МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

Факультет компьютерных и физико-математических наук

Кафедра прикладной математики и информатики

Допущена к защите

Заведующий кафедрой прикладной математики и информатики

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Е. В. Разова

Выпускная квалификационная работа

РАЗРАБОТКА ЧАТ-БОТА В ПОДДЕРЖКУ АБИТУРИЕНТА ВЯТГУ

Выполнил

студент гр. ФИб-4301-51-00 Ощепков Д. О.

(шифр) (подпись) (Ф. И. О.) (дата)

Руководитель

доцент каф. ПМИ \_\_\_\_\_\_\_\_ Разова Е. В. \_\_\_\_\_\_\_\_

(уч. степень, должность) (подпись) (Ф. И. О.) (дата)

Киров 2025

**Р Е Ф Е Р А Т**

**Содержание**

[Введение 6](#_Toc199116251)

[1. Обзор предметной области и существующих решений 8](#_Toc199116252)

[1.1 Telegram-боты 8](#_Toc199116253)

[1.2 Способы получения информации от пользователей 8](#_Toc199116254)

[1.3 Основные ограничения Telegram-ботов 8](#_Toc199116255)

[1.4 Обзор аналогов 9](#_Toc199116256)

[1.4.1 Admissions KFU (Telegram @kpfu\_admissions\_bot) 9](#_Toc199116257)

[1.4.2 Московский политех (Telegram @MoscowPolytechBot) 11](#_Toc199116258)

[1.4.3 Хочу в Политех (Telegram @SPbPUbot) 12](#_Toc199116259)

[1.5 Составление требований 16](#_Toc199116260)

[1.5.1 Пользовательские требования 16](#_Toc199116261)

[1.5.2 Бизнес-правила 16](#_Toc199116262)

[1.5.3 Функциональные требования 16](#_Toc199116263)

[1.5.4 Функциональные требования 17](#_Toc199116264)

[1.5.5 Технические требования 17](#_Toc199116265)

[1.5.6 Требования к интерфейсам 17](#_Toc199116266)

[1.5.7 Требования к данным 17](#_Toc199116267)

[2. Аргументация технических решений 19](#_Toc199116268)

[2.1 Языки программирования 19](#_Toc199116269)

[2.1.1 Python (https://www.python.org/) 19](#_Toc199116270)

[2.1.2 C# (https://dotnet.microsoft.com/en-us/languages/csharp) 19](#_Toc199116271)

[2.1.3 Java (https://www.java.com/ru/) 20](#_Toc199116272)

[2.1.4 Обоснование выбора Python 21](#_Toc199116273)

[2.2 Фреймворки 22](#_Toc199116274)

[2.2.1 Python-telegram-bot (https://github.com/python-telegram-bot/python-telegram-bot) 22](#_Toc199116275)

[2.2.2 Aiogram (https://github.com/aiogram/aiogram) 22](#_Toc199116276)

[2.2.3 Pyrogram (https://github.com/pyrogram/pyrogram) 23](#_Toc199116277)

[2.2.4 Обоснование выбора aiogram 23](#_Toc199116278)

[2.3 СУБД 24](#_Toc199116279)

[2.3.1 PostgreSQL (https://github.com/postgres/postgres) 24](#_Toc199116280)

[2.3.2 SQLite (https://github.com/sqlite/sqlite) 25](#_Toc199116281)

[2.3.3 MySQL (https://github.com/mysql/mysql-server) 25](#_Toc199116282)

[2.3.4 Обоснование выбора PostgreSQL 26](#_Toc199116283)

[2.4 Система семантического поиска по текстовым данным 26](#_Toc199116284)

[Далее потребуется несколько понятий. 26](#_Toc199116285)

[2.4.1 Необходимость векторного поиска 27](#_Toc199116286)

[2.4.2 Обзор векторных баз данных 27](#_Toc199116287)

[2.4.3 Обоснование выбора Qdrant 29](#_Toc199116288)

[2.5 Системы хранения кэшируемых данных 29](#_Toc199116289)

[2.5.1 Redis (https://github.com/redis/redis) 29](#_Toc199116290)

[2.5.2 Memcached (https://github.com/memcached/memcached) 30](#_Toc199116291)

[2.5.3 Dragonfly (https://github.com/dragonflydb/dragonfly) 30](#_Toc199116292)

[2.5.4 Обоснование выбора Redis 31](#_Toc199116293)

[2.6 Среда исполнения и развертывание приложения 32](#_Toc199116294)

[2.6.1 Проблемы локального развертывания 32](#_Toc199116295)

[2.6.2 Docker в качестве решения проблемы (https://www.docker.com/) 32](#_Toc199116296)

[2.7 Итог техническим решениям 33](#_Toc199116297)

[2.8 Выбор лингвистической модели векторизации 34](#_Toc199116298)

[3. Разработка чат-бота в поддержку абитуриента ВятГУ 34](#_Toc199116299)

[3.1 Архитектура системы 34](#_Toc199116300)

[3.1.1 Описание компонентов 34](#_Toc199116301)

[3.1.2 Процесс развертывания 36](#_Toc199116302)

[3.1.3 Инфраструктура CI/CD 37](#_Toc199116303)

[3.1.4 Архитектура бота 37](#_Toc199116304)

[3.1.5 Структура базы данных 39](#_Toc199116305)

[3.2 Сценарии взаимодействия с пользователем 42](#_Toc199116306)

[3.2.1 Вопрос о поступлении 43](#_Toc199116307)

[3.2.2 Получение рекомендованных направлений 49](#_Toc199116308)

[3.3 Тестирование 53](#_Toc199116309)

[3.4 Перспективы развития 54](#_Toc199116310)

[Заключение 54](#_Toc199116311)

Введение

**Актуальность темы** исследования обусловлена необходимости повышения качества и доступности информационной поддержки абитуриентов ВятГУ. Часто поступающие сталкиваются с проблемами информационного характера при выборе направления подготовки. При этом более 60 миллионов россиян пользуются Telegram ежедневно (<https://www.vedomosti.ru/technology/articles/2024/07/11/1049269-bolee-60-mln-rossiyan-ezhednevno-polzuyutsya-telegram>). Среди молодежи суточный охват занимает 76% (https://adindex.ru/news/digital/2024/11/13/327316.phtml). Однако, на сегодняшний день ВятГУ не имеет специализированного телеграмм-бота в поддержку абитуриента, который бы оперативно и структурировано предоставлял информацию, интересующую поступающего. Это делает разработку Telegram-бота востребованным и актуальным направлением исследования, которая потенциально улучшить осведомленность будущих студентов ВятГУ.

**Объектом исследования** является программные средства предоставления справочной информации и автоматической рекомендации на основе фильтров и методов машинного обучения в формате чат-бота.

Предметом исследования выступает Telegram-бот, обеспечивающий доступ к информации и предоставляющий рекомендации на основе предпочтений абитуриента.

**Цель работы** – разработка Telegram-бота для предоставления справочной информации и рекомендаций абитуриентам ВятГУ на основе данных, размещенных на официальных источниках ВятГУ.

Для реализации поставленной цели необходимо решить следующие **задачи**:

- собрать информацию с сайта для абитуриентов ВятГУ (<https://new.vyatsu.ru/>);

- систематизировать полученную информацию для эффективного доступа к ней;

- разработать архитектуру Telegram-бота с удобным пользовательским интерфейсом;

- реализовать рекомендательную систему на основе описания направлений;

- провести тестирование работоспособности и удобства использования чат-бота.

В рамках разработки Telegram-бота использовались прикладные методы программной инженерии. Методами системного анализа была выявлена оптимальная структура данных и логика взаимодействия с пользователем. Для обработки пользовательских запросов использовались элементы обработки естественного языка и простейшая классификация. Были реализованы и протестированы основные сценарии использования, включая навигацию по информационным разделам и подбор образовательных программ по заданным экзаменам. В качестве источника данных использовалась информация с официального сайта ВятГУ, прошедшая этап предварительной структуризации и нормализации.

1. Обзор предметной области и существующих решений
   1. Telegram-боты

Telegram-боты – это программные агенты, которые взаимодействуют с пользователем через интерфейс мессенджера Telegram. Они используют Telegram Bot API (https://core.telegram.org/bots/api) для взаимодействия с серверами Telegram. Эти агенты позволяют организовать удобный интерфейс от навигации по информационной базе до интеграции с внешними сервисами.

* 1. Способы получения информации от пользователей

Для получения сообщений от пользователей Telegram Bot API предоставляет два основных метода взаимодействия с серверами: long polling и webhook.

Long polling – бот периодически запрашивает у Telegram наличие новых сообщений. Благодаря такому подходу бот не требует публичного IP-адреса. (<https://en.wikipedia.org/wiki/Push_technology#Long_polling>).

Webhook – метод, при котором Telegram сам отправляет сообщения на указанный ботом URL-адрес. Этот способ требует открытый IP-адрес и защищенное соединение по HTTPS с действующим SSL-сертификатом. Этот способ более производительный, чем long polling (<https://core.telegram.org/bots/webhooks>).

* 1. Основные ограничения Telegram-ботов

Отправка сообщений имеет следующие ограничения:

- не более 1 сообщения в секунду в каждый чат;

- в группах и каналах: не более 20 сообщений в минуту на одну группу или канал;

- не более 30 сообщений в секунду для одного бота в целом.

Превышение этих лимитов может привести к ошибке 429 (Too Many Requests). Кратковременные превышения лимитов допустимы.

Сообщения тоже имеют ограничения:

- текст сообщения не должен превышать 4096 символов

- максимальный размер загружаемого файла равен 50 мегабайтам, но при использовании локального сервера Bot API максимальный размер равен 2000 мегабайт (<https://core.telegram.org/bots/faq#my-bot-is-hitting-limits-how-do-i-avoid-this>).

Ограничения на inline-клавиатуры и кнопки следующие:

- максимальное количество кнопок в одной строке: до 8 кнопок;

- общее количество кнопок в сообщении: до 100 кнопок;

- максимальный размер данных callback\_data: до 64 байт.

Превышение этих лимитов может привести к ошибке 400: REPLY\_MARKUP\_TOO\_LONG (https://core.telegram.org/method/messages.editMessage).

Telegram предоставляет возможность увеличения лимитов отправки сообщений для ботов через платную функцию Paid Broadcasts. Это позволяет боту отправлять до 1000 сообщений в секунду, но при выполнении следующих условий:

- более 100 000 звезд (собственная внутренняя валюта);

- более 100 000 активных пользователей в месяц.

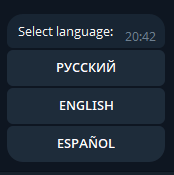
Каждое сообщение свыше лимита 30 сообщений в секунду оплачивается из баланса бота (https://core.telegram.org/bots/api#paid-broadcasts).

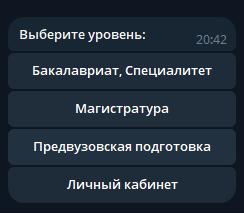
Особенно важно учитывать ограничения на отправку сообщений в секунду, так как это может привести к неожиданному отказу бота при высоких нагрузках. При несоблюдении других остальных лимитов, бот просто не будет работать еще в тестовой среде.

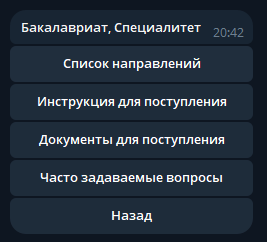
* 1. Обзор аналогов
     1. Admissions KFU (Telegram @kpfu\_admissions\_bot)

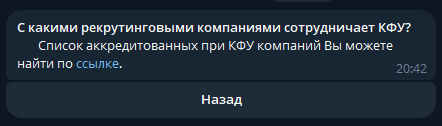
Официальный бот для иностранных абитуриентов Казанского (Приволжского) федерального университета (КФУ). Предназначен для повышения информированности абитуриентов из-за рубежа. Бот позволяет подписываться на интересующие программы подготовки, присылает уведомления о сроках приёма документов и начале вступительных испытаний. С его помощью иностранный студент может узнать о программах обучения, конкурсах, минимальных баллах и необходимых документах. Поддерживает русский, английский и испанский языки.

Этот бот не имеет возможности задания вопроса в свободной форме и не имеет функциональности рекомендации какого-либо направления вуза. Как было сказано ранее, бот предназначен для студентов из-за рубежа.





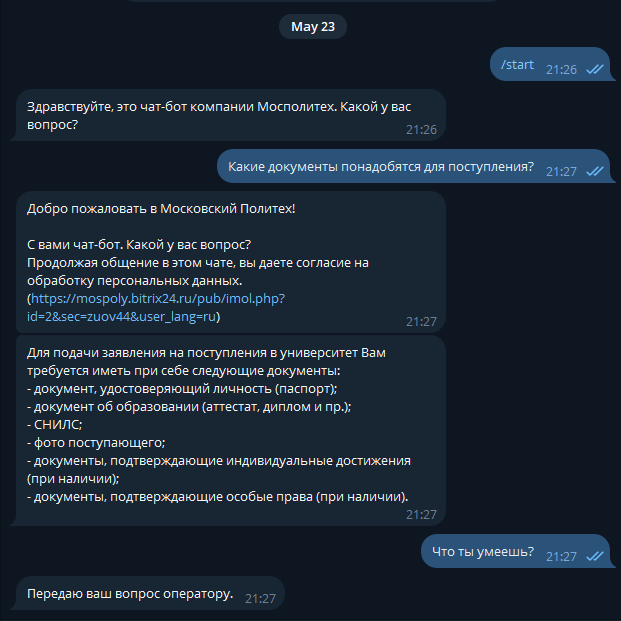




* + 1. Московский политех (Telegram @MoscowPolytechBot)

Бот Московского политехнического университета. Позволяет задавать вопросы в свободной форме. Если не находит ответ в базе направляет на оператора.

Не ясно, что в целом умеет бот. По всей видимости, он предусматривает только задание вопроса в свободной форме.

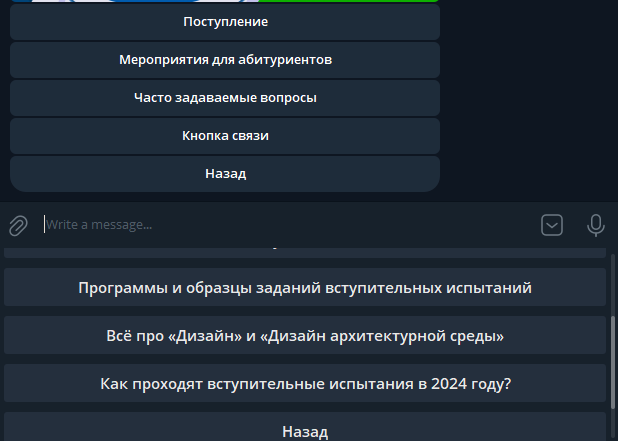


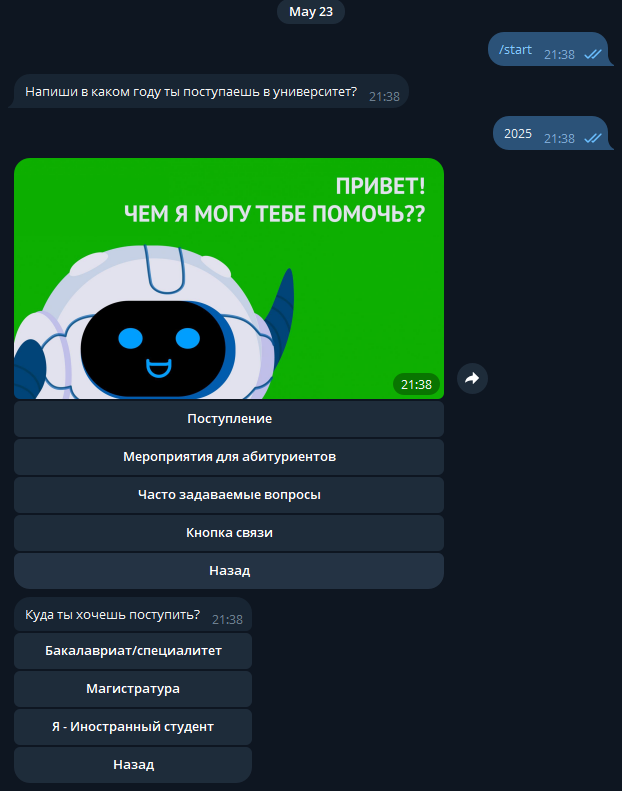
* + 1. Хочу в Политех (Telegram @SPbPUbot)

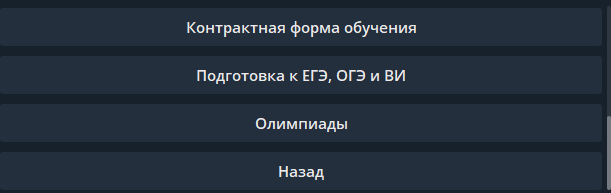
Бот университета СПбГМТУ (Санкт-Петербургский государственный морской технический университет) представляет собой информационный сервис для абитуриентов. Позволяет открывать нормативные документы не выходя из телеграмма через мини приложения (<https://core.telegram.org/bots/webapps>), cодержит информацию для иностранных студентов. Предоставляет сведения о приёмной кампании, нормативных документах, мероприятиях и контактах приёмной комиссии. Стоит отметить, что к большинству ответов приложена ссылка, чтобы проверить актуальность данных, либо в случае документа дата вступления в силу. Бот хорошо структурирован и позволяет выбрать год планируемого поступления.

Сочетание Inline и обычной клавиатуры заставляет пользователя часто переключаться между ними, что не очень удобно. Картинки занимают много места, но не несут никакой пользы. Также отсутствует раздел с рекомендациями.









* 1. Составление требований

На основе преимуществ и недостатков рассмотренных решений и на основе требований заказчика были составлены следующие требования к боту.

* + 1. Пользовательские требования

- Пользователь должен иметь возможность получить актуальную информацию о направлениях подготовки, где возможно требуется прикрепить ссылку на официальный источник;

- требуется хорошо структурировать меню бота для наилучшего пользовательского опыта;

- пользователь должен иметь возможность получить рекомендации на основе интересов, уровня ЕГЭ и других параметров;

- бот должен уметь принимать вопросы в свободной форме и отвечать на основе базы знаний.

* + 1. Бизнес-правила

- информация должна поступать только из официальных источников;

- основная цель бота – помочь абитуриенту с вопросами и с определение направления подготовки;

- бот не должен сообщать сведения, не подтвержденные в официальных источниках.

* + 1. Функциональные требования

Структурированное меню должно включать:

- пункт частые вопросы, разбитые по категориям;

- пункт рекомендации направления.

Обрабатывая естественный язык, система должна давать ответ из базы знаний, при отсутствии подходящего ответа требуется предложить переформулировать запрос

Рекомендательная система должна:

- включать выбор интересов или предметов ЕГЭ;

- давать возможность сформулировать вопрос в свободной форме без выборов ЕГЭ и интересов;

- уметь подбирать подходящих направлений.

* + 1. Функциональные требования

Система должна справляться с опечатками пользователя.

* + 1. Технические требования

- Использовать очередь для соблюдения лимитов на отправку сообщений;

- соблюдать ограничения на сообщения;

- использовать сервисы для кеширования запросов к базе данных.

* + 1. Требования к интерфейсам

- Бот должен использовать inline кнопки там, где это возможно;

- интерфейс должен быть интуитивно понятным и единообразным. Он не должен требовать специальных технических знаний;

- ответы должны быть краткими и адаптированными под мобильные устройства;

- интерфейс должен быть спроектирован так, чтобы его можно было легко дополнить новыми функциями.

* + 1. Требования к данным

Данные должны быть нормализованы и хранится в СУБД. База данных должна содержать следующую информацию:

- пользователи. Необходимо хранить ID пользователя для отправки сообщений;

- категории и вопросы. База данных должна хранить в себе дерево категорий и вопросы, которые относятся к той или иной категории;

- образовательные программы. Требуется хранить основные сведения (название, ссылка, уровень образования, форма и срок обучения, описание и информация о карьере);

- экзамены и предметы. Необходимо хранить список вступительных испытаний для каждой программы с указанием типа конкурсной программы, предметов и того, являются ли они обязательными.

- синонимы для предметов вступительных испытаний. Для реализации сервиса исправления опечаток в предметах, требуется составить словарь синонимов для каждого предмета;

- проходные и средние баллы. База данных должна хранить данные конкурсной статистики, учитывая основания поступления (бюджет, платное и т.д.);

- важные даты процесса поступления. Требуется хранить ключевые точки для возможности уведомлять пользователя.

1. Аргументация технических решений
   1. Языки программирования

В этом разделе сравним несколько популярных языков программирования, которые лучше всего подходят для разработки Telegram-бота.

* + 1. Python (<https://www.python.org/>)

По данным TIOBE Index, популярность Python достигла 25,35% на состояние мая 2025 года, что является самой высокой долей за всю историю индекса (<https://www.tiobe.com/tiobe-index/>). Согласно отчету GitHub Octoverse 2024, Python обогнал JavaScript и стал самым популярным языком на платформе (https://github.blog/news-insights/octoverse/octoverse-2024/). Компании широко применяют в своих продуктах, язык имеет большое комьюнити и поддержку.

Python имеет встроенную поддержку асинхронных вызовов, что отлично подходит для создания производительных Telegram-ботов.

Существует ряд фреймворков специально для разработки Telegram-ботов (например, aiogram, pyTelegramBotApi, telethon), которые облегчают взаимодействие с Telegram Bot Api. Официальный репозиторий Python Package Index (PyPI) содержит более 614 000 пакетов, охватывающих широкий спектр задач, таких как автоматизация, анализ данных, базы данных (<https://pypistats.org/>).

Благодаря библиотекам машинного обучения, такими как TensorFlow, PyTorch, scikit-learn, transformers, Python является отличным выбором для машинного обучения.

* + 1. C# (https://dotnet.microsoft.com/en-us/languages/csharp)

По данным TIOBE Index за май 2025 C# занимает пятое место среди самых популярных языков с долей 4,22%. Согласно отчету GitHub Octoverse 2024, C# занимает 5 место по количеству пользователей, вносящих вклад в проекты, на github.com (https://github.blog/news-insights/octoverse/octoverse-2024/). Язык стабильно входит в топ 5 TIOBE, что говорит о его востребованности в индустрии (<https://www.tiobe.com/tiobe-index/>).

Поддерживает асинхронное программирование, что отлично подходит для разработки Telegram-ботов

На официальной платформе NuGet доступно более 100 000 уникальных пакетов (<https://learn.microsoft.com/en-us/nuget/what-is-nuget>). По сравнению с Python экосистема C# менее разнообразна, но все еще предоставляет достаточное количество решений для разработки бота. Для упрощенного взаимодействия с API Telegram существуют разнообразные решение (например, Telegram.Bot, Telegram.BotAPI, TelegramBotFramework)

Так же, как и Python содержит необходимые для машинного обучения библиотеки (например, transformers). Это позволяет легко интегрировать интеллектуальные системы в приложения на C#.

* + 1. Java (https://www.java.com/ru/)

По данным TIOBE Index за май 2025 года, Java занимает четвёртое место среди самых популярных языков программирования с долей 9,31%. Согласно данным GitHub Octoverse 2024, Java занимает 4 место по количеству пользователей, вносящих вклад в проекты, на github.com (https://github.blog/news-insights/octoverse/octoverse-2024/). Так же как C#, стабильно входит в топ 5 языков программирования согласно этому индексу (<https://www.tiobe.com/tiobe-index/>).

Поддерживает асинхронные механизмы вызова функций (например, CompletableFuture), что делает язык подходящим для использования в Telegram-боте. Хотя это не так удобно, как async/await вызовы, как в Python или C#.

Для создания Telegram-бота существуют разнообразные решения для упрощения создания приложения, например, TelegramBots, java-telegram-bot-api, TDLight Java.

Java не имеет официальной библиотеки от Hugging Face, но существуют сторонние решения позволяющие интегрировать модели в Java-приложение.

* + 1. Обоснование выбора Python

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **Python** | **C#** | **Java** |
| Популярность (TIOBE, май 2025) | 1 место (25,35%) | 5 место (4,22%) | 4 место (9,31%) |
| Популярность (GitHub Octoverse) | 1 место | 5 место | 4 место |
| Асинхронность | Полная поддержка async/await | Полная поддержка async/await | Поддержка через CompletableFuture и аналоги |
| Библиотеки для Telegram-ботов | aiogram, python-telegram-bot, pyTelegramBotApi | Telegram.Bot, Telegram.BotAPI и др. | TelegramBots, java-telegram-bot-api и др. |
| Экосистема пакетов | Более 614,000 пакетов на PyPI | Более 100,000 пакетов на NuGet | Огромное количество, более 16,5 млн. пакетов на maven central repository |
| Машинное обучение | TensorFlow, PyTorch, scikit-learn, transformers | Transformers (портирован), ML.NET | Ограниченная поддержка, сторонние адаптации |

Исходя из преимуществ и недостатков рассмотренных языков был сделан вывод, что python является наилучшим выбором для разработки Telegram-бота благодаря своей простоте, встроенной поддержке асинхронности, огромному количеству специализированных библиотек, а также активному сообществу и экосистеме. Он позволяет быстро и просто создавать интеллектуальные решения на основе искусственного интеллекта и машинного обучения. В отличии от Java и C# имеет более удобный синтаксис и более адаптирован под быструю разработку.

* 1. Фреймворки

В этом разделе будут рассматриваться популярные фреймворки для создания Telegram-ботов.

* + 1. Python-telegram-bot (https://github.com/python-telegram-bot/python-telegram-bot)

Одна из самых популярных и поддерживаемых библиотек для работы с Telegram Bot Api (имеет 27,6 тысяч звезд на состояние 24 мая 2025 года на gihub.com).

Имеет следующие преимущества:

- поддерживает асинхронный режим работы, используя asyncio в свое реализации;

- поддерживает FS;

- поддерживает webhook и polling;

- регулярно обновляется;

- широко используется в образовательных курса, open source и коммерческих проектах.

Пожалуй, имеет один недостаток – отсутствие middleware.

* + 1. Aiogram (https://github.com/aiogram/aiogram)

Один из первых фреймворков на Python, который изначально был асинхронным. Все взаимодействия с Telegram API, обработчики, middleware и фильтры реализованы как корутины, что делает aiogram особенно эффективным при высокой нагрузке.

Имеет следующие преимущества:

- поддерживает FSM;

- поддерживает webhook и polling;

- регулярно обновляется;

- интеграция с aiogram\_dialog. Это расширение позволяет создавать сложные формы и диалоги с управление состояния. Предоставляет готовые компоненты для создания красивых и удобных интерфейсов.

Сообщество меньше, чем у python-telegram-bot (5,1 тысяч звезд по состоянию на 24 мая 2025 мая на github.com).

* + 1. Pyrogram (https://github.com/pyrogram/pyrogram)

Этот фреймворк ориентирован на работу с пользовательскими аккаунтами через Telegram MTProto API (<https://docs.pyrogram.org/topics/mtproto-vs-botapi>). Однако, он поддерживает взаимодействие и через Bot API, что делает его более гибким для сложных Telegram-приложений. Не смотря на свою популярность (4,5 тысячи звезд на состояние 24 мая 2025 года на github.com), репозиторий не получал значительных обновлений уже 2 года (репозиторий заархивирован владельцем с 24 декабря 2024 года, и теперь только для чтения).

Имеет следующие преимущества:

- имеет асинхронный режим работы;

- поддерживает webhook и polling.

Недостатки:

- прекращененно развитие фреймворка, решением может быть перейти на один из форков;

- не имеет встроенной поддержки middleware;

- избыточен для обычного информационного бота.

* + 1. Обоснование выбора aiogram

Сводная таблица, обобщающая достоинства и недостатки рассмотренных фреймворков:

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **python-telegram-bot** | **aiogram** | **pyrogram** |
| Количество звёзд на GitHub | 27,6 тыс. | 5,1 тыс. | 4,5 тыс. (архивирован) |
| Асинхронность | Да | Да | Да |
| Поддержка FSM | Да | Да | Нет |
| Поддержка webhook / polling | Да / Да | Да / Да | Да / Да |
| Middleware | Нет | Да | Нет |
| Интеграция с фреймворком компонент aiogram\_dialog | Нет | Да | Нет |
| Обновления | Регулярные | Регулярные | Прекращены |
| Популярность / сообщество | Очень большое | Среднее | Было активное, сейчас – стагнация |
| Поддержка MTProto | Нет | Нет | Да |
| Подходит для обычных ботов | Да | Да | Да |

Aiogram предлагает встроенную поддержку FSM, middleware, webhook и polling, а также интеграцию с aiogram\_dialog для создания продвинутых интерфейсов и сценариев. Он регулярно обновляется и активно развивается, что делает его надёжным выбором для современных Telegram-ботов, где важна производительность и расширяемость. Это обуславливает выбор в сторону этого фреймворка.

* 1. СУБД

В этом разделе будут рассмотрены несколько популярных СУБД.

* + 1. PostgreSQL (https://github.com/postgres/postgres)

PostgreSQL – объектно-реляционная СУБД с открытым исходным кодом, активно развиваемая с 1996 года. По данным DB-Engines Ranking за май 2025 года, PostgreSQL занимает 4-е место среди всех СУБД, уступая только Oracle, MySQL и Microsoft SQL Server (<https://db-engines.com/en/ranking>).

49% разработчиков используют PostgreSQL в своих проектах. Это самая популярная СУБД второй год подряд согласно stackoverflow developer survey по данным 2024 года (<https://survey.stackoverflow.co/2024/>).

Имеет более 17,6 тысяч звезд на github.com по состоянию на 24 мая 2025 года (<https://github.com/postgres/postgres>).

Показывает высокую производительность на практике.

Бенчмарки или ссылки на как-то???

* + 1. SQLite (https://github.com/sqlite/sqlite)

SQLite — встроенная реляционная СУБД, не требующая установки сервера и функционирующая через простой файл базы данных. Это делает её особенно удобной для мобильных приложений, десктопных программ и встраиваемых систем. По данным DB-Engines Ranking за май 2025 года, SQLite занимает 10-е место (https://db-engines.com/en/ranking). Имеет более 7,8 тысяч звезд на github.com по состоянию на 24 мая 2025 года (<https://github.com/sqlite/sqlite>).

Особенна популярна в мобильной разработке.

Хотя SQLite показывает высокую производительность и простоту внедрения, она не предназначена для высоконагруженных распределённых систем и имеет ограниченную поддержку параллелизма и масштабирования.

SQLite использует механизм блокировки на уровне файла, что означает, что в любой момент времени может выполняться только одна операция записи. Это ограничение может стать узким местом в приложениях с высокой частотой записей.

Показывает высокую производительность.

Бенчмарки или ссылки на как-то???

* + 1. MySQL (https://github.com/mysql/mysql-server)

MySQL – одна из самых популярных реляционных СУБД с открытым исходным кодом, разработанная в 1995 году и в настоящее время поддерживаемая Oracle Corporation. По данным DB-Engines Ranking за май 2025 года, MySQL занимает второе место среди всех СУБД, уступая только Oracle (https://db-engines.com/en/ranking).

Показывает высокую производительность, хотя уступает PostgreSQL.

* + 1. Обоснование выбора PostgreSQL

|  |  |  |  |
| --- | --- | --- | --- |
| **Критерий** | **PostgreSQL** | **SQLite** | **MySQL** |
| Тип | Объектно-реляционная СУБД | Встроенная реляционная СУБД | Реляционная СУБД |
| Рейтинг DB-Engines | 4 место | 10 место | 2 место |
| Звёзды на GitHub | 17,6 тыс. | 7,8 тыс. | 11,4 тыс. |
| Популярность среди разработчиков | 49% (StackOverflow Survey 2024) | Очень популярна в мобильной разработке | Широко используется |
| Производительность | Высокая | Высокая при чтении, особенно на малом объёме данных | Высокая, но как правило чуть меньше, чем у PostgreSQL |
| Параллелизм | Поддерживается | Ограниченный (блокировка на уровне файла) | Поддерживается |

PostgreSQL был выбрана в качестве основной СУБД благодаря своей надёжности, активному сообществу и широким возможностям. Эта СУБД предлагает полноценный параллелизм и гибкую настройку масштабируемости.

* 1. Система семантического поиска по текстовым данным

Далее потребуется несколько понятий.

**Персистентность** – это способность системы сохранять данные между сеансами работы. В контексте баз данных и кэш-хранилищ это означает, что данные не теряются при перезапуске, сбоях питания или сбоях приложения.

**Репликация** – это процесс копирования данных с одного узла (мастера) на другие узлы. Это используется для повышения отказоустойчивости и распределённого доступа к данным.

**Кластеризация** – это объединение нескольких серверов (узлов) в одну логическую систему – кластер. В кластере задачи и данные распределяются между узлами для повышения производительности, отказоустойчивости и масштабируемости.

**Семантическая близость** – это мера, показывающая, насколько два объекта (например, слова, фразы или тексты) близки по смыслу.

* + 1. Необходимость векторного поиска

В разрабатываемом проекте требуется сравнивать описание образовательных программ вуза с произвольным пользовательским запросом на естественном языке. Стандартные реляционные СУБД не приспособлены для семантического поиска, поскольку они оперируют точными совпадениями и фильтрацией по жёстким условиям. Для решения задачи сопоставления по смыслу лучше всего использовать векторное представление текста, где каждый текст кодируется в виде числового вектора с помощью языковой модели. Сравнение векторов позволяет измерять семантическую близость между фразами даже при отсутствии общих слов.

Таким образом задача требует системы, оптимизированной для поиска по многомерным векторам с использованием метрик близости. Векторные базы данных созданы для этого.

(Pan J. J., Wang J., Li G. Survey of vector database management systems //The VLDB Journal. – 2024. – Т. 33. – №. 5. – С. 1591-1615.)

* + 1. Обзор векторных баз данных

Были рассмотрены следующие решения:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Лицензия** | **Фильтрация по метаданным** | **Особенности** | **Звезды GitHub (24.05.2025)** |
| Qdrant[[1]](#footnote-1) | Apache 2.0 | Полноценная поддержка | gRPC/REST, Fast embeddings search, кластеризация | 23,7 т. звезд |
| Pinecone[[2]](#footnote-2) | Коммерческая | Частично (через API) | SaaS[[3]](#footnote-3), легко масштабируется, без локального запуска | – |
| Weaviate[[4]](#footnote-4) | Apache 2.0 | Через GraphQL | модули для трансформерных моделей, встроенные модели | 13,4 т. звезд |
| FAISS[[5]](#footnote-5) | MIT | Нет встроенной поддержки | Библиотека от Meta[[6]](#footnote-6), хороша для оффлайн-поиска, но не полноценная БД | 35,1 т. звезд |
| Milvus[[7]](#footnote-7) | Apache 2.0 | Через JSON-фильтры | Поддержка кластеров, | 34,9 т. звезд |

Стоит отметить, что все рассмотренные варианты имеют официальную поддержку Python и все, кроме FAISS, входят в десятку лучших векторных баз данных по версии рейтинга DB-Engines Ranking на 24 мая 2025 года (https://db-engines.com/en/ranking/vector+dbms).

* + 1. Обоснование выбора Qdrant

Для решения задачи был выбран именно Qdrant по следующим причинам:

- полноценная поддержка фильтрации по метаданным, таким как вступительные экзамены, форма обучения, ID программы, и т.д., что удобно для суживания области поиска;

- готовый Docker образ;

- активное сообщество и документация;

- высокая производительность;

- открытая лицензия Apache 2.0.

Таким образом Qdrant – оптимальный выбор для реализации системы интеллектуального поиска с точки зрения удобства, гибкости и производительности.

* 1. Системы хранения кэшируемых данных

Системы хранения кэшируемых данных позволяют значительно ускорить работу приложений, снижая нагрузку на основную базу данных и сервер, уменьшая время отклика и повышая масштабируемость при большом числе запросов.

Далее понадобится термин шардирование.

**Шардирование** – это способ распределения данных по нескольким серверам, при котором каждая часть (шард) содержит только часть общей информации, что позволяет улучшить масштабируемость и производительность системы.

* + 1. Redis (https://github.com/redis/redis)

Это высокопроизводительное хранилище данных в оперативной памяти, которое поддерживает различные структуры, такие как строки, списки, множества и хэши. Благодаря этому Redis подходит для широкого спектра задач: кэширования, хранения пользовательских сессий, реализации очередей задач и обмена сообщениями между частями системы. Имеет 69,4 т. звезд на github.com и занимает 1 место среди key-value хранилищ согласно DB-Engines (24 мая 2025 года).

Redis реализует механизм Pub/Sub (от англ. Publish/Subscribe), позволяющий одной части системы публиковать сообщения, а другим – подписываться на их получение. Это удобно, например, для уведомлений или обновлений в реальном времени.

Кроме того, Redis поддерживает TTL (Time-To-Live) – срок жизни ключа. Это означает, что данные автоматически удаляются через заданный интервал времени, что особенно полезно при кэшировании и управлении временными объектами.

Возможности кластеризации и репликации обеспечивают высокую доступность и масштабируемость Redis, позволяя использовать его в отказоустойчивых и распределённых системах.

* + 1. Memcached (https://github.com/memcached/memcached)

**Э**то простое, высокопроизводительное кэширующее хранилище, использующее память для хранения пар «ключ-значение». Оно особенно эффективно для ускорения веб-приложений, за счёт хранения часто запрашиваемых данных в оперативной памяти. Имеет 13,8 т. звезд на github.com и занимает 4 место среди key-value хранилищ согласно DB-Engines (24 мая 2025 года).

В отличие от Redis, **Memcached не поддерживает сложные структуры данных**, а также отсутствуют такие функции, как Pub/Sub или персистентность данных.

Тем не менее, за счёт своей простоты и скорости, Memcached часто используется для **кэширования** в веб-приложениях и API, где требуется минимальная задержка и нет необходимости в сложных операциях с данными.

* + 1. Dragonfly (https://github.com/dragonflydb/dragonfly)

Это высокопроизводительное хранилище в оперативной памяти, разработанное как многопоточная альтернатива Redis с архитектурой «shared-nothing», что обеспечивает лучшее масштабирование и использование многопроцессорных систем. Полностью совместим с API Redis и Memcached, поддерживает основные структуры данных, Pub/Sub и TTL. По производительности Dragonfly в разы быстрее (на странице репозитория говорится, что в 25 раз), чем Redis. К тому же требует меньше ресурсов (на странице репозитория говорится, что на 80% процентов меньше). Это делает Dragonfly привлекательным выбором для приложений с высокими требованиями к скорости и масштабируемости. Имеет 27 т. звезд на github.com и занимает 35 место среди key-value хранилищ согласно DB-Engines (24 мая 2025 года).

* + 1. Обоснование выбора Redis

|  |  |  |  |
| --- | --- | --- | --- |
| **Характеристика** | **Redis** | **Memcached** | **Dragonfly** |
| GitHub звёзды | 69,3 тыс. | 13,8 тыс. | 27 тыс. |
| Место в DB-Engines | 1 | 4 | 35 |
| Поддержка структур | Строки, списки, множества, хэши и др. | Только строки (ключ-значение) | Строки, списки, множества, хэши и др. |
| TTL (время жизни ключа) | Да | Д | Да |
| Pub/Sub | Да | Нет | Да |
| Персистентность | Да (RDB, AOF) | Нет | Да (улучшена) |
| Масштабируемость | Кластеры, репликация | Простая горизонтальная (шардирование вручную) | Многопоточность, shared-nothing |
| Совместимость | – | – | Совместим с Redis и Memcached |
| Зрелость проекта | С 2009 года | С 2003 года | С 2022 года |
| Поддержка Python | Официальная библиотека redis-py | Официальная библиотека pymemcache | Через совместимость с Redis |

Redis был выбран благодаря своей зрелости, широким возможностям (включая поддержку различных структур данных, TTL, Pub/Sub и персистентности), высокой производительности и устойчивости. В отличие от более простого Memcached и относительно нового Dragonfly, Redis предлагает проверенный баланс между надёжностью, масштабируемостью и гибкой функциональностью, необходимой для кэширования и обмена данными в современных приложениях.

* 1. Среда исполнения и развертывание приложения
     1. Проблемы локального развертывания

При локальном развертывании приложения, которое будет включать в себя сервисы: PostgreSQL, PostgreSQL, Qdrant и собственные сервисы – могут следующие сложности:

- необходимость вручную устанавливать и настраивать каждую зависимость;

- риск несовместимости версий и различий в окружениях;

- трудности с воспроизведением ошибок;

- сложность с Qdrant, который требуется собирать из исходного кода под ту или иную операционную систему;

- увеличение времени на начальную настройку и запуск.

Эти проблемы могут существенно усложнить разработку и поддержку Telegram-бота.

* + 1. Docker в качестве решения проблемы (https://www.docker.com/)

Для решения указанных проблем было принято решение использовать Docker, который успел стать стандартом индустрии для контейнеризации приложений.

Преимущества:

- изоляция компонентов и воспроизводимость окружения;

- быстрая настройка и запуск;

- интеграция с GitHub Actions;

- масштабируемость и гибкость;

- зрелость экосистемы и широкий выбор готовых образов в Docker Hub.

Недостатки:

- усложнение отладки в контейнеризированной среде;

- небольшие накладные расходы на производительность.

* 1. Итог техническим решениям

В результате сравнительного анализа и практических требований к проекту Telegram-бота были приняты следующие ключевые технические решения:

- язык программирования – Python. Благодаря огромному сообществу (25,35 % по TIOBE, май 2025), удобству, встроенной поддержке асинхронности, а также обширному набору специализированных библиотек и фреймворков. Python обеспечивает высокую скорость разработки, простоту сопровождения и лёгкую интеграцию решений на основе машинного обучения;

- фреймворк – aiogram. Выбран за асинхронность, встроенную поддержку FSM и middleware, регулярные обновления и расширение через aiogram\_dialog. Он позволяет строить сложные сценарии диалога с управлением состояния;

- основаная СУБД для хранения данных – **PostgreSQL.** Объектно-реляционная СУБД с активным сообществом, продвинутыми возможностями параллелизма и расширений, стабильной производительностью и высоким рейтингом (4-е место DB-Engines, 49% использования по опросу StackOverflow 2024). PostgreSQL обеспечивает надёжное хранение структурированных данных и лёгкую интеграцию с ORM;

- векторная база данных – Qdrant. Выбрана за готовый Docker-образ, поддержку фильтрации по метаданным, высокую производительность и открытую лицензию Apache 2.0. Qdrant позволяет эффективно решать задачу семантического поиска по векторным представлениям текстов. Кроме того, она хорошо интегрируется с Python;

- кэш – **Redis.** Опытный и зрелый проект (с 2009 г.), лидирует среди key-value хранилищ (1-е место DB-Engines), предлагает богатый набор структур данных, встроенный Pub/Sub, TTL и механизмы персистентности. Redis обеспечивает достаточно быстрое кэширование и высокую отказоустойчивость;

- контейнеризация с помощью Docker. Docker гарантирует единообразие окружения, изоляцию зависимостей и ускоряет развертывание всех сервисов (бот, PostgreSQL, Redis, Qdrant, сервис поиска по базе данных, тестовая база данных, CI контейнер). Интеграция с GitHub Actions позволяет автоматически проверять корректность сборки и выполнения тестов при каждой отправке в репозиторий.

В совокупности, выбранные решения обеспечивают оптимальный баланс между быстротой разработки, надёжностью, производительностью и возможностями масштабирования проекта в долгосрочной перспективе.

* 1. Выбор лингвистической модели векторизации

1. Разработка чат-бота в поддержку абитуриента ВятГУ
   1. Архитектура системы
      1. Описание компонентов

Архитектура разработана с учётом требований надёжности, масштабируемости и гибкости. Проект реализован в виде набора изолированных микросервисов, каждый из которых отвечает за отдельный участок функциональности, что потенциально должно облегчить сопровождение и развитие системы в будущем.

Далее компоненты, из которых состоит система.

Бот-сервис – основной модуль, реализующий взаимодействие с пользователями. Он отвечает за обработку сообщений, запуск диалогов, отображение интерфейсов, кроме того, именно он отвечает за базу данных (в нем находятся миграции), так как все остальные сервисы направлены на взаимодействие с ним.

Redis. Используется в качестве промежуточного слоя. Предназначен для создания trottling middleware, которые ограничивают спам атаки от пользователя, и для кеширования часто используемых данных с целью повышения скорости и снижения нагрузки на основную СУБД.

Сервис векторного поиска. Отдельный микросервис, реализующий семантический поиск по базе часто задаваемых вопросов. Он использует Qdrant в связке с моделью sentence-transformers для хранения и поиска. Выделение этой функциональности в отдельный сервис позволяет разгрузить основной бот и повысить устойчивость всей системы за счёт изоляции ресурсоёмких операций. Кроме того, такая архитектура упрощает масштабирование и предоставляет возможности для последующего расширения. На данном этапе с ним взаимодействует только бот сервис через FastAPI.

PostgreSQL. Сюда загружается вся нужная информация об образовательных программах. Структура управляется с помощью alembic из основного сервиса бота. Доступ к данным осуществляется через ORM SQLAlchemy

Qdrant. Векторное хранилище данных. Обслуживает сервис векторного поиска. Позволяет выполнить быстрый поиск по embedding-представлениям вопросов.

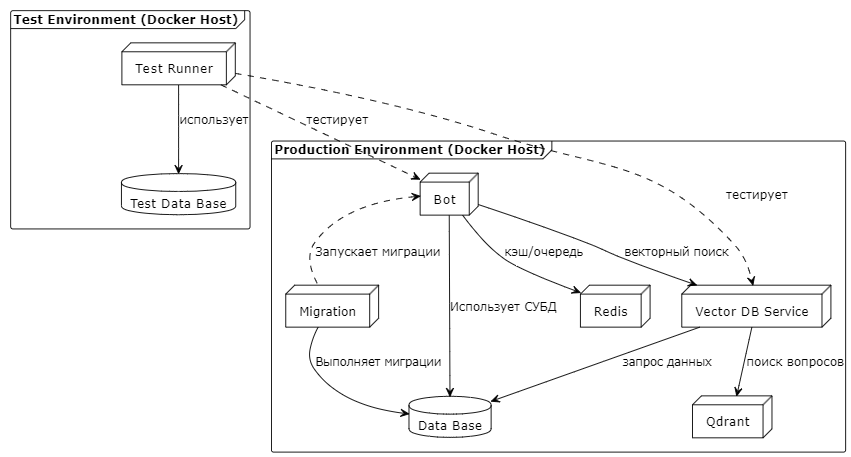
Контейнер запуска тестов. Используется для автоматического запуска тестов. Включает в себя копии исходных файлов и связан с отдельной тестовой базой данных, обеспечивая изоляцию рабочей среды.

Тестовая PostgreSQL. Используется исключительно для тестирования, для изоляции рабочей среды.

Контейнер миграции базы данных. Специальный сервис только для выполнения миграций базы данных. Используется для автоматизации развертывания и принципа разделения ответственности.

* + 1. Процесс развертывания

Сначала рассмотрим рисунок, наглядно демонстрирующий взаимодействие компонентов между собой.



Подробнее разберем процесс развертывания, изображенный на рисунке выше:

- вначале разворачиваются независимые контейнеры: Data Base, Qdarant, Redis, Test Data Base;

- после развертывании контейнер Migration запускает миграционные скрипты и выполняет их над Data Base;

- одновременно с этим Test Runner запускает тесты;

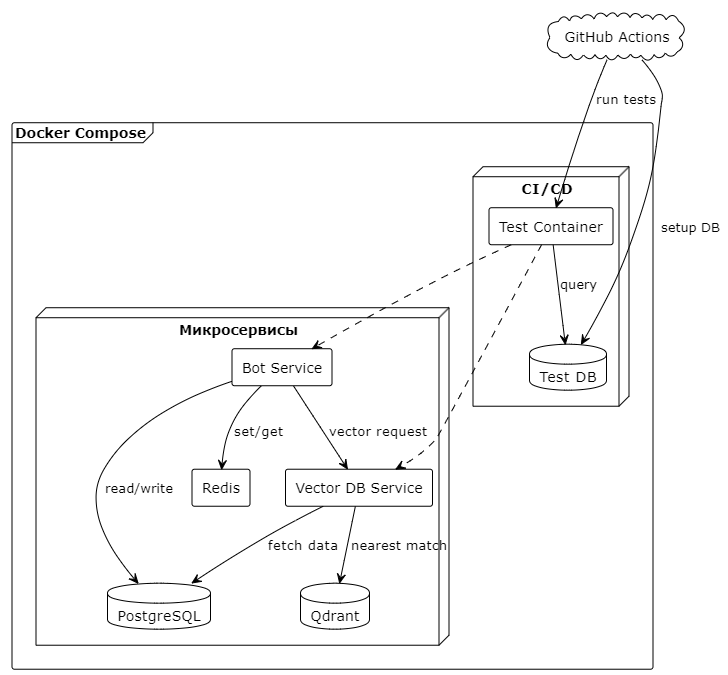
- запускается Vector DB Service и сразу же обращается к базе данных для обновления Qdaran;

- в конце запускается Bot, который так или иначе используется все компоненты в Prodaсtion окружении.

Такой поток взаимодействия обеспечивает четкое разграничение ответственности и надежное функционирование всей системы. Кроме того, процесс автоматический.

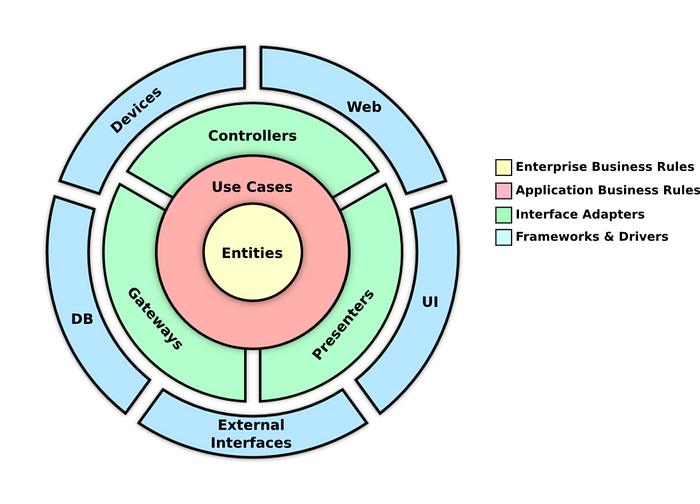
* + 1. Инфраструктура CI/CD

Для автоматического тестирования было принято решение использовать GitHub Actions. Он запускает Test Container и Test DB при слиянии изменений в ветку main. Ниже рисунок, который показывает это.



* + 1. Архитектура бота

Для реализации целевого сервиса-бота была выбрана чистая архитектура. В основе этой архитектуры лежит правило зависимостей, которое гласит, что внутренние слои приложения не должны зависеть от реализаций внешних слоев, а должны определять интерфейсы-контракты, которые будут реализовывать внешние слои. Рассмотрим рисунок (номер рисунка).



(<https://medium.com/swlh/clean-architecture-a-little-introduction-be3eac94c5d1>)

В этой архитектуре выделяют 4 слоя:

- доменный слой (на рисунке Enterpice Busisness Rules). Он содержит бизнес-объекты и логику, которая не зависит от внешних систем и должна меняться;

- слой приложения (на рисунке Application Business Rules). В нем описываются бизнес-правила и сценарии использования. Он использует интерфейсы, а не конкретные реализации;

- слой представления (на рисунке Interface Adapters). Отвечает за взаимодействие с пользователем через интерфейс Telegram. В нем расположены интерфейсы репозиториев для взаимодействия с базой данных и интерфейсы других сервисов. Тут происходит внедрение зависимостей;

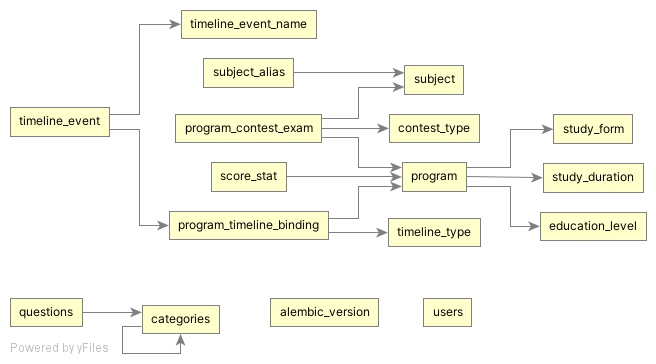
- инфраструктурный слой (на рисунке Frameworks & Drivers). Здесь расположены конкретные реализации для связи с внешними сервисами.

Благодаря строгому разделению ответственности и правилу зависимостей, бизнес-логика остаётся изолированной от внешних технологий, что позволяет легко заменять инфраструктурные компоненты, адаптироваться к новым требованиям и повторно использовать ядро приложения в других интерфейсах или проектах (Martin R. C. Clean architecture [Электронный ресурс].).

* + 1. Структура базы данных

В этом разделе представлена структура база данных, используемая для хранения информации о направлениях и другой важной информации.

Рассмотрим взаимодействие сущностей базы данных на рисунке ниже.



Краткое описание сформированных таблиц:

- users содержит информацию о пользователях. На текущем этапе идентификатор пользователя, например, из Telegram;

- categories представляет иерархию категорий, используется для группировки вопросов. Имеет возможность указывать родительскую категорию;

- questions содержит часто задаваемые вопросы и ответы. Каждый вопрос относится к одной листовой категории;

- program представляет образовательную программу вуза. Включает название, ссылку, описание программы и карьеры, а также внешние ключи на уровень образования, форму и срок обучения;

- education\_level – справочник уровней образования (например, бакалавриат, магистратура);

- study\_form – справочник форм обучения (очная, заочная и т.п.);

- study\_duration – справочник сроков обучения (например, 4 года);

- contest\_type – справочник типов вступительных испытаний. Используется в связке с программами через промежуточную таблицу;

- subject – справочник учебных предметов;

- subject\_alias содержит альтернативные названия предметов. Используется для поиска по различным формулировкам одного и того же предмета;

- program\_contest\_exam – промежуточная таблица, связывающая программу, предмет и тип вступительного испытания. Также указывает, является ли предмет необязательным;

- score\_stat содержит статистику по программе: проходные и средние баллы, количество бюджетных, платных, целевых и квотных мест. Привязана к конкретной программе;

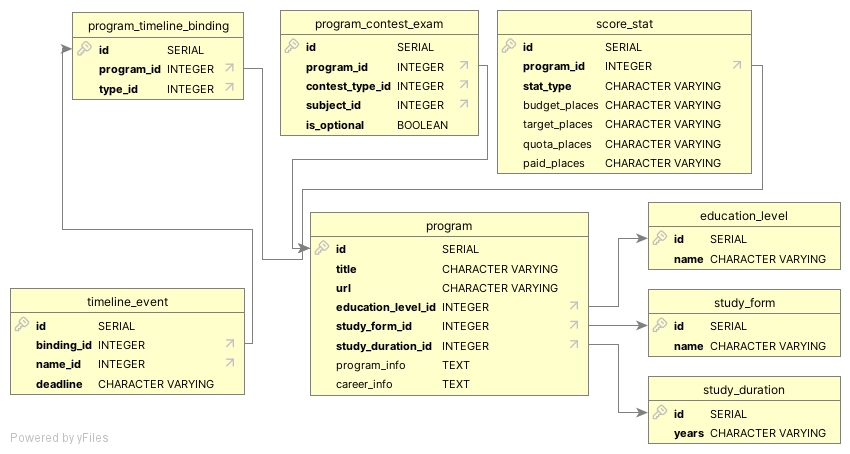
- program\_timeline\_binding связывает программу с типом графика приема. Используется для организации событий в графике поступления;

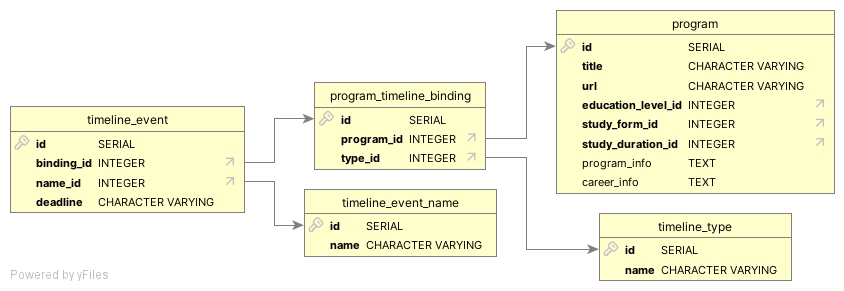
- timeline\_type – справочник типов графиков. Используется для группировки событий и предоставляет возможность добавлять новые типы графиков поступления;

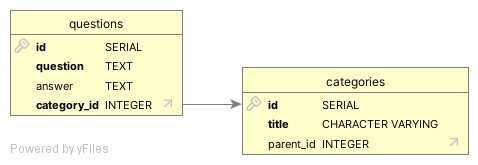
- timeline\_event представляет конкретное событие в графике поступления (например, начало приема документов). Связано с программой через program\_timeline\_binding и с типом события через timeline\_event\_name;

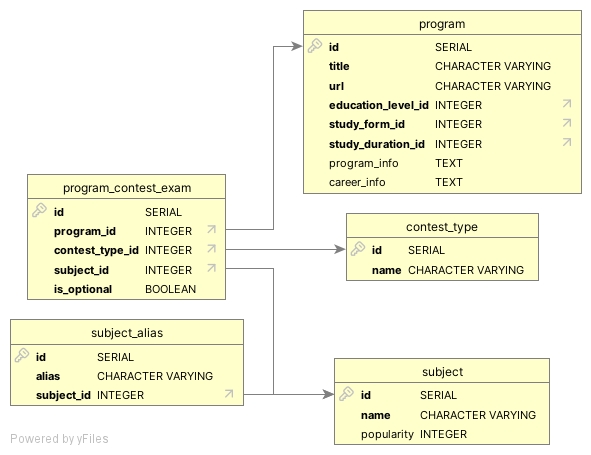
- timeline\_event\_name – cправочник названий событий в графике поступления. Используется для унификации и повторного использования событий в разных графиках.

Ниже в целях повышения детализации и читаемости идут диаграммы, представленные по отдельности для каждой ключевой сущности со связанными с ними таблицами. Они более подробно описывают структуру базы данных.



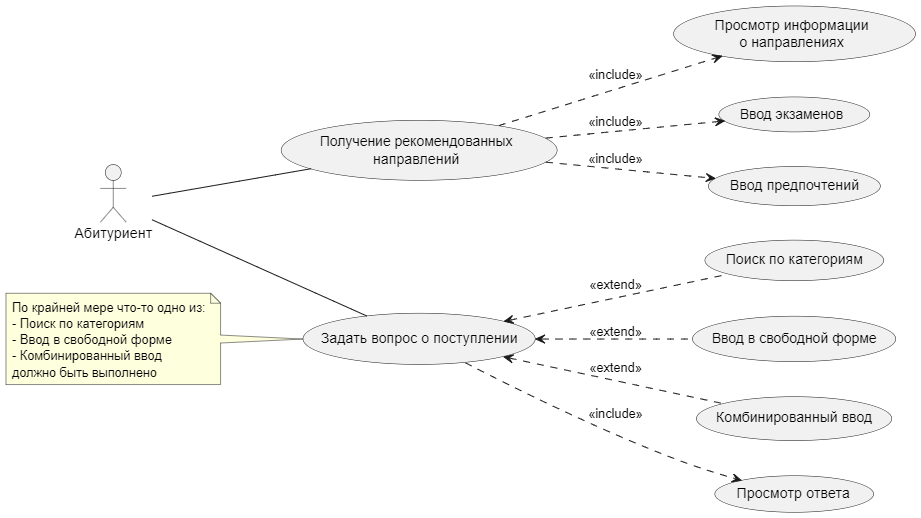






* 1. Сценарии взаимодействия с пользователем

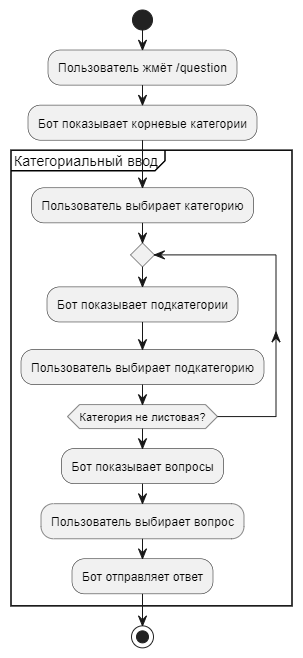
В этом разделе рассмотрим основные взаимодействия с пользователем. Ниже диаграмма использования, которая показывает, какие функции включены в бота.



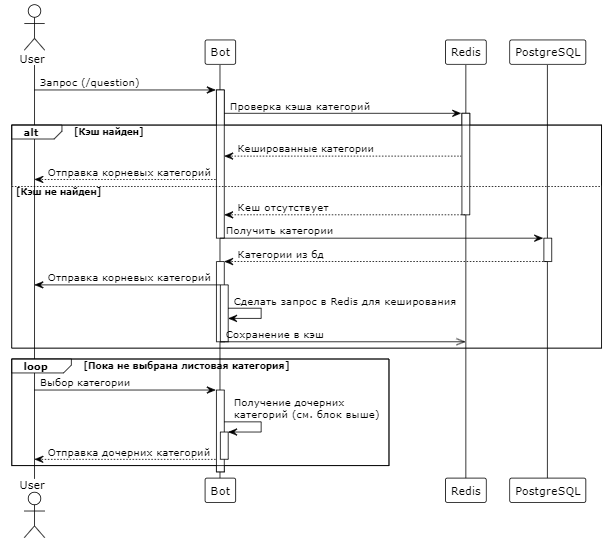
* + 1. Вопрос о поступлении

Пользователь может задать интересующий его вопрос о поступлении двумя основными способами: через навигацию по категориям либо во вводе в свободной форме. Это позволяет учитывать как формализованные, так и неструктурированные запросы.

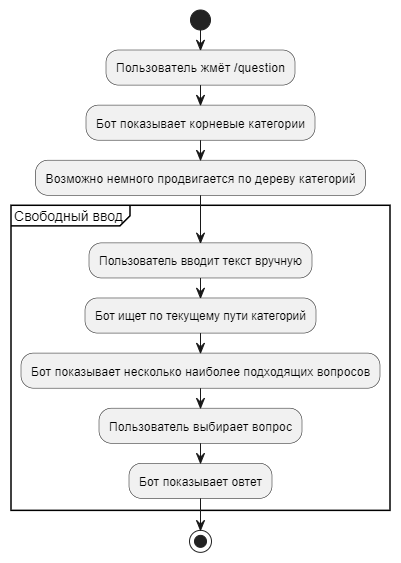
Диаграмма активности на рисунке ниже демонстрирует, как пользователь может взаимодействовать с системой.



Для рассмотрения процесса с точки зрения взаимодействия компонентов системы далее приведена диаграмма последовательности, иллюстрирующая сценарий выбора категорий пользователем. На ней продемонстрировано взаимодействие с хранилищем Redis и PostreSQL.

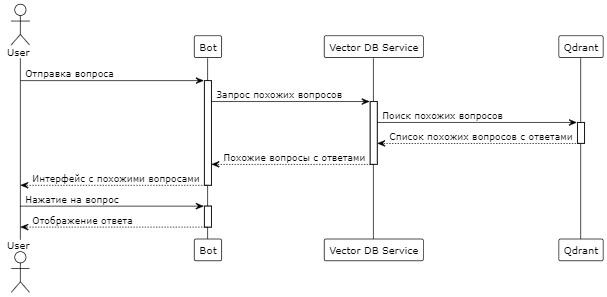


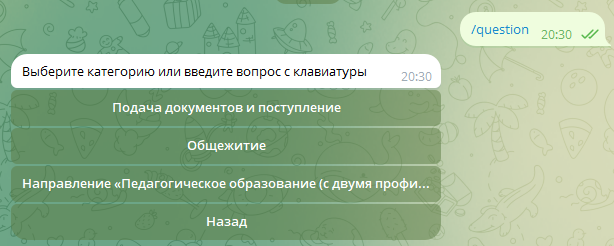
Вариант с категорией предполагает пошаговое движение по иерархии тематических разделов. Но пользователь может не понять, в какую категорию нужно зайти, чтобы увидеть свой вопрос. Для решения этой проблемы в любой момент пользователь имеет возможность задать вопрос с клавиатуры в свободной форме. Ниже на рисунке диаграмма последовательности для этого сценария.



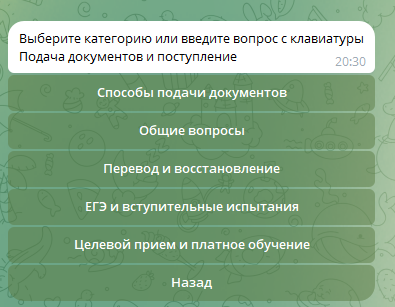
Также пользователь может нажать кнопку назад, чтобы перейти к предыдущему пункту.

Для рассмотрения процесса с точки зрения взаимодействия компонентов системы далее приведена диаграмма последовательности, иллюстрирующая сценарий задания вопроса пользователем в свободной форме. На ней продемонстрировано взаимодействие с хранилищем Vector DB Service и Qdrant.

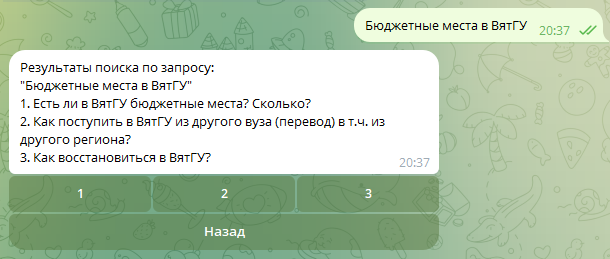


Для лучшего понимания логики взаимодействия с пользователем в мессенджере Telegram ниже представлены скриншоты, иллюстрирующие пошаговую работу сценария «Вопрос о поступлении». 

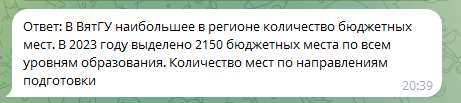
Пользователь нажал на кнопку «Подача документов и поступление». Бот изменил меню, как показано на рисунке ниже



Пользователь ввел запрос с клавиатуры «Бюджетные места в ВятГУ» и отправил сообщение. Бот изменил меню, как показано на рисунке ниже

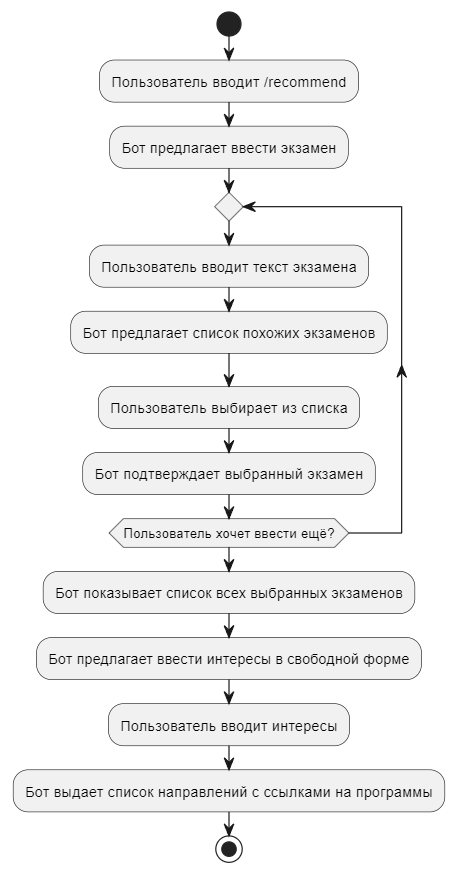


Пользователь нажал на кнопку «1». Бот отправил сообщение, как на рисунке ниже.

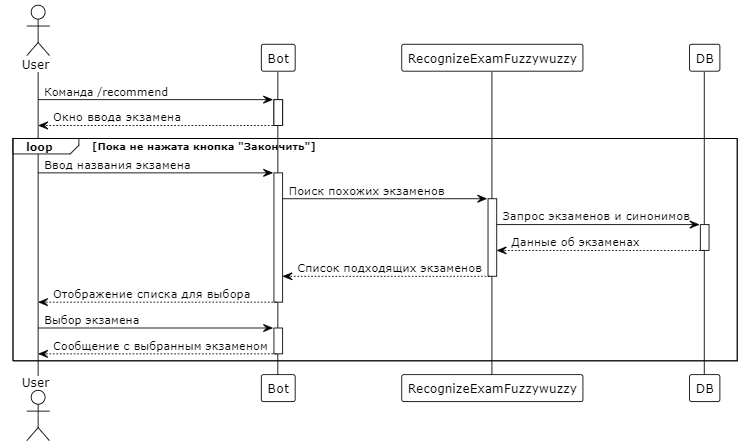


* + 1. Получение рекомендованных направлений

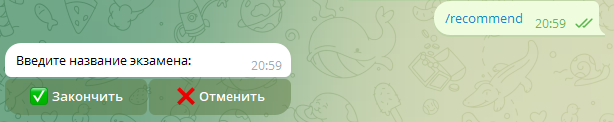
Данный сценарий начинается с команды /recommend, после чего бот запускает пошаговый процесс сбора информации об экзаменах и предпочтениях пользователя. На рисунке ниже диаграмма активности для этого сценария.



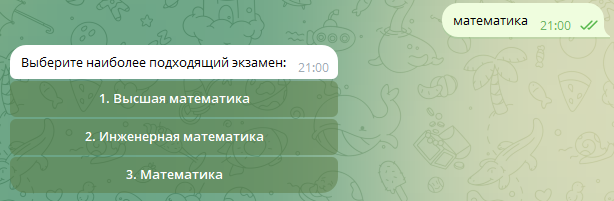
Для рассмотрения процесса с точки зрения взаимодействия компонентов системы далее приведена диаграмма последовательности, иллюстрирующая выборы экзаменов для сценария получения рекомендаций по направлениям. На ней продемонстрировано взаимодействие с сервисом исправления опечаток в экзаменах и базой данных.



Для лучшего понимания логики взаимодействия с пользователем в мессенджере Telegram ниже представлены скриншоты, иллюстрирующие пошаговую работу сценария «Получение рекомендованных направлений».

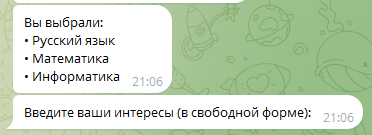


Пользователь ввел на клавиатуре «математика», бот отправил интерфейс, как на рисунке ниже.

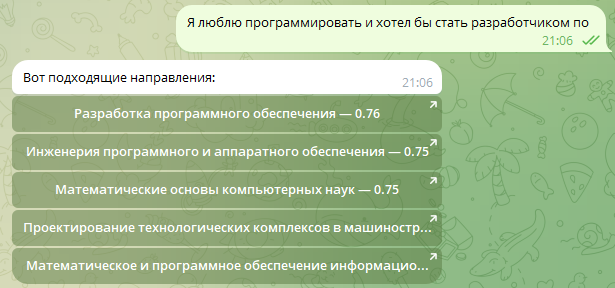


Пользователь нажал на кнопку «3. Математика», бот отправил сообщение, что экзамен добавлен и предложил ввести новый экзамен.

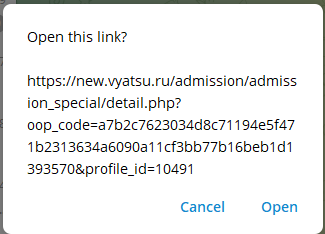
Пользователь ввел подобным образом экзамены: информатика русский язык – и нажал на закончить. Бот отправил экзамены, которые выбрал пользователь и предложение ввести интересы в свободной форме (рисунок ниже).



Пользователь ввел сообщение «Я люблю программировать и хотел бы стать разработчиком по», бот отправил меню, как на рисунке ниже.



Пользователь нажал на первую кнопку. Telegram вывел меню с предложением открыть ссылку на указанную программу (рисунок ниже).



* 1. Тестирование

Разработка приложения велась с соблюдением чистой архитектуры (Martin R. C. Clean architecture [Электронный ресурс].), что позволило легко протестировать компоненты независимо друг от друга.

В рамках проекта были проведены следующие виды тестирования:

- **модульное**. Включало проверку отдельных компонентов (например, сервис исправления опечаток) в изоляции;

- **интеграционное**. Включало проверку взаимодействия между модулями, включая бизнес-логику и базу данных;

- **функциональное. Включало** проверку пользовательских сценариев (задание вопроса, просмотр ответов, подбор направлений).

- **тестирование интерфейса. Включало** проверку корректности отображения сообщений, inline-кнопок и реакций Telegram-бота в различных клиентах;

- **кросс-платформенное. Включало** проверку приложения в мобильных и десктопных версиях Telegram.

В процессе тестирования использовались следующие инструменты:

- pytest для модульных тестов;

- telegram desktop и telegram mobile для ручного тестирования

- фикстуры для изоляции завсимостей;

- docker для воспроизведения окружения.

Кроме того, было настроено автоматическое выполнение тестов с помощью GitHub Actions, что предупреждает разработчика о необходимости исправить код при ошибках в тестовом контейнере.

* 1. Перспективы развития

В текущей версии реализована базовая функциональность, позволяющая абитуриентам взаимодействовать с ботом для получения информации о поступлении и рекомендаций по направлениям. Однако, архитектура приложения спроектирована с учетом масштабирования и дальнейшего расширения. В будущем планируется реализовать:

- механизм подписки на уведомления о важных событиях;

- аналитику и учетность для администраторов системы;

- автоматическое обновление данных раз в определенное время (на данный момент процесс полуавтоматический);

- **web-интерфейс** на базе ядра приложения, реализованного в рамках чистой архитектуры.

Заключение

В рамках данной работы был спроектирован и реализован Telegram-бот, предназначенный для информационной поддержки абитуриентов Вятского государственного университета. Основной целью проекта являлось создание удобного, доступного и функционального инструмента, способного отвечать на часто задаваемые вопросы о поступлении, а также помогать абитуриентам с выбором образовательных программ на основе их экзаменов и интересов.

Архитектура приложения построена в соответствии с принципами чистой архитектуры, что обеспечивает высокую модульность, тестируемость и гибкость решения. Были проведены различные виды тестирования, включая модульное, интеграционное и функциональное, а также настроено автоматическое модульное тестирование с использованием GitHub Actions для повышения надёжности разработки.

Проект полностью контейнеризирован с использованием Docker, что позволяет быстро и удобно развернуть приложение в любом окружении. Использование docker-compose обеспечивает воспроизводимость среды и облегчает настройку всех необходимых для работы сервисов.

Система поддерживает как ручной, так и интеллектуальный ввод вопросов. Она может масштабироваться и дополняться новыми функциями без существенного изменения архитектуры.

Результаты работы подтверждают, что использование современных подходов к проектированию программных систем, таких как чистая архитектура и контейнеризация, способствует созданию устойчивых и развиваемых решений, способных эффективно решать задачи взаимодействия с пользователем в образовательной сфере.

1. https://github.com/qdrant/qdrant [↑](#footnote-ref-1)
2. https://www.pinecone.io/ [↑](#footnote-ref-2)
3. **SaaS** – это модель распространения программного обеспечения, при которой приложения размещаются в облаке и предоставляются пользователю как сервис по подписке. [↑](#footnote-ref-3)
4. https://github.com/weaviate/weaviate [↑](#footnote-ref-4)
5. https://github.com/facebookresearch/faiss [↑](#footnote-ref-5)
6. Признана экстремистской и запрещена на территории РФ [↑](#footnote-ref-6)
7. https://github.com/milvus-io/milvus [↑](#footnote-ref-7)