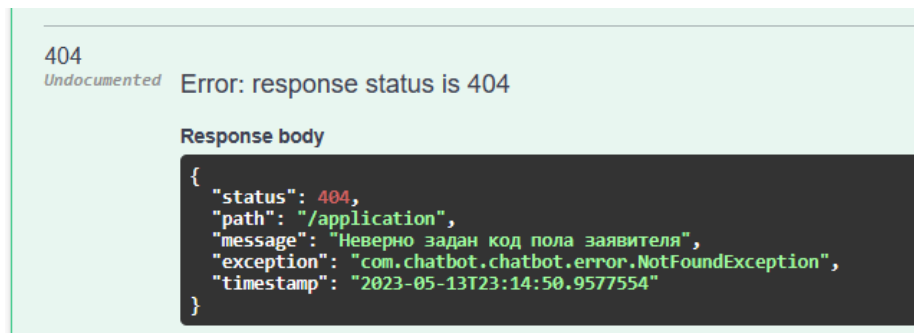


Рисунок 52 – Неверно задан код пола заявителя

В ответ на запрос получена ошибка с HTTP-кодом 404 и сообщением «Неверно задан код пола заявителя».

Проверки серии и номера бланка паспорта РФ:

- заполнить поле `document.serial` некорректным значением, например, «55f»,
- выполнить запрос, результат выполнения представлен на рисунке 53,
- получить ошибку,
- заполнить поле `document.number` некорректным значением, например, «00aa»,
- выполнить запрос, результат выполнения представлен на рисунке 54,
- получить ошибку.

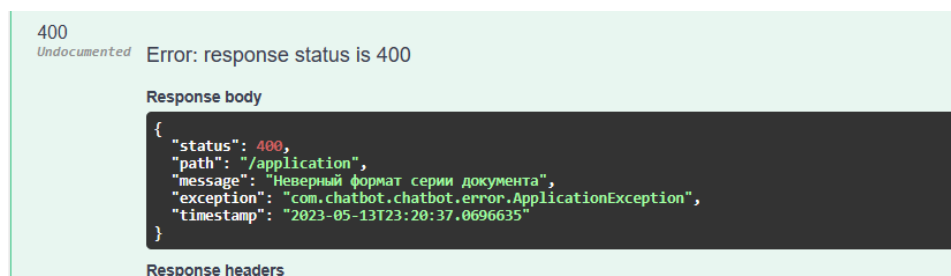


Рисунок 53 – Неверный формат серии документа

В ответ на запрос получена ошибка с HTTP-кодом 400 и сообщением «Неверный формат серии документа».

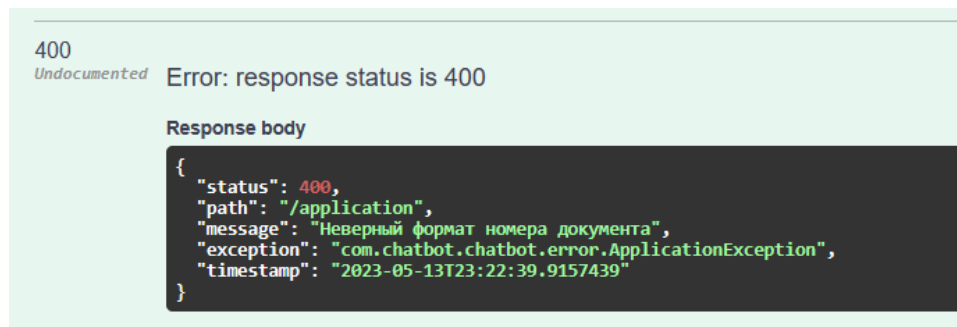


Рисунок 54 – Неверный формат номера документа

В ответ на запрос получена ошибка с HTTP-кодом 400 и сообщением «Неверный формат номера документа».

Проверка сохранения корректных данных:

- заполнить тело запроса на подачу заявления корректными данными,
- выполнить запрос, результат выполнения представлен на рисунке 55,
- успешное сохранение.

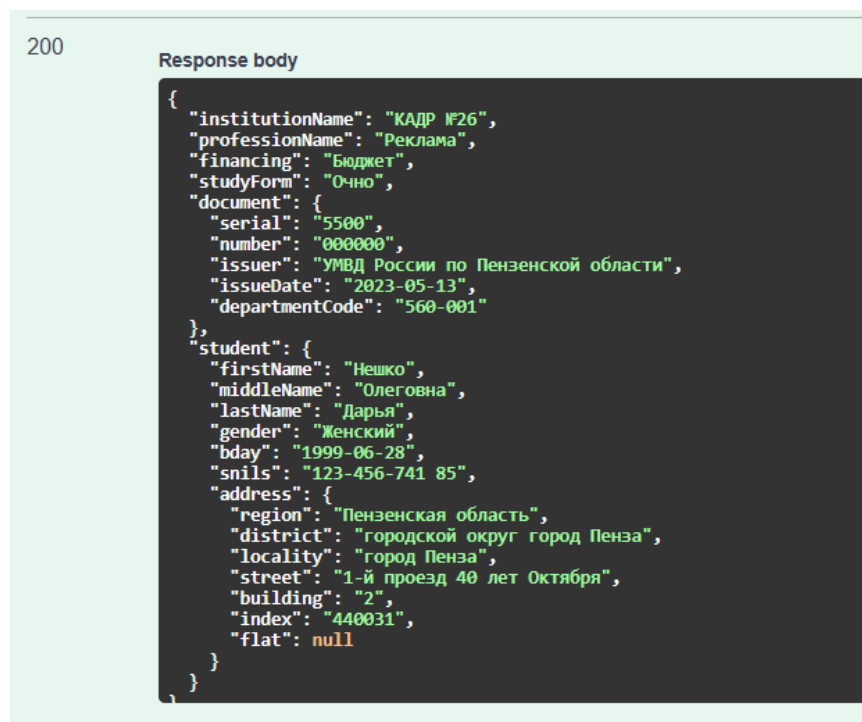


Рисунок 55 – Заявление успешно сохранено

В ответ на запрос получены с HTTP-код 200 и тело ответа с данными заявления. Метод работает корректно.

3.12 Проверка метода поиска заявлений студента по СНИЛС

- в Swagger UI выбрать вкладку «Заявления», метод «Поиск заявлений студента по СНИЛС», которые представлены на рисунке 56,
- в параметр snils ввести значение документа, например, 111 – 111 – 111 - 00,
- выполнить запрос, результат запроса представлен на рисунке 57,
- убедиться, что получена ошибка NotFoundException с HTTP-кодом 404, что соответствует отсутствию данных в БД.

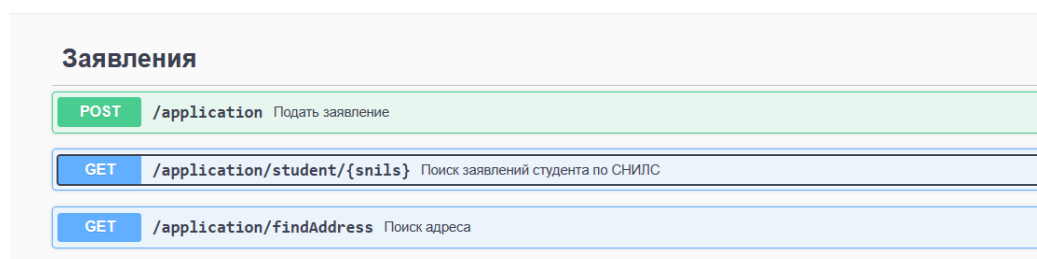


Рисунок 56 – Метод «Поиск заявлений студента по СНИЛС»



Рисунок 57 – Результат работы метода поиска заявлений с кодом 404

Результат работы метода совпадает с ожидаемым результатов: получена ошибка NotFoundException с HTTP-кодом 404 и сообщением об ошибке «Не найдены заявления по студенту».

Далее рассмотрим поиск заявления, поданного в пункте 3.11:

- в параметр snils ввести значение из пункта 3.11 – 123-456-741 85,
- выполнить запрос, результат запроса представлен на рисунке 58,
- убедиться, что данные найдены и соответствуют данным, сохраненным в пункте 3.11.

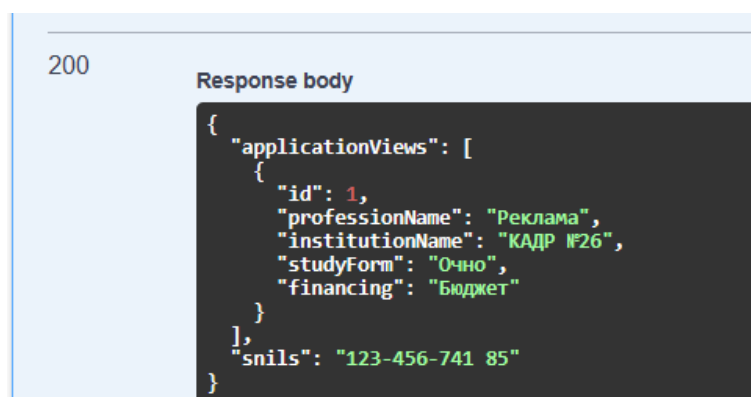


Рисунок 58 – Успешный поиск заявления по СНИЛС

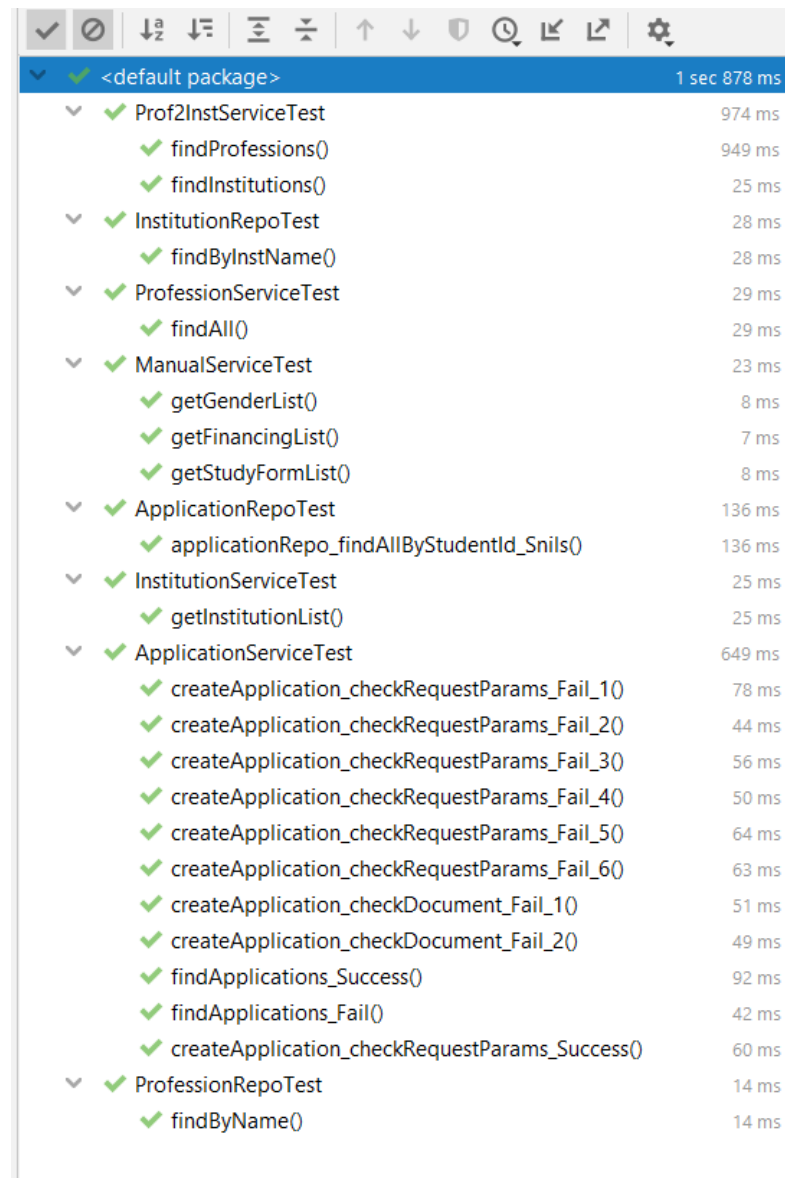
3.13 Модульное тестирование

Модульные тесты покрывают все методы приложения, проверяют методы как на корректных данных для получения успешного результата выполнения, так и на некорректных данных, для проверки получения ошибок.

Отчет об успешном прохождении модульных тестов представлен на рисунке 59.

3.14 Выводы

Проведенные функциональное и модульное тестирования доказали соответствие программного продукта заявленным требованиям. Программное обеспечение реализует валидацию и корректировку входных данных, предоставляет методы для ознакомления с доступными специальностями и образовательными организациями, метод подачи заявлений на зачисление, метод поиска заявлений, а также вспомогательные методы, необходимые для интеграции ПО в другую системы: методы получения справочных данных, метод поиска адреса.



✓	<default package>	1 sec 878 ms
✓	Prof2InstServiceTest	974 ms
✓	findProfessions()	949 ms
✓	findInstitutions()	25 ms
✓	InstitutionRepoTest	28 ms
✓	findByInstName()	28 ms
✓	ProfessionServiceTest	29 ms
✓	findAll()	29 ms
✓	ManualServiceTest	23 ms
✓	getGenderList()	8 ms
✓	getFinancingList()	7 ms
✓	getStudyFormList()	8 ms
✓	ApplicationRepoTest	136 ms
✓	applicationRepo_findAllByStudentId_Snails()	136 ms
✓	InstitutionServiceTest	25 ms
✓	getInstitutionList()	25 ms
✓	ApplicationServiceTest	649 ms
✓	createApplication_checkRequestParams_Fail_1()	78 ms
✓	createApplication_checkRequestParams_Fail_2()	44 ms
✓	createApplication_checkRequestParams_Fail_3()	56 ms
✓	createApplication_checkRequestParams_Fail_4()	50 ms
✓	createApplication_checkRequestParams_Fail_5()	64 ms
✓	createApplication_checkRequestParams_Fail_6()	63 ms
✓	createApplication_checkDocument_Fail_1()	51 ms
✓	createApplication_checkDocument_Fail_2()	49 ms
✓	findApplications_Success()	92 ms
✓	findApplications_Fail()	42 ms
✓	createApplication_checkRequestParams_Success()	60 ms
✓	ProfessionRepoTest	14 ms
✓	findByName()	14 ms

Рисунок 59 – Отчет об успешном прохождении модульных тестов

ЗАКЛЮЧЕНИЕ

В ходе выполнения магистерской диссертации была проанализирована предметная область, определен стек используемых технологий, рассмотрены способы осуществления быстрого поиска с учетом корректировки пользовательского запроса, а также определены необходимые функциональные составляющие для приложения виртуального ассистента для дистанционной подачи заявлений на зачисление в образовательные организации, разработана подробная API-документация для интеграции ПО в иные системы, выполнена валидация входящих данных для исключения ошибок, связанных с человеческим фактором. Оформлена сопутствующая документация.

Сервис включает в себя модульные тесты, также было проведено функциональное тестирование. Проведенные испытания и оценка функционала подтвердили, что препроцессор валидации и корректировки, его основные функции работают корректно и удовлетворяют все поставленные задачи.

За время проектирования и реализации данного проекта была освоена работа с NoSQL базами данных, в частности, с Elasticsearch, освоен механизм нечеткого поиска в Elasticsearch, улучшены навыки разработки приложений на языке программирования Java с использованием Spring Framework. Усовершенствованы умения проектирования и администрирования БД в системе управления базами данных PostgreSQL. Улучшены умения написания модульных тестов для Java приложений с помощью фреймворка Junit и библиотеки Mockito.

Результаты магистерской диссертации были апробированы в рамках научной статьи «Virtual assistant for remote submission of applications for enrollment in educational organizations», представленной на конференции «Information Technologies and Intelligent Decision Making System» (ITIDMS-II-

2021), статья опубликована в журнале «CEUR workshop proceedings», проиндексирован в Scopus.

Оформлено свидетельство о государственной регистрации программы для ЭВМ «Виртуальный ассистент для дистанционной подачи заявлений на зачисление в образовательные организации», номер свидетельства: RU 2023615616.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Крачтен Ф. Введение в Rational Unified Process. 2-е изд. М.: Вильямс, 2002. - 240 с.
2. Лобанов А. А. Большие данные: проблемы обработки //Вестник МГТУ МИРЭА «MSTU MIREA HERALD». – 2014. – №. 3. – с. 50.
3. Бельченко И. В., Дьяченко Р. А., Бельченко В. Е. К вопросу повышения производительности базы данных за счет оптимальной индексации таблиц //Электронный сетевой политематический журнал "Научные труды КубГТУ". – 2018. – №. 3. – С. 584-586.
4. Elasticsearch B. V. Elasticsearch [software], version. – 2018. – Т. 6. – №. 1.
5. Subramanian S., Inozemtseva L., Holmes R. Live API documentation //Proceedings of the 36th international conference on software engineering. – 2014. – С. 643-652.
6. Cheh C., Chen B. Analyzing openapi specifications for security design issues //2021 IEEE Secure Development Conference (SecDev). – IEEE, 2021. – с. 15-22.
7. Walls C. Spring Boot in action. – Simon and Schuster, 2015.
8. Чернова Е. В. Миграция базы данных //Международный научно-исследовательский журнал. – 2015. – №. 5-2 (36). – с. 113-114.
9. КОНДУРОВ И. В. Модульное тестирование программного обеспечения на Java с применением библиотек JUnit и Mockito //Программно-техническое обеспечение автоматизированных систем. – 2021. – с. 11-17.
10. Pham K., Reyes F., Rynhart J. Building a Library Search Infrastructure with Elasticsearch //Code4Lib Journal. – 2020. – №. 48.

ПРИЛОЖЕНИЕ А Листинг программы

Файл ChatbotApplication.java

```
@SpringBootApplication
public class ChatbotApplication {
    public static void main(String[] args) {
        SpringApplication.run(ChatbotApplication.class, args);
    }
}
```

Файл MessageUtils.java

```
@Component
public class MessageUtils {
    @Qualifier(value = "messageSource")
    private final MessageSource messageSource;

    public MessageUtils(MessageSource messageSource) {
        this.messageSource = messageSource;
    }

    public String getMessage(String selector, Object...
parameters) {
        return messageSource.getMessage(selector, parameters,
LocaleContextHolder.getLocale());
    }

    public String getMessage(String selector) {
        return messageSource.getMessage(selector, new
Object[] {}, LocaleContextHolder.getLocale());
    }
}
```

Файл ProfessionService.java

```
@Service
public class ProfessionService {
    private final ProfessionRepo repo;
    private final ProfessionToViewConverter
professionToViewConverter;

    public ProfessionService(ProfessionRepo repo,
ProfessionToViewConverter professionToViewConverter) {
        this.repo = repo;
        this.professionToViewConverter =
professionToViewConverter;
    }

    public ProfessionView findByName(String name) {
        Profession profession =
repo.findByName(name).orElseThrow(() -> new
NotFoundException(name));
        return professionToViewConverter.convert(profession);
    }
}
```

```

    }

    public Page<ProfessionView> findAll(Pageable pageable) {
        Page<Profession> professions = repo.findAll(pageable);
        List<ProfessionView> list = new ArrayList<>();
        professions.forEach(profession ->
list.add(professionToViewConverter.convert(profession)));
        return new PageImpl<>(list, pageable,
professions.getTotalElements());
    }
}

```

Файл ProfessionRepo.java

```

@Repository
public interface ProfessionRepo extends
JpaRepository<Profession, Long> {
    Optional<Profession> findByName(String name);
}

```

Файл Prof2InstService.java

```

@Service
public class Prof2InstService {
    private final Prof2InstRepo repo;
    private final InstitutionToViewConverter
institutionToViewConverter;
    private final ProfessionToViewConverter
professionToViewConverter;

    public Prof2InstService(Prof2InstRepo repo,
InstitutionToViewConverter institutionToViewConverter,
ProfessionToViewConverter professionToViewConverter) {
        this.repo = repo;
        this.institutionToViewConverter =
institutionToViewConverter;
        this.professionToViewConverter =
professionToViewConverter;
    }

    public Page<InstitutionView> findInstitutions(Long
professionId, Pageable pageable) {
        Page<Prof2Inst> instPage =
repo.findByProf_Id(professionId, pageable);
        List<InstitutionView> list = new ArrayList<>();
        instPage.forEach(prof2Inst ->
list.add(institutionToViewConverter.convert(prof2Inst.getInstitu
tion())));
        return new PageImpl<>(list, pageable,
instPage.getTotalElements());
    }

    public Page<ProfessionView> findProfessions(Long
institutionId, Pageable pageable) {

```

```

        Page<Prof2Inst> prof2InstList =
repo.findByInstitution_Id(institutionId, pageable);
        List<ProfessionView> professionViews = new
ArrayList<>();
        prof2InstList.forEach(prof2Inst ->
professionViews.add(professionToViewConverter.convert(prof2Inst.
getProf())));
        return new PageImpl<>(professionViews, pageable,
prof2InstList.getTotalElements());
    }
}

```

Файл Prof2InstRepo.java

```

@Repository
public interface Prof2InstRepo extends JpaRepository<Prof2Inst,
Long> {
    Page<Prof2Inst> findByProf_Id(Long professionId, Pageable
pageable);

    List<Prof2Inst> findByProf_Id(Long professionId);

    Page<Prof2Inst> findByInstitution_Id(Long institutionId,
Pageable pageable);
}

```

Файл ProfessionController.java

```

@RestController
@RequestMapping("/profession")
@Tag(name = "Специальности")
public class ProfessionController {
    private final ProfessionService professionService;
    private final Prof2InstService prof2InstService;

    public ProfessionController(ProfessionService
professionService, Prof2InstService prof2InstService) {
        this.professionService = professionService;
        this.prof2InstService = prof2InstService;
    }

    @Operation(summary = "Поиск образовательных организаций,
реализующих конкретную специальность")
    @GetMapping("/institution/list")
    public Page<InstitutionView>
findInstitutionsByProfessionId(@RequestParam Long professionId,

    @RequestParam(name = "size", required = false, defaultValue =
"10") Integer size,

    @Parameter(description = "Номер страницы", example = "0")

    @RequestParam(name = "page", required = false, defaultValue =
"0") Integer page,

```

```

@SortDefault(sort = "id", direction = Sort.Direction.ASC) Sort
sort) {
    Pageable pageable = PageRequest.of(page, size, sort);
    return prof2InstService.findInstitutions(professionId,
pageable);
}

@Operation(summary = "Получение списка всех доступных
специальностей")
@GetMapping("list")
public Page<ProfessionView> findAll(@RequestParam(name =
"size", required = false, defaultValue = "10") Integer size,
@Parameter(description =
"Номер страницы", example = "0")
@SortDefault(sort =
"page", required = false, defaultValue = "0") Integer page,
@SortDefault(sort =
"id", direction = Sort.Direction.ASC) Sort sort) {
    Pageable pageable = PageRequest.of(page, size, sort);
    return professionService.findAll(pageable);
}

@Operation(summary = "Поиск специальности по названию")
@GetMapping("/search")
public ProfessionView findProfessionByName(@RequestParam
String name) {
    return professionService.findByName(name);
}
}

```

Файл ProfessionView.java

```

public class ProfessionView implements Serializable {
    @Schema(description = "Идентификатор")
    private Long id;
    @Schema(description = "Наименование")
    private String name;

    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}

```

Файл Prof2Inst.java

```
@Entity
@Table(name = "prof2inst")
public class Prof2Inst {
    @Id
    @SequenceGenerator(name = "prof2inst_seq", sequenceName =
"prof2inst_seq", allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = "prof2inst_seq")
    @Column(name = "id")
    private Long id;
    @ManyToOne
    @JoinColumn(name = "institution_id")
    private Institution institution;
    @ManyToOne
    @JoinColumn(name = "prof_id")
    private Profession prof;
    public Prof2Inst() {
    }
    public Prof2Inst(Institution institution, Profession prof) {
        this.institution = institution;
        this.prof = prof;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public Institution getInstitution() {
        return institution;
    }
    public void setInstitution(Institution institutionId) {
        this.institution = institutionId;
    }
    public Profession getProf() {
        return prof;
    }
    public void setProf(Profession profId) {
        this.prof = profId;
    }
}
```

Файл Profession.java

```
@Entity
@Table(name = "profession")
public class Profession {
    @Id
    @SequenceGenerator(name = "profession_seq", sequenceName =
"profession_seq", allocationSize = 1)
```

```

    @GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = "profession_seq")
    @Column(name = "id")
    private Long id;

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

    @OneToMany(mappedBy = "prof", fetch = FetchType.EAGER)
    private Set<Prof2Inst> prof2Insts;

    @OneToMany(mappedBy = "profId", fetch = FetchType.EAGER)
    private Set<Application> applications;

    public Profession(String name) {
        this.name = name;
    }
    public Profession() {
    }
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public Set<Prof2Inst> getProf2Insts() {
        return prof2Insts;
    }
    public void setProf2Insts(Set<Prof2Inst> prof2Insts) {
        this.prof2Insts = prof2Insts;
    }
    public Set<Application> getApplications() {
        return applications;
    }
    public void setApplications(Set<Application> applications) {
        this.applications = applications;
    }
}

```

Файл ProfessionToViewConverter.java

```

@Component
public class ProfessionToViewConverter implements
Converter<Profession, ProfessionView> {
    @Override
    public ProfessionView convert(Profession profession) {

```

```

        ProfessionView view = new ProfessionView();
        view.setId(profession.getId());
        view.setName(profession.getName());
        return view;
    }
}

```

Файл ManualController.java

```

@RestController
@RequestMapping("/manual")
@Tag(name = "Справочные данные")
public class ManualController {
    private final ManualService manualService;

    public ManualController(ManualService manualService) {
        this.manualService = manualService;
    }

    @Operation(summary = "Получение справочника Пол")
    @GetMapping("/gender")
    public List<ManualResp> getGenderList() {
        return manualService.getGenderList();
    }

    @Operation(summary = "Получение справочника Финансирование")
    @GetMapping("/financing")
    public List<ManualResp> getFinancingList() {
        return manualService.getFinancingList();
    }

    @Operation(summary = "Получение справочника Форма обучения")
    @GetMapping("/studyForm")
    public List<ManualResp> getStudyFormList() {
        return manualService.getStudyFormList();
    }
}

```

Файл ManualResp.java

```

public class ManualResp implements Serializable {
    @Schema(title = "Код")
    private int code;
    @Schema(title = "Значение")
    private String value;

    public ManualResp(int code, String value) {
        this.code = code;
        this.value = value;
    }

    public int getCode() {
        return code;
    }

    public void setCode(int code) {
        this.code = code;
    }

    public String getValue() {
        return value;
    }
}

```

```

        public void setValue(String value) {
            this.value = value;
        }
    }
}

```

Файл ManualService.java

```

@Service
public class ManualService {
    public List<ManualResp> getGenderList() {
        List<ManualResp> resp = new ArrayList<>();
        Gender[] genders = Gender.values();
        for (Gender gender : genders) {
            resp.add(new ManualResp(gender.ordinal(),
gender.getDescription()));
        }
        return resp;
    }

    public List<ManualResp> getFinancingList() {
        List<ManualResp> resp = new ArrayList<>();
        FinancingType[] financingTypes = FinancingType.values();
        for (FinancingType financingType : financingTypes) {
            resp.add(new ManualResp(financingType.ordinal(),
financingType.getDescription()));
        }
        return resp;
    }

    public List<ManualResp> getStudyFormList() {
        List<ManualResp> resp = new ArrayList<>();
        StudyForm[] studyForms = StudyForm.values();
        for (StudyForm studyForm : studyForms) {
            resp.add(new ManualResp(studyForm.ordinal(),
studyForm.getDescription()));
        }
        return resp;
    }
}

```

Файл InstitutionView.java

```

public class InstitutionView implements Serializable {
    @Schema(description = "Идентификатор")
    private Long id;
    @Schema(description = "Наименование")
    private String instName;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getInstName() {
        return instName;
    }
}

```

```

    }
    public void setInstName(String instName) {
        this.instName = instName;
    }
}

```

Файл InstitutionService.java

```

@Service
public class InstitutionService {
    private final InstitutionRepo institutionRepo;
    private final InstitutionToViewConverter
institutionToViewConverter;
    public InstitutionService(InstitutionRepo institutionRepo,
InstitutionToViewConverter institutionToViewConverter) {
        this.institutionRepo = institutionRepo;
        this.institutionToViewConverter =
institutionToViewConverter;
    }
    public Page<InstitutionView> getInstitutionList(Pageable
pageable) {
        Page<Institution> institutionPage =
institutionRepo.findAll(pageable);
        List<InstitutionView> institutionViews = new
ArrayList<>();
        institutionPage.forEach(institution ->
institutionViews.add(institutionToViewConverter.convert(institut
ion)));
        return new PageImpl<>(institutionViews, pageable,
institutionPage.getTotalElements());
    }
    public InstitutionView findByName(String name) {
        Institution institution =
institutionRepo.findByInstName(name).orElseThrow(() -> new
NotFoundException(name));
        return institutionToViewConverter.convert(institution);
    }
}

```

Файл InstitutionRepo.java

```

@Repository
public interface InstitutionRepo extends
JpaRepository<Institution, Long> {
    Optional<Institution> findByInstName(String name);
}

```

Файл Institution.java

```

@Entity
@Table(name = "institution")
public class Institution {

    @Id

```

```

        @SequenceGenerator(name = "institution_seq", sequenceName =
"institution_seq", allocationSize = 1)
        @GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = "institution_seq")
        @Column(name = "id")
        private Long id;
        @Column(name = "inst_name")
        private String instName;
        @OneToMany(mappedBy = "institution", fetch =
FetchType.EAGER)
        private Set<Prof2Inst> prof2Insts;
        @OneToMany(mappedBy = "institutionId", fetch =
FetchType.EAGER)
        private Set<Application> applications;
        public Institution(String instName) {
            this.instName = instName;
        }
        public Institution() {
        }
        public Long getId() {
            return id;
        }
        public void setId(Long id) {
            this.id = id;
        }
        public String getInstName() {
            return instName;
        }
        public void setInstName(String instName) {
            this.instName = instName;
        }
        public Set<Prof2Inst> getProf2Insts() {
            return prof2Insts;
        }
        public void setProf2Insts(Set<Prof2Inst> prof2Insts) {
            this.prof2Insts = prof2Insts;
        }

        public Set<Application> getApplications() {
            return applications;
        }
        public void setApplications(Set<Application> applications) {
            this.applications = applications;
        }
    }
}

```

Файл InstitutionController.java

```

@RestController
@RequestMapping("/institution")
@Tag(name = "Образовательные организации")
public class InstitutionController {
    private final InstitutionService institutionService;
}

```

```

        private final Prof2InstService prof2InstService;
        public InstitutionController(InstitutionService
institutionService, Prof2InstService prof2InstService) {
            this.institutionService = institutionService;
            this.prof2InstService = prof2InstService;
        }
        @Operation(summary = "Получение списка доступных
образовательных организаций")
        @GetMapping("/list")
        public Page<InstitutionView>
getInstitutionList(@RequestParam(name = "size", required =
false, defaultValue = "10") Integer size,

@Parameter(description = "Номер страницы", example = "0")

@RequestParam(name = "page", required = false, defaultValue =
"0") Integer page,

@SortDefault(sort = "id", direction = Sort.Direction.ASC) Sort
sort) {
            Pageable pageable = PageRequest.of(page, size, sort);
            return institutionService.getInstitutionList(pageable);
        }
        @Operation(summary = "Поиск образовательной организации по
названию")
        @GetMapping("/search")
        public InstitutionView findByName(@RequestParam String name)
{
            return institutionService.findByName(name);
        }

        @Operation(summary = "Поиск специальностей в образовательной
организации")
        @GetMapping("/profession/list")
        public Page<ProfessionView>
findProfessionsByInstitutionId(@RequestParam Long institutionId,

@RequestParam(name = "size", required = false, defaultValue =
"10") Integer size,

@Parameter(description = "Номер страницы", example = "0")

@RequestParam(name = "page", required = false, defaultValue =
"0") Integer page,

@SortDefault(sort = "id", direction = Sort.Direction.ASC) Sort
sort) {
            Pageable pageable = PageRequest.of(page, size, sort);
            return prof2InstService.findProfessions(institutionId,
pageable);
        }
    }

```

Файл InstitutionToViewConverter.java

```
@Component
public class InstitutionToViewConverter implements
Converter<Institution, InstitutionView> {
    @Override
    public InstitutionView convert(Institution institution) {
        InstitutionView view = new InstitutionView();
        view.setId(institution.getId());
        view.setInstName(institution.getInstName());
        return view;
    }
}
```

Файл ApiExceptionHandlerController.java

```
@ControllerAdvice
public class ApiExceptionHandlerController extends
ApiExceptionHandler {
}
```

Файл ApiError.java

```
public interface ApiError extends Serializable {
}
```

Файл ApiErrorFactory.java

```
public class ApiErrorFactory {
    public static ApiError buildFromObjectError(ObjectError e) {
        if(e instanceof FieldError) {
            return ApiFieldError.valueOf((FieldError) e);
        } else {
            return ApiGlobalError.valueOf(e);
        }
    }
}
```

Файл ApiErrorImpl

```
public class ApiErrorImpl implements ApiError {
    private String message;
    private String code;
    private String objectName;
    private ApiErrorType type;
    protected ApiErrorImpl(ObjectError e) {
        this.message = e.getDefaultMessage();
        this.code = e.getCode();
        this.objectName = e.getObjectName();
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
```

```

        this.message = message;
    }

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public String getObjectname() {
        return objectName;
    }

    public void setObjectName(String objectName) {
        this.objectName = objectName;
    }

    public ApiErrorType getType() {
        return type;
    }

    protected void setType(ApiErrorType type) {
        this.type = type;
    }
}

```

Файл ApiResponse.java

```

public class ApiResponse implements Serializable {
    private Integer status;
    private String path;
    private String message;
    private String exception;
    private LocalDateTime timestamp;
    @JsonInclude(JsonInclude.Include.NON_EMPTY)
    private List<ApiError> errors;

    private ApiResponse(Integer status, String path, String
message, String exception) {
        this.status = status;
        this.path = path;
        this.message = message;
        this.exception = exception;
        this.timestamp = LocalDateTime.now();
    }

    public static ApiResponse valueOf(Integer status,
String path, String message, String exception) {
        return new ApiResponse(status, path, message,
exception);
    }
}

```

```

public ApiErrorResponse() {
}

public Integer getStatus() {
    return status;
}

public void setStatus(Integer status) {
    this.status = status;
}

public String getPath() {
    return path;
}

public void setPath(String path) {
    this.path = path;
}

public String getMessage() {
    return message;
}

public void setMessage(String message) {
    this.message = message;
}

public String getException() {
    return exception;
}

public void setException(String exception) {
    this.exception = exception;
}

public LocalDateTime getTimestamp() {
    return timestamp;
}

public void setTimestamp(LocalDateTime timestamp) {
    this.timestamp = timestamp;
}

public List<ApiError> getErrors() {
    return errors;
}

public void setErrors(List<ApiError> errors) {
    this.errors = errors;
}
}

```

Файл ApiErrorType.java

```
public enum ApiErrorType {  
    GLOBAL_ERROR, FIELD_ERROR  
}
```

Файл ApiFieldError.java

```
public class ApiFieldError extends ApiErrorImpl {  
    private String field;  
    private Object value;  
    private ApiFieldError(FieldError e) {  
        super(e);  
        super.setType(ApiErrorType.FIELD_ERROR);  
        this.field = e.getField();  
        Object obj = e.getRejectedValue();  
        if (obj instanceof MultipartFile) {  
            obj = ((MultipartFile) obj).getOriginalFilename();  
        }  
        this.value = obj;  
    }  
  
    public static ApiFieldError valueOf(FieldError e) {  
        return new ApiFieldError(e);  
    }  
  
    public String getField() {  
        return field;  
    }  
  
    public void setField(String field) {  
        this.field = field;  
    }  
  
    public Object getValue() {  
        return value;  
    }  
  
    public void setValue(Object value) {  
        this.value = value;  
    }  
}
```

Файл ApiGlobalError java

```
public class ApiGlobalError extends ApiErrorImpl {  
    private ApiGlobalError(ObjectError e) {  
        super(e);  
        super.setType(ApiErrorType.GLOBAL_ERROR);  
    }  
    public static ApiGlobalError valueOf(ObjectError e) {  
        return new ApiGlobalError(e);  
    }  
}
```

Файл ApiExceptionHandler.java

```
public class ApiExceptionHandler extends
ResponseEntityExceptionHandler {

    @ExceptionHandler({ApplicationException.class})
    public ResponseEntity<Object>
handleApplicationException(Exception ex, WebRequest request) {
        return prepareRestException(ex, request,
HttpStatus.BAD_REQUEST);
    }
    @ExceptionHandler({NotFoundException.class})
    public ResponseEntity<Object>
handleNotFoundException(Exception ex, WebRequest request) {
        return prepareRestException(ex, request,
HttpStatus.NOT_FOUND);
    }
    private ResponseEntity<Object>
prepareRestException(Exception ex, WebRequest request,
HttpStatus status) {
        logger.info(String.format("%s: %s",
ex.getClass().getName(), ex.getMessage()), ex);
        String error = ex.getLocalizedMessage();
        if (ex.getCause() != null) {
            error += String.format(". %s",
ex.getCause().getMessage());
        } else if (StringUtils.isEmpty(error)) {
            error = "message not available";
        }
        final ApiErrorResponse response =
ApiErrorResponse.valueOf(
            status.value(), getPath(request), error,
ex.getClass().getName());
        return new ResponseEntity<>(response, new HttpHeaders(),
status);
    }
    private String getPath(WebRequest request) {
        return ((ServletWebRequest) request)
            .getRequest()
            .getRequestURI();
    }
    @NonNull
    @Override
    protected ResponseEntity<Object>
handleMethodArgumentNotValid(@NonNull
MethodArgumentNotValidException ex, @NonNull HttpHeaders
headers,

@NonNull HttpStatus status, @NonNull WebRequest request) {
        logger.info(String.format("%s: %s",
ex.getClass().getName(), ex.getMessage()), ex);
        BindingResult bindingResult = ex.getBindingResult();
```

```

        Object target = bindingResult.getTarget();
        String message = String.format("Validation failed for
object='%s'. Error count: %s",
            target != null ?
target.getClass().getSimpleName() : "undefined",
            bindingResult.getErrorCount());
        final ApiErrorResponse response =
ApiErrorResponse.valueOf(
            HttpStatus.BAD_REQUEST.value(),
getPath(request), message, ex.getClass().getName());
        bindingResult.getAllErrors().forEach(e ->

response.getErrors().add(ApiErrorFactory.buildFromObjectError(e)
));
        return handleExceptionInternal(ex, response, headers,
HttpStatus.BAD_REQUEST, request);
    }

    @NonNull
    @Override
    protected ResponseEntity<Object>
handleTypeMismatch(TypeMismatchException ex, @NonNull
HttpHeaders headers, @NonNull HttpStatus status,
                                                                    @NonNull
WebRequest request) {
        logger.warn(ex.getClass().getName());
        return prepareRestException(ex, request,
HttpStatus.BAD_REQUEST);
    }

    @NonNull
    @Override
    protected ResponseEntity<Object>
handleServletRequestBindingException(ServletRequestBindingExcept
ion ex, @NonNull HttpHeaders headers,

@NonNull HttpStatus status, @NonNull WebRequest request) {
        logger.warn(ex.getClass().getName());
        return prepareRestException(ex, request,
HttpStatus.BAD_REQUEST);
    }

    @NonNull
    @Override
    protected ResponseEntity<Object>
handleHttpMessageNotReadable(HttpMessageNotReadableException ex,
@NonNull HttpHeaders headers,

@NonNull HttpStatus status, @NonNull WebRequest request) {
        logger.warn(ex.getClass().getName());
        return prepareRestException(ex, request,
HttpStatus.BAD_REQUEST);
    }

```

```

    }

    @ExceptionHandler({AuthorizationException.class})
    public ResponseEntity<Object>
    handleAuthorizationException(Exception ex, WebRequest request) {
        return prepareRestException(ex, request,
        HttpStatus.UNAUTHORIZED);
    }

    @NonNull
    @Override
    protected ResponseEntity<Object>
    handleNoHandlerFoundException(NoHandlerFoundException ex,
    @NonNull HttpHeaders headers,

    @NonNull HttpStatus status, @NonNull WebRequest request) {
        logger.error(ex.getClass().getName());
        return prepareRestException(ex, request,
        HttpStatus.BAD_REQUEST);
    }
}

```

Файл ApplicationException.java

```

public class ApplicationException extends RuntimeException {
    public ApplicationException() {
        super();
    }

    public ApplicationException(String message) {
        super(message);
    }

    public ApplicationException(String message, Throwable cause)
    {
        super(message, cause);
    }
}

```

Файл AuthorizationException.java

```

public class AuthorizationException extends RuntimeException {
    public AuthorizationException() {
        super();
    }

    public AuthorizationException(String message) {
        super(message);
    }

    public AuthorizationException(String message, Throwable
    cause) {
        super(message, cause);
    }
}

```

```
}
```

Файл NotFoundException.java

```
public class NotFoundException extends ApplicationException {
    public static final String ID_NOT_EXIST = "Таблица не
    содержит записи с id = ";

    public NotFoundException() {
        super();
    }

    public NotFoundException(String message) {
        super(message);
    }

    public NotFoundException(Long id) {
        super(ID_NOT_EXIST + id);
    }

    public NotFoundException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

Файл ElasticSearchApiClient.java

```
@Service
public class ElasticSearchApiClient {
    private final String SEARCH = "/_search";

    private final ElasticConfiguration elasticConfiguration;

    public ElasticSearchApiClient(ElasticConfiguration
    elasticConfiguration) {
        this.elasticConfiguration = elasticConfiguration;
    }

    public SearchResp findAddress(String query) {
        SearchResp searchResp = null;
        SearchReq searchReq = new SearchReq(new ESQuery(new
        MultiMatch(query)));
        URI uri = URI.create(elasticConfiguration.getUri() +
        elasticConfiguration.getIndex() + SEARCH);
        RestTemplate restTemplate = new RestTemplate();
        restTemplate.getInterceptors()
            .add(new
        BasicAuthenticationInterceptor(elasticConfiguration.getUsername(
        ), elasticConfiguration.getPassword()));
        try {
            searchResp = restTemplate.postForObject(uri,
            searchReq, SearchResp.class);
        } catch (Exception ignore) {
        }
    }
}
```

```

        }
        return searchResp;
    }
}

```

Файл ElasticSearchApiClient.java

```

public class ESQuery implements Serializable {
    @JsonProperty("multi_match")
    private MultiMatch multiMatch;

    public ESQuery(MultiMatch multiMatch) {
        this.multiMatch = multiMatch;
    }

    public MultiMatch getMultiMatch() {
        return multiMatch;
    }

    public void setMultiMatch(MultiMatch multiMatch) {
        this.multiMatch = multiMatch;
    }
}

```

Файл Hits.java

```

public class Hits {
    private Total total;
    @JsonProperty("hits")
    private List<HitsResult> hitsResult;

    public Hits() {
    }

    public Total getTotal() {
        return total;
    }

    public void setTotal(Total total) {
        this.total = total;
    }

    public List<HitsResult> getHitsResult() {
        return hitsResult;
    }

    public void setHitsResult(List<HitsResult> hitsResult) {
        this.hitsResult = hitsResult;
    }

    private class Total {
        private int value;
        private String relation;
    }
}

```

```

    public Total() {
    }

    public int getValue() {
        return value;
    }

    public void setValue(int value) {
        this.value = value;
    }

    public String getRelation() {
        return relation;
    }

    public void setRelation(String relation) {
        this.relation = relation;
    }
}
};

```

Файл HitsResult.java

```

public class HitsResult {
    @JsonProperty("_index")
    private String index;
    @JsonProperty("_id")
    private int id;
    @JsonProperty("_source")
    private SourceData source;
    public HitsResult() {
    }

    public HitsResult(String index, int id, SourceData source) {
        this.index = index;
        this.id = id;
        this.source = source;
    }

    public String getIndex() {
        return index;
    }

    public void setIndex(String index) {
        this.index = index;
    }

    public int getId() {
        return id;
    }
}

```

```

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

public SourceData getSource() {
    return source;
}

public void setSource(SourceData source) {
    this.source = source;
}

public static class SourceData {
    private String region;
    private String district;
    private String locality;
    private String street;
    private String building;
    private String index;

    public SourceData() {
    }

    public String getRegion() {
        return region;
    }

    public void setRegion(String region) {
        this.region = region;
    }

    public String getDistrict() {
        return district;
    }

    public void setDistrict(String district) {
        this.district = district;
    }

    public String getLocality() {
        return locality;
    }

    public void setLocality(String locality) {
        this.locality = locality;
    }

    public String getStreet() {
        return street;
    }
}

```

```

        public void setStreet(String street) {
            this.street = street;
        }

        public String getBuilding() {
            return building;
        }

        public void setBuilding(String building) {
            this.building = building;
        }

        public String getIndex() {
            return index;
        }

        public void setIndex(String index) {
            this.index = index;
        }
    }
}

```

Файл MultiMatch.java

```

public class MultiMatch {
    private String query;
    private final List<String> fields = Arrays.asList("street",
"index", "building", "locality", "district", "region");
    private final String analyzer = "russian";
    private final String fuzziness = "AUTO";
    @JsonProperty("minimum_should_match")
    private final String minimumShouldMatch = "70%";
    @JsonProperty("tie_breaker")
    private final BigDecimal tieBreaker = new BigDecimal("0.2");

    public MultiMatch(String query) {
        this.query = query;
    }

    public String getQuery() {
        return query;
    }

    public void setQuery(String query) {
        this.query = query;
    }

    public List<String> getFields() {
        return fields;
    }
}

```

```

    public String getAnalyzer() {
        return analyzer;
    }

    public String getFuzziness() {
        return fuzziness;
    }

    public String getMinimumShouldMatch() {
        return minimumShouldMatch;
    }

    public BigDecimal getTieBreaker() {
        return tieBreaker;
    }
}

```

Файл SearchReq.java

```

public class SearchReq implements Serializable {
    private int size = 5;
    private ESQuery query;

    public SearchReq(ESQuery query) {
        this.query = query;
    }

    public ESQuery getQuery() {
        return query;
    }

    public void setQuery(ESQuery query) {
        this.query = query;
    }

    public int getSize() {
        return size;
    }

    public void setSize(int size) {
        this.size = size;
    }
}

```

Файл SearchResp.java

```

public class SearchResp implements Serializable {
    @JsonProperty("_shards")
    private Shards shards;
    private Hits hits;
}

```

```

public SearchResp() {
}

public Shards getShards() {
    return shards;
}

public void setShards(Shards shards) {
    this.shards = shards;
}

public Hits getHits() {
    return hits;
}

public void setHits(Hits hits) {
    this.hits = hits;
}
}

```

Файл Shards.java

```

public class Shards {
    private int total;
    private int successful;
    private int skipped;
    private int failed;

    public Shards() {
    }

    public int getTotal() {
        return total;
    }

    public void setTotal(int total) {
        this.total = total;
    }

    public int getSuccessful() {
        return successful;
    }

    public void setSuccessful(int successful) {
        this.successful = successful;
    }

    public int getSkipped() {
        return skipped;
    }

    public void setSkipped(int skipped) {
        this.skipped = skipped;
    }
}

```

```

    }

    public int getFailed() {
        return failed;
    }

    public void setFailed(int failed) {
        this.failed = failed;
    }
}

```

Файл ElasticConfiguration.java

```

@Configuration
@ConfigurationProperties(prefix = "elasticsearch")
@EnableConfigurationProperties
public class ElasticConfiguration {
    private String index;
    private String username;
    private String password;
    private String uri;

    public String getIndex() {
        return index;
    }

    public void setIndex(String index) {
        this.index = index;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getUri() {
        return uri;
    }

    public void setUri(String uri) {
        this.uri = uri;
    }
}

```

```
}
```

Файл MessageSourceConfig.java

```
@Configuration
public class MessageSourceConfig {
    @Bean
    public ReloadableResourceBundleMessageSource messageSource()
    {
        ReloadableResourceBundleMessageSource messageBundle =
new ReloadableResourceBundleMessageSource();
        messageBundle.setBasename("classpath:message");
        messageBundle.setDefaultEncoding("UTF-8");
        return messageBundle;
    }
}
```

Файл SwaggerConfig.java

```
@Configuration
public class SwaggerConfig {
    @Bean
    public OpenAPI openAPI() {
        return new OpenAPI()
            .info(new Info().title("Сервис подачи заявлений
на зачисление"));
    }
}
```

Файл StudentRepo.java

```
@Repository
public interface StudentRepo extends JpaRepository<Student,
Long> {
}
```

Файл DocumentRepo.java

```
@Repository
public interface DocumentRepo extends JpaRepository<Document,
Long> {
}
```

Файл ApplicationService.java

```
@Service
public class ApplicationService {
    private final ApplicationRepo applicationRepo;
    private final DocumentRepo documentRepo;
    private final StudentRepo studentRepo;
    private final InstitutionRepo institutionRepo;
    private final ProfessionRepo professionRepo;
    private final ApplicationToCreateApplicationRespConverter
createApplicationRespConverter;
    private final ApplicationToViewConverter
applicationToViewConverter;
    private final MessageUtils messageUtils;
}
```

```

        private final Prof2InstRepo prof2InstRepo;
        private final ElasticSearchApiClient esApiClient;
        private final SourceDataToAddressRespConverter
sourceDataToAddressRespConverter;
        private final AddressRepo addressRepo;

        private final String REGEX_DOC_NUMBER = "^([0-9]{6})?$";
        private final String REGEX_DOC_SERIA = "^([0-9]{4})?$";

        public ApplicationService(ApplicationRepo applicationRepo,
DocumentRepo documentRepo, StudentRepo studentRepo,
                                InstitutionRepo institutionRepo,
ProfessionRepo professionRepo,

ApplicationToCreateApplicationRespConverter
createApplicationRespConverter,
                                ApplicationToViewConverter
applicationToViewConverter, MessageUtils messageUtils,
                                Prof2InstRepo prof2InstRepo,
ElasticSearchApiClient esApiClient,
                                SourceDataToAddressRespConverter
sourceDataToAddressRespConverter, AddressRepo addressRepo) {
            this.applicationRepo = applicationRepo;
            this.documentRepo = documentRepo;
            this.studentRepo = studentRepo;
            this.institutionRepo = institutionRepo;
            this.professionRepo = professionRepo;
            this.createApplicationRespConverter =
createApplicationRespConverter;
            this.applicationToViewConverter =
applicationToViewConverter;
            this.messageUtils = messageUtils;
            this.prof2InstRepo = prof2InstRepo;
            this.esApiClient = esApiClient;
            this.sourceDataToAddressRespConverter =
sourceDataToAddressRespConverter;
            this.addressRepo = addressRepo;
        }

        public CreateApplicationResp
createApplication(CreateApplicationReq req) {
            Optional<Profession> profession =
professionRepo.findById(req.getProfessionId());
            Optional<Institution> institution =
institutionRepo.findById(req.getInstitutionId());
            StudentReq studentReq = req.getStudent();
            CreateAddress addressReq = studentReq.getAddress();
            DocumentReq documentReq = req.getDocument();
            Gender gender = Gender.from(studentReq.getGender());
            FinancingType financing =
FinancingType.from(req.getFinancing());

```

```

        StudyForm studyForm =
StudyForm.from(req.getStudyForm());

        checkRequestParams(profession, institution, gender,
financing, studyForm);
        checkDocument(documentReq);

        Document document = new
Document(documentReq.getSerial(), documentReq.getNumber(),
documentReq.getIssuer(),
        documentReq.getIssueDate(),
documentReq.getDepartmentCode());

        Address address = new Address(addressReq.getRegion(),
addressReq.getDistrict(), addressReq.getLocality(),
        addressReq.getStreet(),
addressReq.getBuilding(), addressReq.getIndex(),
addressReq.getFlat());

        Student student = new Student(studentReq.getFirstName(),
studentReq.getLastName(), studentReq.getMiddleName(),
        studentReq.getBday(), gender, document,
studentReq.getSnils(), address);
        student = studentRepo.saveAndFlush(student);

        Application application = new Application(student,
profession.get(), institution.get(),
        studyForm, financing);
        application = applicationRepo.saveAndFlush(application);

        return
createApplicationRespConverter.convert(application);
    }

    private void checkDocument(DocumentReq documentReq) {
        boolean number =
documentReq.getNumber().matches(REGEX_DOC_NUMBER);
        if (!number)
            throw new
ApplicationException(messageUtils.getMessage("application.DocNum
ber"));
        boolean serial =
documentReq.getSerial().matches(REGEX_DOC_SERIA);
        if (!serial)
            throw new
ApplicationException(messageUtils.getMessage("application.DocSer
ial"));
    }

    private void checkRequestParams(Optional<Profession>
profession, Optional<Institution> institution,

```

```

                                Gender gender, FinancingType
financing, StudyForm studyForm) {
    if (profession.isEmpty()) {
        throw new
NotFoundException(messageUtils.getMessage("profession.notFound")
);
    }
    if (institution.isEmpty()) {
        throw new
NotFoundException(messageUtils.getMessage("institution.notFound"
));
    }
    List<Prof2Inst> instList =
prof2InstRepo.findByProf_Id(profession.get().getId());
    long count = instList.stream()
        .filter(inst ->
inst.getInstitution().getId().equals(institution.get().getId()))
        .count();
    if (count == 0) {
        throw new
NotFoundException(messageUtils.getMessage("profession.noSuchInIn
st"));
    }

    if (gender == null) {
        throw new
NotFoundException(messageUtils.getMessage("gender.NotFound"));
    }
    if (financing == null) {
        throw new
NotFoundException(messageUtils.getMessage("financing.NotFound"));
    }
    if (studyForm == null) {
        throw new
NotFoundException(messageUtils.getMessage("studyForm.NotFound"));
    }
}

    public FindApplicationsResp findApplications(String snils) {
        List<Application> applications =
applicationRepo.findAllByStudentId_Snils(snils);
        List<ApplicationView> applicationViews = new
ArrayList<>();
        if (!applications.isEmpty()) {
            applications.forEach(application ->
applicationViews.add(applicationToViewConverter.convert(applicat
ion)));
            return new FindApplicationsResp(applicationViews,
snils);
        }
    }
}

```

```

        throw new
NotFoundException(messageUtils.getMessage("application.NotFound"
));
    }

    public List<AddressResp> findAddress(String address) {
        List<AddressResp> addressResps = new ArrayList<>();
        SearchResp searchResp =
esApiClient.findAddress(address);
        List<HitsResult> hitsResults =
searchResp.getHits().getHitsResult();
        hitsResults.forEach(hitsResult -> addressResps

.add(sourceDataToAddressRespConverter.convert(hitsResult.getSource())));
        if (addressResps.size() == 0) {
            throw new
NotFoundException(messageUtils.getMessage("address.NotFound"));
        }
        return addressResps;
    }
}

```

Файл ApplicationRepo.java

```

@Repository
public interface ApplicationRepo extends
JpaRepository<Application, Long> {
    List<Application> findAllByStudentId_Snils(String snils);
}

```

Файл AddressRepo.java

```

@Repository
public interface AddressRepo extends JpaRepository<Address,
Long> {
}

```

Файл AddressResp.java

```

public class AddressResp {
    private String region;
    private String district;
    private String locality;
    private String street;
    private String building;
    private String index;

    public AddressResp() {
    }

    public String getRegion() {
        return region;
    }
}

```

```

    public void setRegion(String region) {
        this.region = region;
    }

    public String getDistrict() {
        return district;
    }

    public void setDistrict(String district) {
        this.district = district;
    }

    public String getLocality() {
        return locality;
    }

    public void setLocality(String locality) {
        this.locality = locality;
    }

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getBuilding() {
        return building;
    }

    public void setBuilding(String building) {
        this.building = building;
    }

    public String getIndex() {
        return index;
    }

    public void setIndex(String index) {
        this.index = index;
    }
}

```

Файл ApplicationController.java

```

@RestController
@Tag(name = "Заявления")
@RequestMapping("/application")
public class ApplicationController {
    private final ApplicationService service;
}

```

```

    public ApplicationController(ApplicationService service) {
        this.service = service;
    }

    @Operation(summary = "Подать заявление")
    @PostMapping
    public CreateApplicationResp createApplication(@RequestBody
    @Valid CreateApplicationReq req) {
        return service.createApplication(req);
    }

    @Operation(summary = "Поиск заявлений студента по СНИЛС")
    @GetMapping("/student/{snils}")
    public FindApplicationsResp findApplication(@PathVariable
    String snils) {
        return service.findApplications(snils);
    }

    @Operation(summary = "Поиск адреса")
    @GetMapping("/findAddress")
    public List<AddressResp> findAddress(@RequestParam String
    address) {
        return service.findAddress(address);
    }
}

```

Файл CreateAddress.java

```

public class CreateAddress {
    @NotNull
    private String region;
    @NotNull
    private String district;
    @NotNull
    private String locality;
    @NotNull
    private String street;
    @NotNull
    private String building;
    @NotNull
    private String index;
    private String flat;

    public CreateAddress() {
    }

    public CreateAddress(String region, String district, String
    locality, String street, String building, String index, String
    flat) {
        this.region = region;
        this.district = district;
        this.locality = locality;
        this.street = street;
    }
}

```

```

        this.building = building;
        this.index = index;
        this.flat = flat;
    }

    public String getRegion() {
        return region;
    }

    public void setRegion(String region) {
        this.region = region;
    }

    public String getDistrict() {
        return district;
    }

    public void setDistrict(String district) {
        this.district = district;
    }

    public String getLocality() {
        return locality;
    }

    public void setLocality(String locality) {
        this.locality = locality;
    }

    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getBuilding() {
        return building;
    }

    public void setBuilding(String building) {
        this.building = building;
    }

    public String getIndex() {
        return index;
    }

    public void setIndex(String index) {
        this.index = index;
    }
}

```

```

    public String getFlat() {
        return flat;
    }

    public void setFlat(String flat) {
        this.flat = flat;
    }
}

```

Файл CreateApplicationReq.java

```

public class CreateApplicationReq {
    @Schema(title = "Идентификатор ОО")
    @NotNull
    private Long institutionId;

    @Schema(title = "Идентификатор специальности")
    @NotNull
    private Long professionId;

    @Schema(title = "Код финансирования")
    @NotNull
    private Integer financing;

    @Schema(title = "Код формы обучения")
    @NotNull
    private Integer studyForm;

    @Schema(description = "Документ удостоверяющий личность")
    @Valid
    private DocumentReq document;

    @Schema(description = "Абитуриент")
    @Valid
    private StudentReq student;

    public CreateApplicationReq() {
    }

    public CreateApplicationReq(Long institutionId, Long
professionId, Integer financing, Integer studyForm,
                                DocumentReq document, StudentReq
student) {
        this.institutionId = institutionId;
        this.professionId = professionId;
        this.financing = financing;
        this.studyForm = studyForm;
        this.document = document;
        this.student = student;
    }

    public Long getInstitutionId() {

```

```

        return institutionId;
    }

    public void setInstitutionId(Long institutionId) {
        this.institutionId = institutionId;
    }

    public Long getProfessionId() {
        return professionId;
    }

    public void setProfessionId(Long professionId) {
        this.professionId = professionId;
    }

    public Integer getFinancing() {
        return financing;
    }

    public void setFinancing(Integer financing) {
        this.financing = financing;
    }

    public Integer getStudyForm() {
        return studyForm;
    }

    public void setStudyForm(Integer studyForm) {
        this.studyForm = studyForm;
    }

    public DocumentReq getDocument() {
        return document;
    }

    public void setDocument(DocumentReq document) {
        this.document = document;
    }

    public StudentReq getStudent() {
        return student;
    }

    public void setStudent(StudentReq student) {
        this.student = student;
    }
}

```

Файл CreateApplicationResp.java

```

public class CreateApplicationResp implements Serializable {
    @Schema(title = "Наименование ОО")
    private String institutionName;

```

```

@Schema(title = "Наименование специальности")
private String professionName;

@Schema(title = "Финансирование")
private String financing;

@Schema(title = "Форма обучения")
private String studyForm;

@Schema(title = "Документ удостоверяющий личность")
private DocumentResp document;

@Schema(title = "Абитуриент")
private StudentResp student;

public String getInstitutionName() {
    return institutionName;
}

public void setInstitutionName(String institutionName) {
    this.institutionName = institutionName;
}

public String getProfessionName() {
    return professionName;
}

public void setProfessionName(String professionName) {
    this.professionName = professionName;
}

public String getFinancing() {
    return financing;
}

public void setFinancing(String financing) {
    this.financing = financing;
}

public String getStudyForm() {
    return studyForm;
}

public void setStudyForm(String studyForm) {
    this.studyForm = studyForm;
}

public DocumentResp getDocument() {
    return document;
}

```

```

    public void setDocument(DocumentResp document) {
        this.document = document;
    }

    public StudentResp getStudent() {
        return student;
    }

    public void setStudent(StudentResp student) {
        this.student = student;
    }
}

```

Файл DocumentReq.java

```

public class DocumentReq {
    @Schema(title = "Серия")
    @NotNull
    private String serial;

    @Schema(title = "Номер")
    @NotNull
    private String number;

    @Schema(title = "Кем выдан")
    @NotNull
    private String issuer;

    @Schema(title = "Дата выдачи")
    @NotNull
    private LocalDate issueDate;

    @Schema(title = "Код подразделения")
    @NotNull
    private String departmentCode;

    public DocumentReq() {
    }

    public DocumentReq(String serial, String number, String
issuer, LocalDate issueDate, String departmentCode) {
        this.serial = serial;
        this.number = number;
        this.issuer = issuer;
        this.issueDate = issueDate;
        this.departmentCode = departmentCode;
    }

    public String getSerial() {
        return serial;
    }

    public void setSerial(String serial) {
        this.serial = serial;
    }
}

```

```

    }

    public String getNumber() {
        return number;
    }

    public void setNumber(String number) {
        this.number = number;
    }

    public String getIssuer() {
        return issuer;
    }

    public void setIssuer(String issuer) {
        this.issuer = issuer;
    }

    public LocalDate getIssueDate() {
        return issueDate;
    }

    public void setIssueDate(LocalDate issueDate) {
        this.issueDate = issueDate;
    }

    public String getDepartmentCode() {
        return departmentCode;
    }

    public void setDepartmentCode(String departmentCode) {
        this.departmentCode = departmentCode;
    }
}

```

Файл DocumentReq.java

```

public class DocumentResp extends DocumentReq {
}

```

Файл FindApplicationsResp.java

```

public class FindApplicationsResp implements Serializable {
    @Schema(description = "Список заявлений")
    private List<ApplicationView> applicationViews;

    @Schema(title = "Номер СНИЛС")
    private String snils;

    public FindApplicationsResp(List<ApplicationView>
applicationViews, String snils) {
        this.applicationViews = applicationViews;
        this.snils = snils;
    }
}

```

```

    public List<ApplicationView> getApplicationViews() {
        return applicationViews;
    }

    public void setApplicationViews(List<ApplicationView>
applicationViews) {
        this.applicationViews = applicationViews;
    }

    public String getSnils() {
        return snils;
    }

    public void setSnils(String snils) {
        this.snils = snils;
    }
}

```

Файл FindApplicationsResp.java

```

public class FindApplicationsResp implements Serializable {
    @Schema(description = "Список заявлений")
    private List<ApplicationView> applicationViews;
}

```

Файл StudentReq.java

```

public class StudentReq {
    @Schema(title = "Имя")
    @NotNull
    private String firstName;

    @Schema(title = "Отчество")
    private String middleName;

    @Schema(title = "Фамилия")
    @NotNull
    private String lastName;

    @Schema(title = "Код пола")
    @NotNull
    private Integer gender;

    @Schema(title = "Дата рождения")
    @NotNull
    private LocalDate bday;

    @Schema(title = "Номер СНИЛС")
    @NotNull
    private String snils;

    @Schema(title = "Адрес")
    @Valid
    private CreateAddress address;
}

```

```

public StudentReq() {
}

public StudentReq(String firstName, String middleName,
String lastName, Integer gender, LocalDate bday,
String snils, CreateAddress address) {
    this.firstName = firstName;
    this.middleName = middleName;
    this.lastName = lastName;
    this.gender = gender;
    this.bday = bday;
    this.snils = snils;
    this.address = address;
}

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getMiddleName() {
    return middleName;
}

public void setMiddleName(String middleName) {
    this.middleName = middleName;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public Integer getGender() {
    return gender;
}

public void setGender(Integer gender) {
    this.gender = gender;
}

public LocalDate getBday() {
    return bday;
}

```

```

    public void setBday(LocalDate bday) {
        this.bday = bday;
    }

    public String getSnils() {
        return snils;
    }

    public void setSnils(String snils) {
        this.snils = snils;
    }

    public CreateAddress getAddress() {
        return address;
    }

    public void setAddress(CreateAddress address) {
        this.address = address;
    }
}

```

Файл StudentResp.java

```

public class StudentResp {
    private String firstName;
    private String middleName;
    private String lastName;
    private String gender;
    private LocalDate bday;
    private String snils;
    private CreateAddress address;

}

```

Файл Address.java

```

@Entity
@Table(name = "address")
public class Address {
    @Id
    @SequenceGenerator(name = "address_seq", sequenceName =
"address_seq", allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = "address_seq")
    @Column(name = "id")
    private Long id;

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

    @Column(name = "district")
    private String district;
}

```

```

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

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

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

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

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

@OneToOne(mappedBy = "address")
private Student student;

public Address() {
}

public Address(String region, String district, String
locality, String street,
                String building, String index, String flat) {
    this.region = region;
    this.district = district;
    this.locality = locality;
    this.street = street;
    this.building = building;
    this.index = index;
    this.flat = flat;
}
}

```

Файл Application.java

```

@Entity
@Table(name = "application")
public class Application {
    @Id
    @SequenceGenerator(name = "application_seq", sequenceName =
"application_seq", allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = "application_seq")
    @Column(name = "id")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "student_id")
    private Student studentId;

    @ManyToOne
    @JoinColumn(name = "prof_id")

```

```

private Profession profId;

@ManyToOne
@JoinColumn(name = "institution_id")
private Institution institutionId;

@Column(name = "study_form")
@Enumerated(EnumType.ORDINAL)
private StudyForm studyForm;

@Column(name = "financing")
@Enumerated(EnumType.ORDINAL)
private FinancingType financing;

public Application(Student studentId, Profession profId,
Institution institutionId, StudyForm studyForm,
FinancingType financing) {
    this.studentId = studentId;
    this.profId = profId;
    this.institutionId = institutionId;
    this.studyForm = studyForm;
    this.financing = financing;
}

public Application() {
}
}

```

Файл ApplicationView.java

```

public class ApplicationView implements Serializable {
    @Schema(title = "Идентификатор заявления")
    private Long id;
    @Schema(title = "Наименование специальности")
    private String professionName;
    @Schema(title = "Наименование ОО")
    private String institutionName;
    @Schema(title = "Форма обучения")
    private String studyForm;
    @Schema(title = "Финансирование")
    private String financing;
    public Long getId() {
        return id;
    }
}

```

Файл Document.java

```

@Entity
@Table(name = "document")
public class Document {
    @Id
    @SequenceGenerator(name = "document_seq", sequenceName =
"document_seq", allocationSize = 1)

```

```

        @GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = "document_seq")
        @Column(name = "id", nullable = false)
        private Long id;
        @Column(name = "doc_seria")
        private String docSeria;
        @Column(name = "doc_number")
        private String docNumber;
        @Column(name = "issuer")
        private String issuer;
        @Column(name = "issue_date")
        private LocalDate issueDate;
        @Column(name = "dep_code")
        private String depCode;
        @OneToOne(mappedBy = "documentId")
        private Student student;

        public Document(String docSeria, String docNumber, String
issuer, LocalDate issueDate, String depCode) {
            this.docSeria = docSeria;
            this.docNumber = docNumber;
            this.issuer = issuer;
            this.issueDate = issueDate;
            this.depCode = depCode;
        }

        public Document() {
        }
    }
}

```

Файл FinancingType.java

```

public enum FinancingType {
    CONTRACT("На договорной основе"),
    BUDGET("Бюджет");
    private final String description;

    FinancingType(String description) {
        this.description = description;
    }

    public static FinancingType from(int ordinal) {
        for (FinancingType financingType :
FinancingType.values()) {
            if (financingType.ordinal() == ordinal)
                return financingType;
        }
        return null;
    }

    public String getDescription() {

```

```

        return description;
    }
}

```

Файл Gender.java

```

public enum Gender {
    MAN("Мужской"),
    WOMAN("Женский");
    private final String description;

    Gender(String description) {
        this.description = description;
    }

    public static Gender from(int genderOrdinal) {
        for (Gender gender : Gender.values()) {
            if (gender.ordinal() == genderOrdinal)
                return gender;
        }
        return null;
    }

    public String getDescription() {
        return description;
    }
}

```

Файл Student.java

```

@Entity
@Table(name = "student")
public class Student {
    @Id
    @SequenceGenerator(name = "student_seq", sequenceName =
"student_seq", allocationSize = 1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = "student_seq")
    @Column(name = "id", nullable = false)
    private Long id;
    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;
    @Column(name = "middle_name")
    private String middleName;
    @Column(name = "bday")
    private LocalDate bday;
    @Column(name = "gender")
    @Enumerated(EnumType.ORDINAL)
    private Gender gender;
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "document_id")

```

```

private Document documentId;
@Column(name = "snils")
private String snils;
@OneToMany(mappedBy = "studentId", fetch = FetchType.EAGER)
private Set<Application> applications;
@OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "address_id")
private Address address;

public Student(String firstName, String lastName, String
middleName, LocalDate bday, Gender gender,
                Document documentId, String snils, Address
address) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.middleName = middleName;
    this.bday = bday;
    this.gender = gender;
    this.documentId = documentId;
    this.snils = snils;
    this.address = address;
}

public Student() {
}

```

Файл StudyForm.java

```

public enum StudyForm {
    EXTRAMURAL("Заочно"),
    FULL_TIME("Очно");
    private final String description;

    StudyForm(String description) {
        this.description = description;
    }

    public static StudyForm from(int ordinal) {
        for (StudyForm studyForm : StudyForm.values()) {
            if (studyForm.ordinal() == ordinal)
                return studyForm;
        }
        return null;
    }

    public String getDescription() {
        return description;
    }
}

```

Файл ApplicationToCreateApplicationRespConverter.java

```

@Component

```

```

public class ApplicationToCreateApplicationRespConverter
implements Converter<Application, CreateApplicationResp> {
    @Override
    public CreateApplicationResp convert(Application
application) {
        StudentResp studentResp = new StudentResp();
        CreateAddress address = new CreateAddress();
        Address appAddress =
application.getStudentId().getAddress();
        address.setRegion(appAddress.getRegion());
        address.setDistrict(appAddress.getDistrict());
        address.setLocality(appAddress.getLocality());
        address.setStreet(appAddress.getStreet());
        address.setBuilding(appAddress.getBuilding());
        address.setIndex(appAddress.getIndex());
        address.setFlat(appAddress.getFlat());
studentResp.setFirstName(application.getStudentId().getFirstName
());

studentResp.setMiddleName(application.getStudentId().getMiddleNa
me());

studentResp.setLastName(application.getStudentId().getLastName()
);

studentResp.setBday(application.getStudentId().getBday());

studentResp.setGender(application.getStudentId().getGender().get
Description());

studentResp.setSnils(application.getStudentId().getSnils());
        studentResp.setAddress(address);

        DocumentResp documentResp = new DocumentResp();
        Document document =
application.getStudentId().getDocumentId();
        documentResp.setNumber(document.getDocNumber());
        documentResp.setSerial(document.getDocSeria());
        documentResp.setIssuer(document.getIssuer());
        documentResp.setIssueDate(document.getIssueDate());
        documentResp.setDepartmentCode(document.getDepCode());

        CreateApplicationResp resp = new
CreateApplicationResp();
        resp.setDocument(documentResp);
        resp.setStudent(studentResp);

resp.setFinancing(application.getFinancing().getDescription());

resp.setStudyForm(application.getStudyForm().getDescription());

```

```

resp.setInstitutionName(application.getInstitutionId().getInstName());

resp.setProfessionName(application.getProfId().getName());
return resp;
}
}

```

Файл ApplicationToViewConverter.java

```

@Component
public class ApplicationToViewConverter implements
Converter<Application, ApplicationView> {
    @Override
    public ApplicationView convert(Application application) {
        ApplicationView view = new ApplicationView();
        view.setId(application.getId());

view.setFinancing(application.getFinancing().getDescription());

view.setInstitutionName(application.getInstitutionId().getInstName());

view.setProfessionName(application.getProfId().getName());

view.setStudyForm(application.getStudyForm().getDescription());
        return view;
    }
}

```

Файл SourceDataToAddressRespConverter.java

```

@Component
public class SourceDataToAddressRespConverter implements
Converter<HitsResult.SourceData, AddressResp> {
    @Override
    public AddressResp convert(HitsResult.SourceData source) {
        AddressResp addressResp = new AddressResp();
        addressResp.setDistrict(source.getDistrict());
        addressResp.setRegion(source.getRegion());
        addressResp.setLocality(source.getLocality());
        addressResp.setStreet(source.getStreet());
        addressResp.setBuilding(source.getBuilding());
        addressResp.setIndex(source.getIndex());
        return addressResp;
    }
}

```

Файл init_DB.sql

```

CREATE SEQUENCE document_seq
INCREMENT 1
START 1
MINVALUE 1

```

```

MAXVALUE 9223372036854775807
CACHE 1;

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

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

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

CREATE SEQUENCE prof2inst_seq
INCREMENT 1
START 1
MINVALUE 1
MAXVALUE 9223372036854775807
CACHE 1;

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

CREATE TABLE DOCUMENT
(
    id          bigint          NOT NULL PRIMARY KEY
DEFAULT nextval('document_seq'::regclass),
    doc_seria   character varying(255) NOT NULL,
    doc_number  character varying(255) NOT NULL,
    issuer      character varying(255) NOT NULL,
    issue_date  date            NOT NULL,
    dep_code    character varying(255) NOT NULL
);

CREATE TABLE STUDENT
(

```

```

        id          bigint          NOT NULL PRIMARY KEY
DEFAULT nextval('student_seq'::regclass),
    first_name  character varying(255) NOT NULL,
    last_name   character varying(255) NOT NULL,
    middle_name character varying(255),
    bday        date                 NOT NULL,
    gender      integer              NOT NULL,
    document_id bigint               NOT NULL,
    FOREIGN KEY (document_id) references DOCUMENT (id) on delete
cascade
        on update cascade
);
CREATE TABLE INSTITUTION
(
    id          bigint          NOT NULL PRIMARY KEY
DEFAULT nextval('institution_seq'::regclass),
    inst_name  character varying(255) NOT NULL
);

CREATE TABLE PROFESSION
(
    id  bigint          NOT NULL PRIMARY KEY DEFAULT
nextval('profession_seq'::regclass),
    name character varying(255) NOT NULL
);

CREATE TABLE PROF2INST
(
    id          bigint NOT NULL PRIMARY KEY DEFAULT
nextval('prof2inst_seq'::regclass),
    institution_id bigint NOT NULL,
    prof_id      bigint NOT NULL,
    FOREIGN KEY (institution_id) references INSTITUTION (id) on
delete cascade
        on update cascade,
    FOREIGN KEY (prof_id) references PROFESSION (id) on delete
cascade
        on update cascade
);

CREATE TABLE APPLICATION
(
    id          bigint NOT NULL PRIMARY KEY DEFAULT
nextval('application_seq'::regclass),
    student_id  bigint NOT NULL,
    prof_id     bigint NOT NULL,
    institution_id bigint NOT NULL,
    study_form  integer NOT NULL,
    financing   integer NOT NULL,
    FOREIGN KEY (student_id) references STUDENT (id) on delete
cascade
        on update cascade,

```

```

        FOREIGN KEY (prof_id) references PROFESSION (id) on delete
cascade
        on update cascade,
        FOREIGN KEY (institution_id) references INSTITUTION (id) on
delete cascade
        on update cascade
);
ALTER TABLE STUDENT
    ADD COLUMN snils character varying(255);
CREATE SEQUENCE address_seq
    INCREMENT 1
    START 1
    MINVALUE 1
    MAXVALUE 9223372036854775807
    CACHE 1;
CREATE TABLE ADDRESS
(
    id          bigint          NOT NULL PRIMARY KEY
DEFAULT nextval('address_seq'::regclass),
    region      character varying(255) NOT NULL,
    district    character varying(255) NOT NULL,
    locality     character varying(255) NOT NULL,
    street       character varying(255) NOT NULL,
    building     character varying(255) NOT NULL,
    index        character varying(255) NOT NULL,
    flat         character varying(255)
);
INSERT INTO ADDRESS
VALUES (default, 'Тестовая область', 'Тестовый округ', 'город
Тест', 'Улица Тестовая', '1', '000000', '');
ALTER TABLE STUDENT
    ADD COLUMN address_id bigint not null DEFAULT 1;
ALTER TABLE STUDENT
    ADD FOREIGN KEY (address_id) references ADDRESS (id)
        on delete cascade on update cascade;
INSERT INTO institution
VALUES (default, 'КАДР №26'), (default, 'ДЗМ №1'), (default, 'КП
№11'), (default, 'МКАГ'), (default, 'ТК №21'), (default, 'КСУ
№10'), (default, 'КАИТ №20'), (default, 'КС №54'), (default,
'КАТ №9'), (default, 'КСУ №3'), (default, 'ТК №24'), (default,
'МК №1'), (default, 'МК №2'), (default, 'ТК №34'), (default,
'ЮК'), (default, 'Колледж тестирования');
INSERT INTO profession
VALUES (default, 'Сестринское дело'), (default, 'Лечебное
дело'), (default, 'Гостиничное дело'), (default, 'Туризм'),
(default, 'Реклама'), (default, 'Секретарь'), (default,
'Портной'), (default, 'Финансы'), (default, 'Повар'), (default,
'Кондитер'), (default, 'Пекарь'), (default, 'Программирование'),
(default, 'Анимация'), (default, 'Ювелир'), (default,
'Архитектура'), (default, 'Сварщик'), (default, 'Логистика'),
(default, 'Сервис на транспорте'), (default, 'Дизайн'),
(default, 'Коммерция'), (default, 'Мехатроника'), (default,
'Системное администрирование'), (default, 'Оптика'), (default,

```

```
'Экономика'), default, 'Бухгалтерский учет'), (default,
'Аналитика данных'), (default, 'Информатика'), (default,
'Тестомес'), (default, 'Смузихлеб'), (default, 'Мастер
закруток');
INSERT INTO prof2inst
VALUES (default, 2, 1), (default, 2, 2), (default, 12, 1),
(default, 12, 2), (default, 13, 1), (default, 13, 2), (default,
1, 5), (default, 1, 13), (default, 1, 19), (default, 1, 20),
(default, 1, 8), (default, 3, 12), (default, 3, 23), (default,
3, 13), (default, 3, 15), (default, 3, 16), (default, 3, 20),
(default, 3, 22), (default, 4, 12), (default, 4, 14), (default,
4, 15), (default, 4, 16), (default, 4, 19), (default, 5, 16),
(default, 5, 19), (default, 6, 3), (default, 6, 9), (default, 6,
10), (default, 6, 11), (default, 10, 9), (default, 10, 10),
(default, 10, 11), (default, 7, 12), (default, 7, 19), (default,
7, 22), (default, 8, 27), (default, 8, 26), (default, 8, 22),
(default, 9, 17), (default, 9, 18), (default, 9, 21), (default,
11, 4), (default, 11, 7), (default, 11, 19), (default, 11, 20),
(default, 11, 21), (default, 11, 22), (default, 11, 12),
(default, 11, 14), (default, 14, 3), (default, 14, 4), (default,
14, 5), (default, 14, 6), (default, 14, 24), (default, 14, 25),
(default, 14, 26), (default, 15, 24), (default, 15, 25),
(default, 15, 8), (default, 16, 28), (default, 16, 29),
(default, 16, 30);
```

Файл application.properties

```
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/chatbot
spring.datasource.username=postgres
spring.datasource.password=111111
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQL82Dialect
spring.jpa.show-sql=false
# Run
server.port=9090
server.servlet.context-path=/
#ElasticSearch
elasticsearch.uri=http://localhost:9200/
elasticsearch.index=assistant
elasticsearch.username=elastic
elasticsearch.password=3_d-DkINeUP4TshhjC8Y
```

Файл message.properties

```
profession.notFound=Специальность не найдена
profession.noSuchInInst=Данная специальность не найдена в
образовательной организации
institution.notFound=Образовательная организация не найдена
application.NotFound=Не найдены заявления по студенту
application.DocSerial=Неверный формат серии документа
application.DocNumber=Неверный формат номера документа
address.NotFound=Подходящий адрес не найден
```

gender.NotFound=Неверно задан код пола заявителя
financing.NotFound=Неверно задан код типа финансирования
studyForm.NotFound=Неверно задан код формы обучения

Файл ApplicationRepoTest.java

```
@SpringBootTest(classes = {ApplicationRepo.class,
DocumentRepo.class, StudentRepo.class, DocumentRepo.class,
ProfessionRepo.class, InstitutionRepo.class})
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@EnableAutoConfiguration
@EntityScan(basePackages = {"com.chatbot.chatbot"})
@EnableJpaRepositories(basePackages = {"com.chatbot.chatbot"})
public class ApplicationRepoTest {
    @Autowired
    private ApplicationRepo applicationRepo;
    @Autowired
    private StudentRepo studentRepo;
    @Autowired
    private ProfessionRepo professionRepo;
    @Autowired
    private InstitutionRepo institutionRepo;
    Application application;
    Student student;
    Document document;
    Profession profession;
    Institution institution;
    Address address;
    @BeforeEach
    public void init() {
        document = new Document("5600", "000000",
            "УМВД РОССИИ", LocalDate.now(), "502-000");
        address = new Address("Тест", "Тест", "Тестов",
            "Тестовая", "2", "440000", "5");
        student = new Student("Имя", "Фамилия", "Отчество",
            LocalDate.now(),
                Gender.MAN, document, "123-456-789", address);
        student = studentRepo.saveAndFlush(student);
        profession = new Profession("Тестовая специальность");
        profession = professionRepo.saveAndFlush(profession);
        institution = new Institution("Тест ООО");
        institution = institutionRepo.saveAndFlush(institution);
        application = new Application(student, profession,
            institution, StudyForm.FULL_TIME, FinancingType.BUDGET);
        applicationRepo.saveAndFlush(application);
    }
    @Test
    public void applicationRepo_findAllByStudentId_Snils() {
        List<Application> applicationList =
            applicationRepo.findAllByStudentId_Snils(student.getSnils());
        Assertions.assertEquals(student.getSnils(),
            applicationList.get(0).getStudentId().getSnils());
    }
}
```

```
}
```

Файл ApplicationServiceTest.java

```
@SpringBootTest(classes = {ApplicationRepo.class,
DocumentRepo.class, StudentRepo.class, AddressRepo.class,
    DocumentRepo.class, ProfessionRepo.class,
InstitutionRepo.class, ApplicationService.class,
    Prof2InstRepo.class,
ApplicationToCreateApplicationRespConverter.class,
ApplicationToViewConverter.class,
    MessageUtils.class, ElasticSearchApiClient.class,
SourceDataToAddressRespConverter.class, CreateAddress.class})
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@EnableAutoConfiguration
@EntityScan(basePackages = {"com.chatbot.chatbot"})
@EnableJpaRepositories(basePackages = {"com.chatbot.chatbot"})
public class ApplicationServiceTest {
    @Autowired
    private ApplicationRepo applicationRepo;
    @Autowired
    private StudentRepo studentRepo;
    @Autowired
    private ProfessionRepo professionRepo;
    @Autowired
    private InstitutionRepo institutionRepo;
    @Autowired
    private Prof2InstRepo prof2InstRepo;
    @Autowired
    private ApplicationToCreateApplicationRespConverter
createApplicationRespConverter;
    @Autowired
    private ApplicationToViewConverter
applicationToViewConverter;
    @Autowired
    private DocumentRepo documentRepo;
    @Autowired
    private SourceDataToAddressRespConverter
sourceDataToAddressRespConverter;
    @Autowired
    private AddressRepo addressRepo;
    @MockBean
    MessageUtils messageUtils;
    @MockBean
    ElasticSearchApiClient esApiClient;
    @Autowired
    ApplicationService applicationService = new
ApplicationService(applicationRepo, documentRepo, studentRepo,
        institutionRepo, professionRepo,
createApplicationRespConverter, applicationToViewConverter,
        messageUtils, prof2InstRepo, esApiClient,
sourceDataToAddressRespConverter, addressRepo);
```

```

Application application;
Student student;
Document document;
Profession profession;
Institution institution;
CreateApplicationReq createApplicationReq;
DocumentReq documentReq;
StudentReq studentReq;
Prof2Inst prof2Inst;
CreateAddress createAddress;
Address address;
@BeforeEach
public void init() {
    initApplication();
    initCreateRequest();
}
private void initCreateRequest() {
    documentReq = new DocumentReq("5555", "000000", "УМВД
России",
        LocalDate.now(), "500-001");
    createAddress = new CreateAddress("Тест", "Тест",
"Тестов", "Тестовая", "2", "440000", "5");
    studentReq = new StudentReq("Имя", "Отчество",
"Фамилия",
        Gender.WOMAN.ordinal(), LocalDate.now(), "123-
456-789", createAddress);
    createApplicationReq = new
CreateApplicationReq(institution.getId(), profession.getId(),
FinancingType.CONTRACT.ordinal(),
        StudyForm.EXTRAMURAL.ordinal(), documentReq,
studentReq);
}

private void initApplication() {
    document = new Document("5600", "000000",
        "УМВД РОССИИ", LocalDate.now(), "502-000");
    address = new Address("Тест", "Тест", "Тестов",
"Тестовая", "2", "440000", "5");
    student = new Student("Имя", "Фамилия", "Отчество",
LocalDate.now(),
        Gender.MAN, document, "123-456-789", address);
    student = studentRepo.saveAndFlush(student);

    profession = new Profession("Тестовая специальность");
    profession = professionRepo.saveAndFlush(profession);

    institution = new Institution("Тест ООО");
    institution = institutionRepo.saveAndFlush(institution);

    prof2Inst = new Prof2Inst(institution, profession);
    prof2InstRepo.saveAndFlush(prof2Inst);
}

```

```

        application = new Application(student, profession,
institution, StudyForm.FULL_TIME, FinancingType.BUDGET);
        applicationRepo.saveAndFlush(application);
    }

    @Test
    public void findApplications_Success() {
        FindApplicationsResp applications =
applicationService.findApplications(student.getSnils());
        Assertions.assertEquals(student.getSnils(),
applications.getSnils());
    }

    @Test
    public void findApplications_Fail() {

Mockito.when(messageUtils.getMessage(anyString())).thenReturn("T
ect");

        Assertions.assertThrows(NotFoundException.class, () -> {
            applicationService.findApplications("11111");
        });
    }

    @Test
    public void createApplication_checkRequestParams_Success() {
        CreateApplicationResp applicationResp =
applicationService.createApplication(createApplicationReq);
        Assertions.assertNotNull(applicationResp);
    }

    @Test
    public void createApplication_checkRequestParams_Fail_1() {
        createApplicationReq.setProfessionId(600L);
        Assertions.assertThrows(NotFoundException.class, () -> {

applicationService.createApplication(createApplicationReq);
        });
    }

    @Test
    public void createApplication_checkRequestParams_Fail_2() {
        createApplicationReq.setInstitutionId(600L);
        Assertions.assertThrows(NotFoundException.class, () -> {

applicationService.createApplication(createApplicationReq);
        });
    }

    @Test
    public void createApplication_checkRequestParams_Fail_3() {
        createApplicationReq.setInstitutionId(1L);
        createApplicationReq.setProfessionId(2L);
        Assertions.assertThrows(NotFoundException.class, () -> {

```

```

        applicationService.createApplication(createApplicationReq);
    });
}

@Test
public void createApplication_checkRequestParams_Fail_4() {
    createApplicationReq.getStudent().setGender(5);
    Assertions.assertThrows(NotFoundException.class, () -> {
        applicationService.createApplication(createApplicationReq);
    });
}

@Test
public void createApplication_checkRequestParams_Fail_5() {
    createApplicationReq.setFinancing(5);
    Assertions.assertThrows(NotFoundException.class, () -> {
        applicationService.createApplication(createApplicationReq);
    });
}

@Test
public void createApplication_checkRequestParams_Fail_6() {
    createApplicationReq.setStudyForm(6);
    Assertions.assertThrows(NotFoundException.class, () -> {
        applicationService.createApplication(createApplicationReq);
    });
}

@Test
public void createApplication_checkDocument_Fail_1() {
    createApplicationReq.getDocument().setSerial("sd5");
    Assertions.assertThrows(ApplicationException.class, () -
> {
        applicationService.createApplication(createApplicationReq);
    });
}

@Test
public void createApplication_checkDocument_Fail_2() {
    createApplicationReq.getDocument().setNumber("sd5");
    Assertions.assertThrows(ApplicationException.class, () -
> {
        applicationService.createApplication(createApplicationReq);
    });
}
}

```

Файл InstitutionRepoTest.java

```
@SpringBootTest(classes = {InstitutionRepo.class})
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@EnableAutoConfiguration
@EntityScan(basePackages = {"com.chatbot.chatbot"})
@EnableJpaRepositories(basePackages = {"com.chatbot.chatbot"})
public class InstitutionRepoTest {
    @Autowired
    private InstitutionRepo institutionRepo;
    Institution institution;
    @BeforeEach
    public void init() {
        institution = new Institution("Тест ООО");
        institution = institutionRepo.saveAndFlush(institution);
    }
    @Test
    public void findByInstName() {
        Optional<Institution> instByName =
institutionRepo.findByInstName(institution.getInstName());
        Assertions.assertNotNull(instByName);
        Assertions.assertEquals(institution.getInstName(),
instByName.get().getInstName());
        Assertions.assertEquals(institution.getId(),
instByName.get().getId());
    }
}
```

Файл InstitutionServiceTest.java

```
@SpringBootTest(classes = {InstitutionRepo.class,
InstitutionService.class, InstitutionToViewConverter.class})
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@EnableAutoConfiguration
@EntityScan(basePackages = {"com.chatbot.chatbot"})
@EnableJpaRepositories(basePackages = {"com.chatbot.chatbot"})
public class InstitutionServiceTest {
    @Autowired
    private InstitutionRepo institutionRepo;
    @Autowired
    private InstitutionToViewConverter
institutionToViewConverter;
    @Autowired
    InstitutionService institutionService = new
InstitutionService(institutionRepo, institutionToViewConverter);
    Institution institution;
    @BeforeAll
    public void init() {
        institution = new Institution("Тест ООО 1");
        institution = institutionRepo.saveAndFlush(institution);
    }
    @Test
    public void getInstitutionList() {
```

```

        Pageable pageable = PageRequest.of(0, 10,
Sort.unsorted());
        Page<InstitutionView> institutionList =
institutionService.getInstitutionList(pageable);
        Assertions.assertNotEquals(0,
institutionList.getTotalElements());
    }
}

```

Файл ManualServiceTest.java

```

@SpringBootTest(classes = {ManualService.class})
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@EnableAutoConfiguration
@EntityScan(basePackages = {"com.chatbot.chatbot"})
@EnableJpaRepositories(basePackages = {"com.chatbot.chatbot"})
public class ManualServiceTest {
    @Autowired
    private ManualService manualService = new ManualService();
    @Test
    public void getGenderList() {
        Gender[] genders = Gender.values();
        List<ManualResp> genderList =
manualService.getGenderList();
        Optional<ManualResp> man = genderList.stream()
            .filter(gender -> gender.getCode() ==
Gender.MAN.ordinal()).findAny();
        Optional<ManualResp> woman = genderList.stream()
            .filter(gender -> gender.getCode() ==
Gender.WOMAN.ordinal()).findAny();
        Assertions.assertEquals(genders.length,
genderList.size());
        Assertions.assertNotNull(man);
        Assertions.assertNotNull(woman);
    }
    @Test
    public void getFinancingList() {
        FinancingType[] financingTypes = FinancingType.values();
        List<ManualResp> financingList =
manualService.getFinancingList();
        Optional<ManualResp> budget = financingList.stream()
            .filter(type -> type.getCode() ==
FinancingType.BUDGET.ordinal()).findAny();
        Optional<ManualResp> contract = financingList.stream()
            .filter(type -> type.getCode() ==
FinancingType.CONTRACT.ordinal()).findAny();

        Assertions.assertEquals(financingTypes.length,
financingList.size());
        Assertions.assertNotNull(budget);
        Assertions.assertNotNull(contract);
    }
}

```

```

@Test
public void getStudyFormList() {
    StudyForm[] studyForms = StudyForm.values();
    List<ManualResp> studyFormList =
manualService.getStudyFormList();
    Optional<ManualResp> fullTime = studyFormList.stream()
        .filter(form -> form.getCode() ==
StudyForm.FULL_TIME.ordinal()).findAny();
    Optional<ManualResp> extramural = studyFormList.stream()
        .filter(form -> form.getCode() ==
StudyForm.EXTRAMURAL.ordinal()).findAny();
    Assertions.assertEquals(studyForms.length,
studyFormList.size());
    Assertions.assertNotNull(fullTime);
    Assertions.assertNotNull(extramural);
}
}

```

Файл Prof2InstServiceTest.java

```

@SpringBootTest(classes = {Prof2InstRepo.class,
InstitutionToViewConverter.class,
ProfessionToViewConverter.class,
    Prof2InstService.class, ProfessionRepo.class,
InstitutionRepo.class})
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@EnableAutoConfiguration
@EntityScan(basePackages = {"com.chatbot.chatbot"})
@EnableJpaRepositories(basePackages = {"com.chatbot.chatbot"})
public class Prof2InstServiceTest {
    @Autowired
    private Prof2InstRepo prof2InstRepo;
    @Autowired
    private ProfessionRepo professionRepo;
    @Autowired
    private InstitutionRepo institutionRepo;
    @Autowired
    private InstitutionToViewConverter
institutionToViewConverter;
    @Autowired
    private ProfessionToViewConverter professionToViewConverter;
    @Autowired
    Prof2InstService prof2InstService = new
Prof2InstService(prof2InstRepo,
        institutionToViewConverter,
professionToViewConverter);
    Profession profession;
    Institution institution;
    Prof2Inst prof2Inst;
    @BeforeAll
    public void init() {
        profession = new Profession("Специальность 1");
        profession = professionRepo.saveAndFlush(profession);
    }
}

```

```

        institution = new Institution("ООО №1");
        institution = institutionRepo.saveAndFlush(institution);
        prof2Inst = new Prof2Inst(institution, profession);
        prof2Inst = prof2InstRepo.saveAndFlush(prof2Inst);
    }
    @Test
    public void findInstitutions() {
        Pageable pageable = PageRequest.of(0, 10,
Sort.unsorted());
        Page<InstitutionView> institutions =
prof2InstService.findInstitutions(profession.getId(), pageable);
        Assertions.assertEquals(1,
institutions.getTotalElements());
        Assertions.assertEquals(institution.getId(),
institutions.getContent().get(0).getId());
    }
    @Test
    public void findProfessions() {
        Pageable pageable = PageRequest.of(0, 10,
Sort.unsorted());
        Page<ProfessionView> professions =
prof2InstService.findProfessions(institution.getId(), pageable);
        Assertions.assertEquals(1,
professions.getTotalElements());
        Assertions.assertEquals(profession.getId(),
professions.getContent().get(0).getId());
    }
}

```

Файл ProfessionRepoTest.java

```

@SpringBootTest(classes = {ProfessionRepo.class})
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@EnableAutoConfiguration
@EntityScan(basePackages = {"com.chatbot.chatbot"})
@EnableJpaRepositories(basePackages = {"com.chatbot.chatbot"})
public class ProfessionRepoTest {
    @Autowired
    private ProfessionRepo professionRepo;
    Profession profession;
    @BeforeEach
    public void init() {
        profession = new Profession("Тестовая специальность 2");
        profession = professionRepo.saveAndFlush(profession);
    }
    @Test
    public void findByName() {
        Optional<Profession> profByName =
professionRepo.findByName(profession.getName());
        Assertions.assertNotNull(profByName);
        Assertions.assertEquals(profession.getName(),
profByName.get().getName());
    }
}

```

```

        Assertions.assertEquals(profession.getId(),
profByName.getId().getId());
    }
}

```

Файл ProfessionServiceTest.java

```

@SpringBootTest(classes = {ProfessionRepo.class,
ProfessionService.class, ProfessionToViewConverter.class})
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
@EnableAutoConfiguration
@EntityScan(basePackages = {"com.chatbot.chatbot"})
@EnableJpaRepositories(basePackages = {"com.chatbot.chatbot"})
public class ProfessionServiceTest {
    @Autowired
    private ProfessionRepo repo;
    @Autowired
    private ProfessionToViewConverter professionToViewConverter;
    @Autowired
    private ProfessionService professionService = new
ProfessionService(repo, professionToViewConverter);
    Profession profession;
    @BeforeAll
    public void init() {
        profession = new Profession("Тестовая специальность 1");
        profession = repo.saveAndFlush(profession);
    }
    @Test
    public void findAll() {
        Pageable pageable = PageRequest.of(0, 10,
Sort.unsorted());
        Page<ProfessionView> professionViews =
professionService.findAll(pageable);
        Assertions.assertNotEquals(0,
professionViews.getTotalElements());
    }
}

```

ПРЕПРОЦЕССОР ВАЛИДАЦИИ И КОРРЕКТИРОВКИ ВХОДНЫХ ДАННЫХ ДЛЯ ВИРТУАЛЬНОГО АССИСТЕНТА

Выполнил студент гр.21BBBm1

Нешко Д.О.

Научный руководитель:

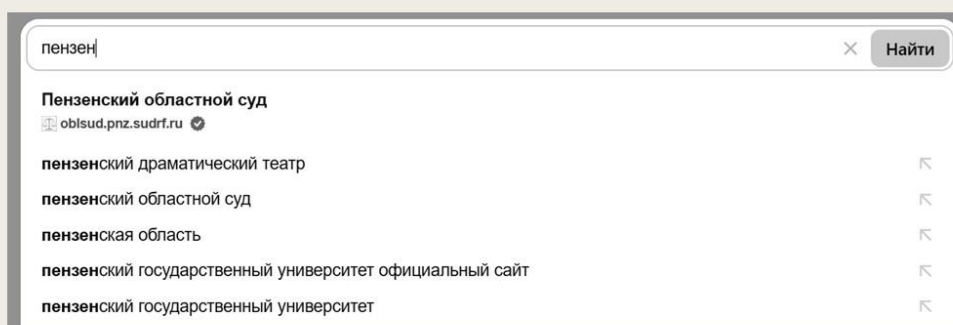
к.т.н., доцент Синев М.П.

г.Пенза, 2023 г.

1

- **Предмет исследования:** препроцессор валидации и корректировки входных данных для виртуального ассистента для дистанционной подачи заявлений на зачисление в образовательные организации
- **Объект исследования:** сервис-виртуальный ассистент, включающий в себя препроцессор корректировки и валидации входных данных пользователя в режиме реального времени
- **Задачи исследования:**
 - сбор данных для их дальнейшей передачи в образовательную организацию в корректном виде;
 - валидация и корректировка введенных пользователем данных;
 - реализация быстрого поиска данных по заданным параметрам.

Поисковые подсказки



Способы быстрого поиска данных

APACHE
LUCENETM

Solr



elastic

Сравнение поисковых серверов ElasticSearch и Solr

4

	Solr	ElasticSearch
Возможности использования	Поисковая система с открытым исходным кодом, которая предназначена для корпоративного полнотекстового поиска. Может использоваться для поиска информации и аналитики	Поисковая система, предназначенная для обработки временных рядов, аналитики, масштабирования. Также может выполнять функции полнотекстового поиска
Преимущества	Широкий функционал для полнотекстового поиска	Самый популярный поисковый сервер Обладает удобным API Имеет хорошую документацию Конструкции запросов просты Является бесплатным
Недостатки	Сложность запросов Ограниченная функциональность Ограниченные возможности аналитики Необходимо использование ZooKeeper для координирования работы Сложность балансировки	Закрытый исходный код

Поиск в ElasticSearch

5

- В новых версиях ElasticSearch используется функция ранжирования Okapi BM25.

$$score(D, Q) = \sum_{i=1}^n IDF(q_i) * \frac{f(q_i, D) * (k_1 + 1)}{f(q_1, D) + k_1 * \left(1 - b + b * \frac{|D|}{avgdl}\right)} \quad (1)$$

$$IDF = \log \frac{N}{n(q_i)} \quad (2)$$

Механизм поиска multi_match

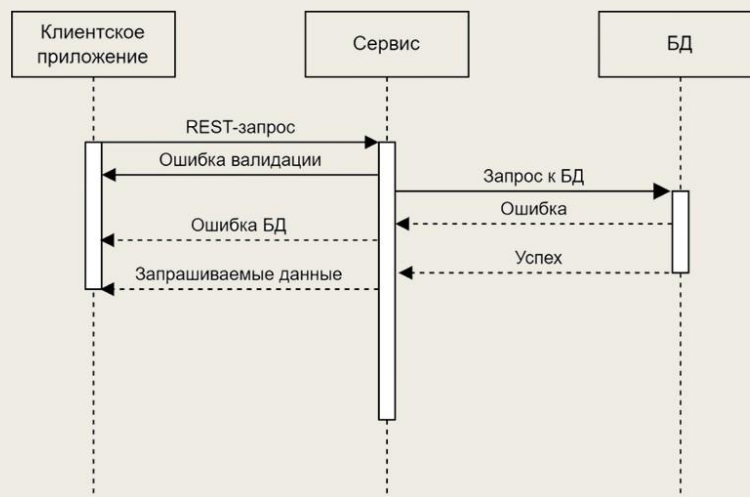
- Параметр fuzziness - настройка нечеткого поиска, автодополнения и корректировки
- Стратегия поиска best_fields
- Параметр tie_breaker - увеличения показателей релевантности документов, соответствующих нескольким предложениям запроса
- Параметр minimum_should_match - указание минимального процента сходства документа запросу
- Параметр size - ограничение количества документов в выдаче

Поиск адреса по строке «проезд 2»

```
[
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "10-й Малиновый проезд",
    "building": "2",
    "index": "440015"
  },
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "1-й проезд 40 лет Октября",
    "building": "2",
    "index": "440031"
  }
],
[
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "10-й Малиновый проезд",
    "building": "5",
    "index": "440015"
  },
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "10-й Малиновый проезд",
    "building": "6",
    "index": "440015"
  },
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "10-й Малиновый проезд",
    "building": "7",
    "index": "440015"
  }
]
```

```
[
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "Проезд Пушкина",
    "building": "2",
    "index": "440000"
  },
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "Проезд Пушкина",
    "building": "3",
    "index": "440000"
  },
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "Улица Пушкина",
    "building": "1",
    "index": "440008"
  },
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "Улица Пушкина",
    "building": "7",
    "index": "440008"
  },
  {
    "region": "Пензенская область",
    "district": "городской округ город Пенза",
    "locality": "город Пенза",
    "street": "Улица Пушкина",
    "building": "8",
    "index": "440008"
  }
]
```

Алгоритм работы сервиса



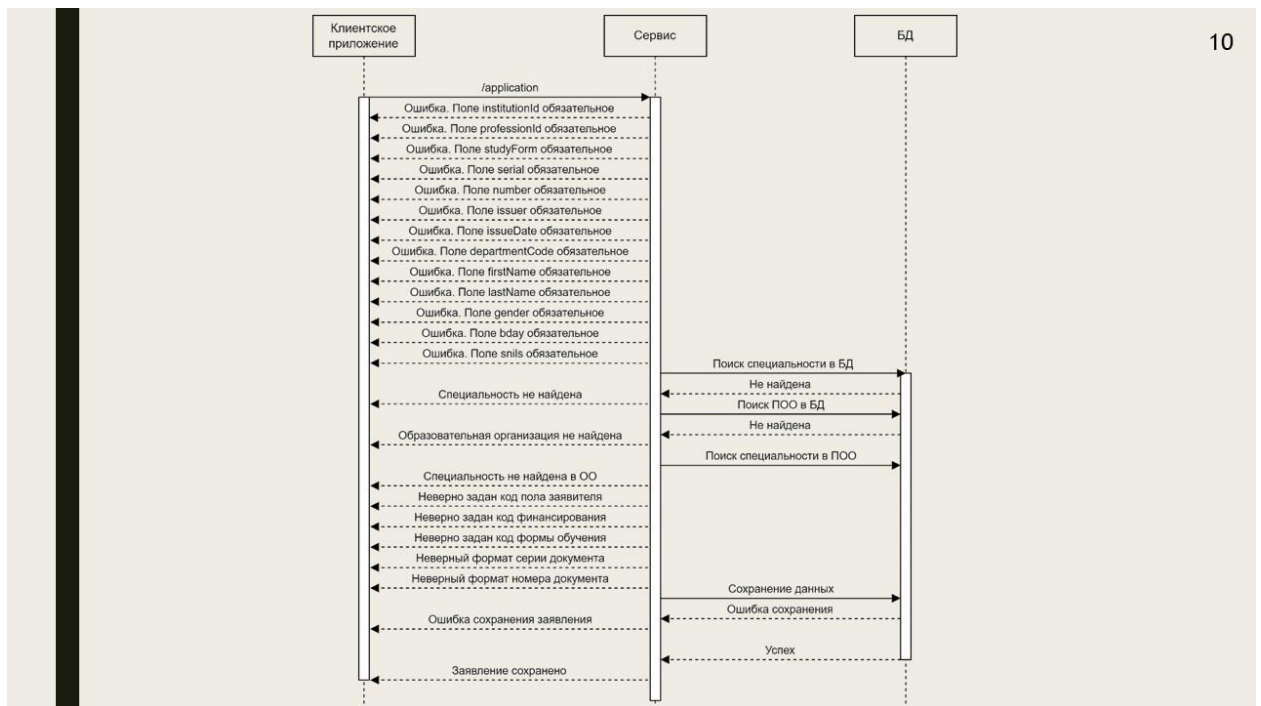
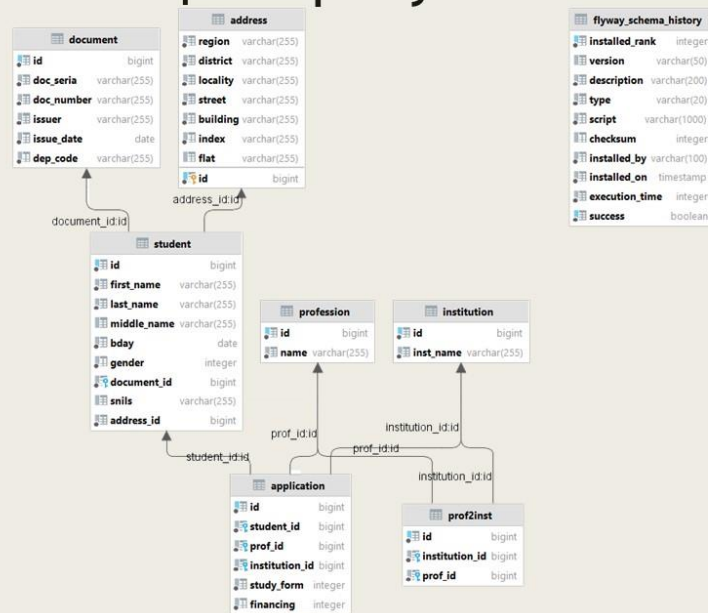


Схема таблиц и атрибутов

12



Описание API сервиса

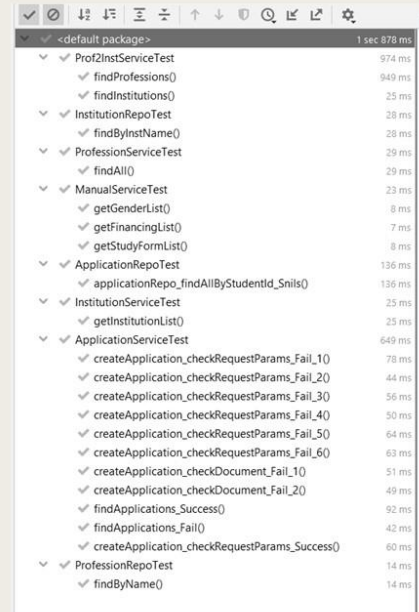
13

Заявления	
POST	/application Подать заявление
GET	/application/student/{snils} Поиск заявлений студента по СНИЛС
GET	/application/findAddress Поиск адреса
Специальности	
GET	/profession/search Поиск специальности по названию
GET	/profession/list Получение списка всех доступных специальностей
GET	/profession/institution/list Поиск образовательных организаций, реализующих конкретную специальность
Справочные данные	
GET	/manual/studyForm Получение справочника Форма обучения
GET	/manual/gender Получение справочника Пол
GET	/manual/financing Получение справочника Финансирование
Образовательные организации	
GET	/institution/search Поиск образовательной организации по названию
GET	/institution/profession/list Поиск специальностей в образовательной организации
GET	/institution/list Получение списка доступных образовательных организаций

Методы испытания

14

- Проведено функциональное тестирование
- Проект включает в себя unit-тесты



Test Name	Execution Time
ProfZInstServiceTest	974 ms
findProfessions()	949 ms
findInstitutions()	25 ms
InstitutionRepoTest	28 ms
findByInstName()	28 ms
ProfessionServiceTest	29 ms
findAll()	29 ms
ManualServiceTest	23 ms
getGenderList()	8 ms
getFinancingList()	7 ms
getStudyFormList()	8 ms
ApplicationRepoTest	136 ms
applicationRepo_findAllByStudentId_Snails()	136 ms
InstitutionServiceTest	25 ms
getInstitutionList()	25 ms
ApplicationServiceTest	649 ms
createApplication_checkRequestParams_Fail_10	78 ms
createApplication_checkRequestParams_Fail_20	44 ms
createApplication_checkRequestParams_Fail_30	56 ms
createApplication_checkRequestParams_Fail_40	50 ms
createApplication_checkRequestParams_Fail_50	64 ms
createApplication_checkRequestParams_Fail_60	63 ms
createApplication_checkDocument_Fail_10	51 ms
createApplication_checkDocument_Fail_20	49 ms
findApplications_Success()	92 ms
findApplications_Fail()	42 ms
createApplication_checkRequestParams_Success()	60 ms
ProfessionRepoTest	14 ms
findByName()	14 ms

Заключение

15

- В ходе выполнения магистерской диссертации была проанализирована предметная область, определен стек используемых технологий, рассмотрены способы осуществления быстрого поиска с учетом корректировки пользовательского запроса, определены необходимые функциональные составляющие для приложения виртуального ассистента, разработана подробная API-документация для интеграции ПО в иные системы, выполнена валидация входящих данных для исключения ошибок, связанных с человеческим фактором.
- Сервис включает в себя модульные тесты, было проведено функциональное тестирование. Испытания и оценка функционала подтвердили, что препроцессор валидации и корректировки работает корректно и удовлетворяют все поставленные задачи.

- За время проектирования и реализации данного проекта была освоена работа с NoSQL базами данных, в частности, с ElasticSearch, освоен механизм нечеткого поиска в ElasticSearch.
- Результаты магистерской диссертации были апробированы в рамках научной статьи «Virtual assistant for remote submission of applications for enrollment in educational organizations», представленной на конференции «Information Technologies and Intelligent Decision Making System» (ITIDMS-II-2021), статья опубликована в журнале «CEUR workshop proceedings», проиндексирован в Scopus.
- Оформлено свидетельство о государственной регистрации программы для ЭВМ «Виртуальный ассистент для дистанционной подачи заявлений на зачисление в образовательные организации», номер свидетельства: RU 2023615616.

**БЛАГОДАРЮ ЗА
ВНИМАНИЕ!**