МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Вятский государственный университет»
(ВятГУ)

Институт математики и информационных систем
Факультет компьютерных и физико-математических наук
Кафедра прикладной математики и информатики

РАЗРАБОТКА ЧАТ-БОТА В ПОДДЕРЖКУ АБИТУРИЕНТА ВЯТГУ

Бакалаврская работа

Киров
2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Вятский государственный университет»
(ВятГУ)
Институт математики и информационных систем
Факультет компьютерных и физико-математических наук
Кафедра прикладной математики и информатики

РАЗРАБОТКА ЧАТ-БОТА В ПОДДЕРЖКУ АБИТУРИЕНТА ВЯТГУ

Бакалаврская работа
144592.020302.01

Выполнил обучающийся гр. ФИб-4301-51-00 _____ / Д. О. Ощепков / _____

Руководитель ВКР к.п.н., зав. кафедрой ПМИ _____ / Е. В. Разова / _____

(подпись) (инициалы, фамилия) (дата)

Киров
2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Кафедра прикладной математики и информатики

УТВЕРЖДАЮ

Зав. кафедрой

_____

«___» _____ 2024 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Студенту Ощепкову Дмитрию Олеговичу

Тема: Разработка чат-бота в поддержку абитуриента ВятГУ

(Утверждена приказом по университету 22.11.2024 г. № 13-02/161)

1. Перечень подлежащих разработке вопросов

1.1. Обзор и анализ научно-технической информации и предметной области: правила приема в ВятГУ, обзор видов чат-ботов, обзор методов и средств построения чат-ботов.

1.2. Обоснование актуальности темы выпускной квалификационной работы. Выбор предметной области. Постановка задачи.

1.3. Проектирование чат-бота в поддержку абитуриента ВятГУ: выбор архитектуры приложения, создание UML-моделей предметной области, сбор данных.

1.4. Разработка чат-бота в поддержку абитуриента ВятГУ: выбор средств разработки ПО, разработка алгоритмов, тестирование ПО.

1.5. Экспериментальное исследование реализованного чат-бота: подготовка наборов данных для исследования, проведение экспериментов, анализ результатов, формирование выводов.

2. Демонстрационный материал

Постановка задачи; UML-диаграммы, схемы основных алгоритмов; основные экранные формы; полученные результаты.

3. Руководитель и консультант по ВКР (с указанием фамилии, имени, отчества, места работы и должности)

3.1. Научный руководитель: Разова Елена Владимировна, кандидат пед. наук, зав. кафедрой ПМИ.

4. Дата выдачи задания 25.11.2024 г.

Руководитель Е.В. Разова

Студент Д.О. Ощепков

Реферат

Ощепков Д. О. Разработка чат-бота в поддержку абитуриента ВятГУ. 144592.020302.01 ВКР / ВятГУ, каф. ПМИ; рук. Е. В. Разова. – Киров, 2025. ПЗ 81 с., 48 рис., 7 табл., 55 источников, 3 прил.

TELEGRAM-БОТ, ВЕКТОРНЫЕ БАЗЫ ДАННЫХ, СЕМАНТИЧЕСКИЙ ПОИСК, PYTHON, AIOGRAM, DOCKER, REDIS, QRDANT

Объект исследования и разработки – программные средства справочной и рекомендательной поддержки абитуриентов ВятГУ в формате Telegram-бота.

Цель выпускной квалификационной работы – разработка Telegram-бота для предоставления информации и рекомендаций поступающим на основе данных с официальных источников ВятГУ.

В ходе работы проведён анализ структуры данных сайта ВятГУ и разработана оптимальная модель их представления. Бот реализован на Python с использованием Telegram API. Реализованы функции просмотра вопросов по базе знаний, подписки на уведомления о событиях и рекомендаций по образовательным программам с учётом интересов пользователя. Используются реляционная и векторная базы данных для поиска, Redis – для кеширования и задач уведомлений.

Результатом является Telegram-бот, предоставляющий абитуриентам структурированный доступ к информации и уведомлениям, повышающий эффективность информационной поддержки и автоматизирующий коммуникацию.

Содержание

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 [53]. Среди молодежи суточный охват занимает 76% [48]. 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, которое направлено улучшить осведомленность будущих студентов ВятГУ.

Объектом исследования являются 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 [**Ошибка! Источник ссылки не найден.**];

- 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

1.1 Telegram-боты

Telegram-боты – это программные агенты, которые взаимодействуют с пользователем через интерфейс мессенджера Telegram. Они используют Telegram Bot API [43] для взаимодействия с серверами Telegram. Эти агенты позволяют организовать удобный интерфейс от навигации по информационной базе до интеграции с внешними сервисами.

1.2 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. [27].

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 [**Ошибка! Источник ссылки не найден.**].

1.3 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 [44].

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 [42].

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 [45].

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.

1.4 Обзор аналогов

1.4.1 Admissions KFU [17]

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.

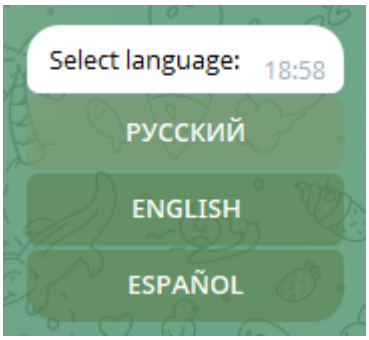


Рисунок 1.1 – Меню выбора языка в «Admissions KFU»

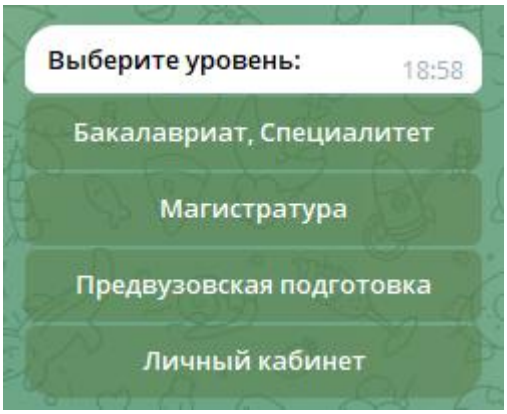


Рисунок 1.2 – Меню выбора уровня образования в «Admissions KFU»

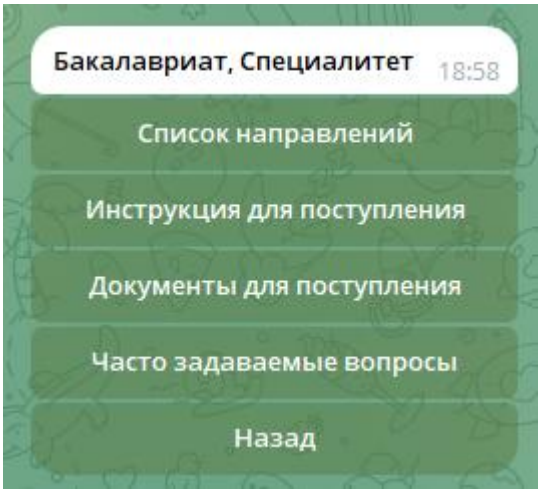


Рисунок 1.3 – Список возможностей в категории Бакалавриат, Специалитет в «Admissions KFU»

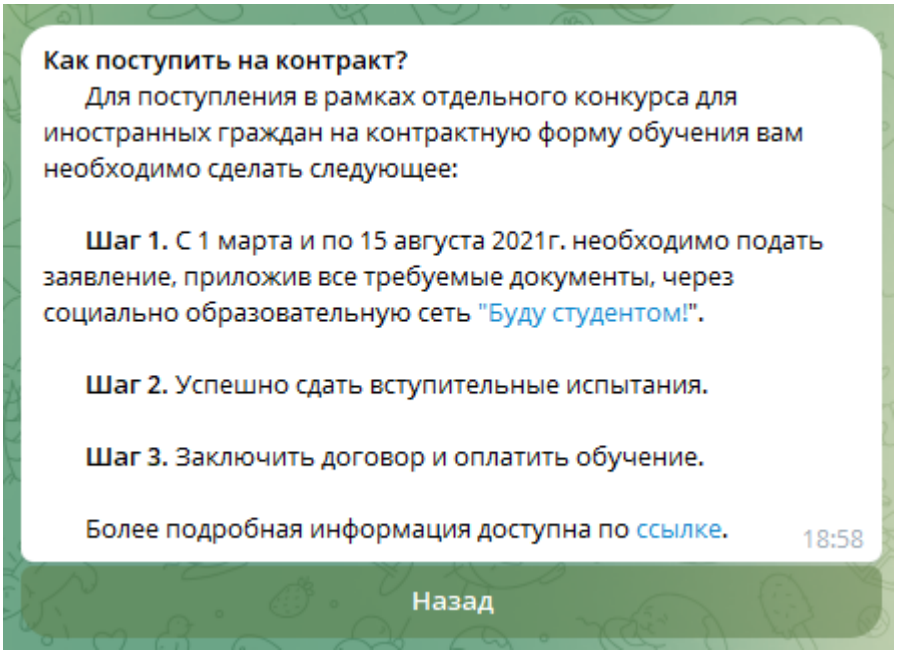


Рисунок 1.4 – Ответ на вопрос в «Admissions KFU»

1.4.2 Московский политех [21]

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.

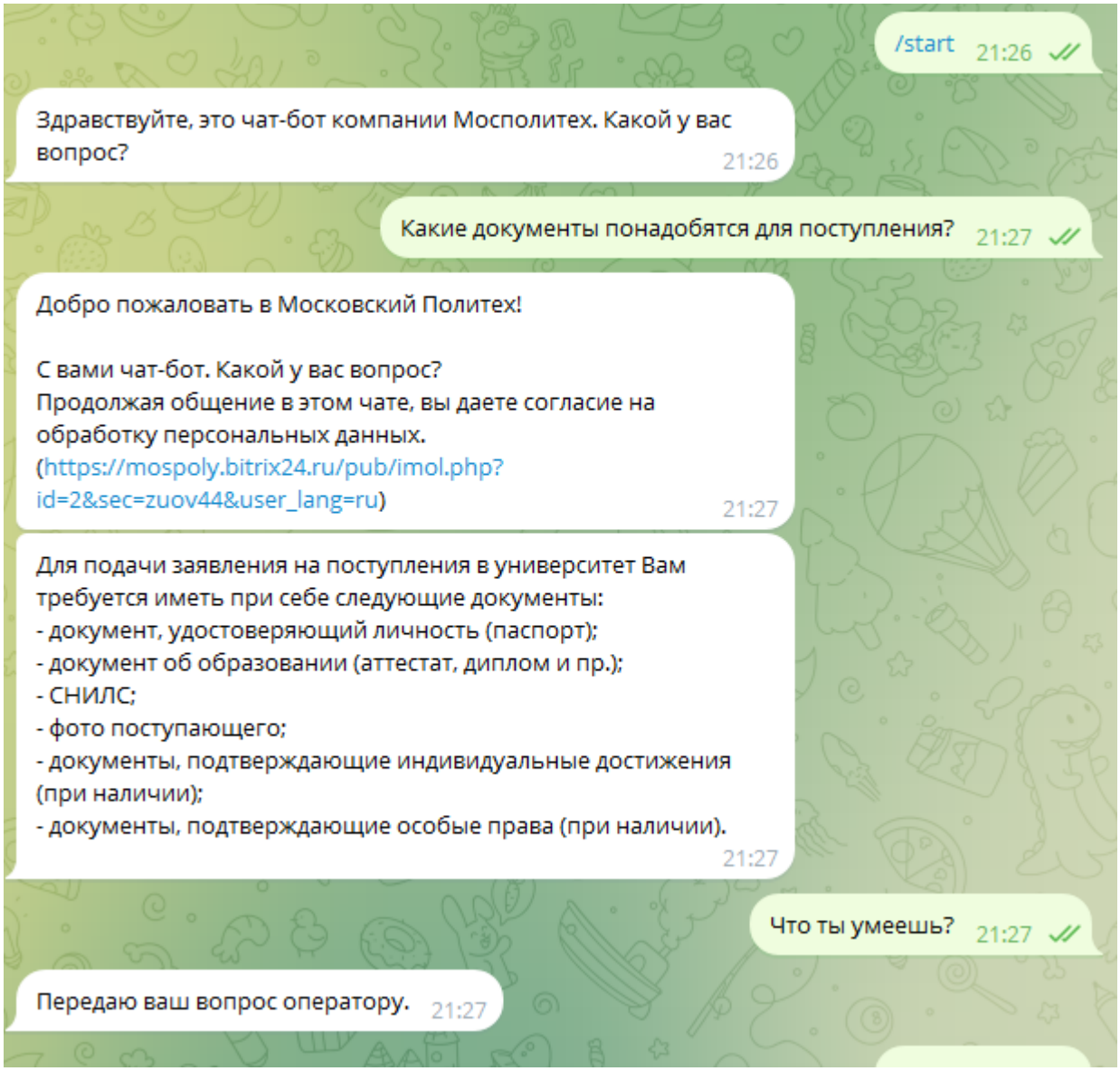


Рисунок 1.5 – Диалог с ботом «Московский политех»

1.4.3 Хочу в Политех [39]

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 [46], содержит 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.

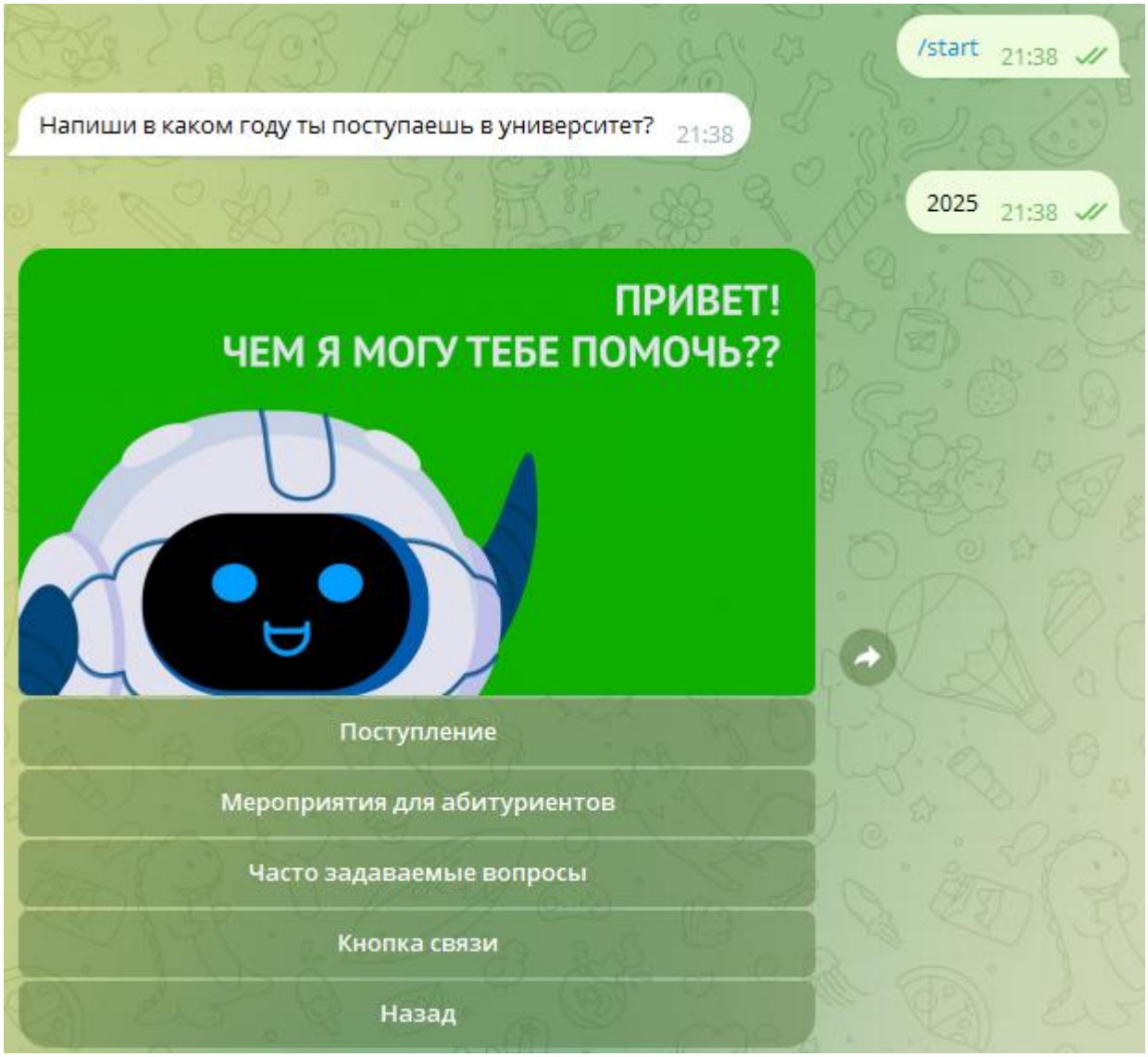


Рисунок 1.6 – Начало диалога с «Хочу в Политех»

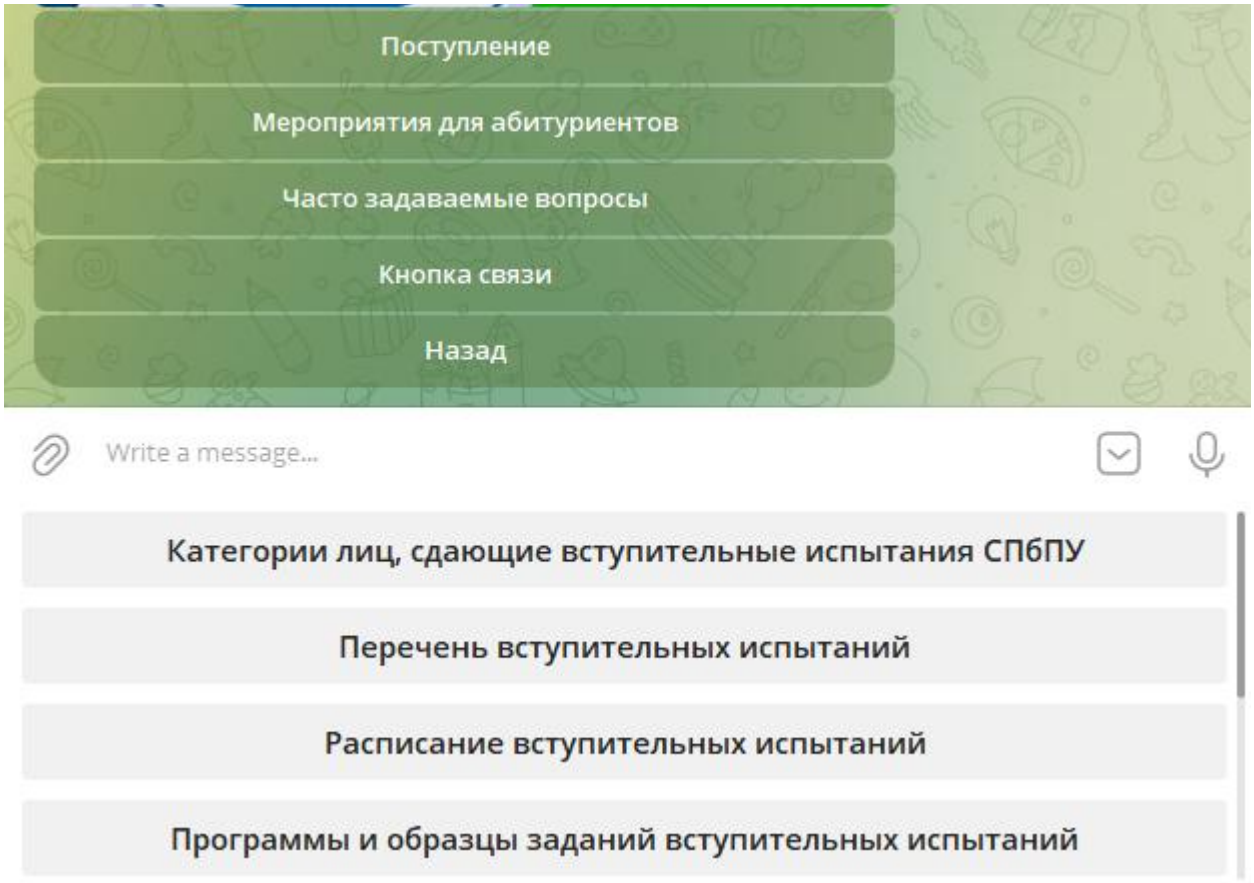


Рисунок 1.7 – Список вопросов в «Хочу в Политех»

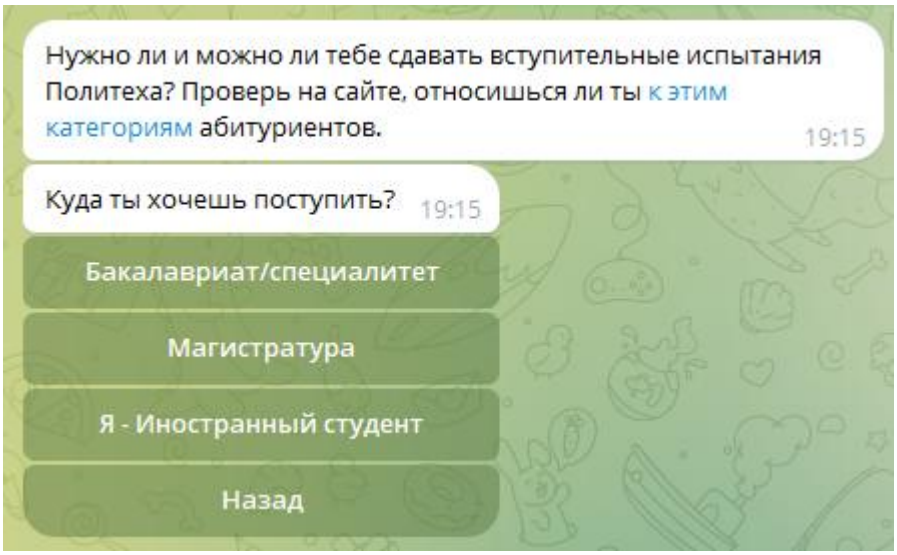


Рисунок 1.8 – Информационное меню по поступлению в «Хочу в Политех»

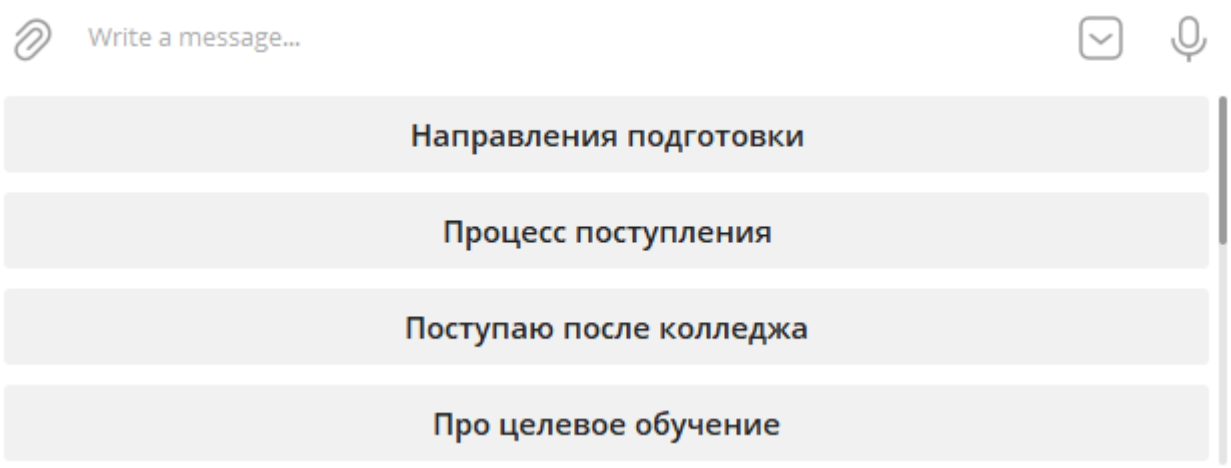


Рисунок 1.9 – Информационное меню по поступлению (клавиатура с вопросами) в «Хочу в Политех»

1.5 Анализ и категоризация вопросов

Одной из ключевых задач при разработке чат-бота для абитуриентов стало создание древовидной структуры вопросов для обеспечения удобной навигации, сокращения количества уточняющий вопросов и улучшения релевантности ответов

Вопросы были составлены на основе официального документа [54]. Кроме того, были собраны часто встречающиеся вопросы на сайте ВятГУ для абитуриентов [**Ошибка! Источник ссылки не найден.**]. В процессе:

– документ быт структурирован на логические разделы;

– к каждому значимому положению правил был сформулирован один или несколько вопросов в разговорном стиле, близком к реальной речи абитуриентов (например: «А можно ли подать документы онлайн?»).

В общей сложности было собрано 259 вопросов. Ниже несколько примеров:

Как можно поступить в ВятГУ: через ЕГЭ, экзамены или без них?

Какие дипломы подойдут, чтобы поступить в ВятГУ?

По какому порядку распределяются квоты при поступлении на бюджет в ВятГУ?

Если профиль один и тот же, будут ли отличаться испытания и баллы на разных формах обучения?

Где посмотреть, приняли ли моё заявление и на каком я месте в списке?

Категории были созданы на основе структуры положения о приеме и на местоположении вопроса на сайте ВятГУ. Пример структуры категории «Часто задаваемые вопросы»:

├── Часто задаваемые вопросы
│ ├── Подача документов и поступление
│ │ ├── Способы подачи документов
│ │ ├── Общие вопросы
│ │ ├── Перевод и восстановление
│ │ ├── ЕГЭ и вступительные испытания
│ │ └── Целевой прием и платное обучение
│ ├── Общежитие
│ └── Направление «Педагогическое образование (с двумя профилями подготовки)»

Полная структура дерева вопросов приведена в приложении А.

Этот этап стал фундаментом для реализации системы поиска ответов на вопросы пользователей о поступлении в ВУЗ.

1.6 Составление требований

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

1.6.1 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.

1.6.2 Бизнес-правила

- информация должна поступать только из официальных источников;
- основная цель бота – помочь абитуриенту с вопросами и с определением направления подготовки;
- бот не должен сообщать сведения, не подтвержденные в официальных источниках.

1.6.3 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 топ-5 подходящих направлений.

1.6.4 Функциональные требования

Система должна справляться с опечатками пользователя.

1.6.5 Технические требования

- Использовать очередь для соблюдения лимитов на отправку сообщений;
- соблюдать ограничения на сообщения;
- использовать сервисы для кеширования запросов к базе данных.

1.6.6 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.

1.6.7 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.

1.7 Итоги анализа предметной области и существующих решений

В ходе обзора предметной области изучены особенности разработки Telegram-ботов. Это определило выбор архитектурных решений при проектировании системы.

Проведён анализ существующих ботов, ориентированных на поддержку абитуриентов. Выявлены их сильные и слабые стороны.

С официального сайта ВятГУ была собрана база из 259 вопросов и

ответов, представляющих интерес для абитуриентов. Они были категорированы для более удобной навигации

Требования к разрабатываемому чат-боту были определены на основе изучения аналогов и на проведенном анализе предметной области. В результате сформулированы три основные функции:

- рекомендация образовательной программы, исходя из интересов пользователя;
- поиск ответов на вопросы с использованием базы знаний;
- уведомления о важных событиях образовательных программ.

2. Аргументация технических решений

2.1 Языки программирования

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

2.1.1 Python [30]

По данным TIOBE Index, популярность Python достигла 25,35% на состояние мая 2025 года, что является самой высокой долей за всю историю индекса [49]. Согласно отчету GitHub Octoverse 2024, Python обогнал JavaScript и стал самым популярным языком на платформе [15]. 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 [**Ошибка! Источник ссылки не найден.**].

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

2.1.2 C# [Ошибка! Источник ссылки не найден.]

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 [15]. Язык стабильно входит в топ-5 TIOBE, что говорит о его востребованности в [49].

Поддерживает асинхронное программирование, что отлично подходит

для разработки Telegram-ботов.

На официальной платформе NuGet доступно более 100 000 уникальных пакетов [52]. 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 решения (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#.

2.1.3 Java [16]

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 [15]. Так же как C#, стабильно входит в топ-5 языков программирования согласно этому индексу [49].

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.

2.1.4 Обоснование выбора Python

Таблица 2.1 – Сравнение языков программирования

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.

2.2 Фреймворки

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

2.2.1 Python-telegram-bot [31]

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.

2.2.2 **Aiogram [**Ошибка! Источник ссылки не найден.**]**

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).

2.2.3 Pyrogram [Ошибка! Источник ссылки не найден.]

Этот фреймворк ориентирован на работу с пользовательскими аккаунтами через Telegram MTProto API [22]. 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:

- прекращено развитие фреймворка, решением может быть переход на один из форков;
- не имеет встроенной поддержки middleware;
- избыточен для обычного информационного бота.

2.2.4 Обоснование выбора aiogram

Сводная таблица, обобщающая достоинства и недостатки рассмотренных фреймворков:

Таблица 2.2 – Сравнение 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.

2.3 СУБД

В этом разделе будут рассмотрены несколько популярных СУБД.

2.3.1 PostgreSQL [26]

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 [8].

49% разработчиков используют PostgreSQL в своих проектах. Это самая популярная СУБД второй год подряд согласно stackoverflow developer survey по данным 2024 года [**Ошибка! Источник ссылки не найден.**].

Имеет более 17,6 тысяч звезд на github.com по состоянию на 24 мая 2025 года [26].

Показывает высокую производительность на практике.

2.3.2 SQLite [Ошибка! Источник ссылки не найден.]

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 [8]. Имеет более 7,8 тысяч звезд на github.com по состоянию на 24 мая 2025 года [**Ошибка! Источник ссылки не найден.**].

Особенно 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.

2.3.3 MySQL [23]

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 [8].

Показывает высокую производительность, хотя уступает PostgreSQL.

2.3.4 Обоснование выбора PostgreSQL

Таблица 2.3 – Сравнение реляционных баз данных

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

2.4 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.

2.4.1 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 [24].

2.4.2 Обзор векторных баз данных

В таблице 2.4 было рассмотрено несколько решений.

Таблица 2.4 – Сравнение векторных баз данных

Название	**Лицензия**	**Фильтрация по метаданным**	**Особенности**	**Звезды GitHub (24.05.2025)**
Qdrant [32]	Apache 2.0	Полноценная поддержка	gRPC/REST, Fast embeddings search, кластеризация	23,7 тыс. звезд
Pinecone [**Ошибка! Источник ссылки не найден.**]	Коммерческая	Частично (через API)	SaaS[1], легко масштабируется, без локального запуска	–
Weaviate [**Ошибка! Источник ссылки не найден.**]	Apache 2.0	Через GraphQL	модули для трансформерных моделей, встроенные модели	13,4 тыс. звезд
FAISS [**Ошибка! Источник ссылки не найден.**]	MIT	Нет встроенной поддержки	Библиотека от Meta[2], хороша для оффлайн-поиска, но не полноценная БД	35,1 тыс. звезд
Milvus [20]	Apache 2.0	Через JSON-фильтры	Поддержка кластеров,	34,9 тыс. звезд

Стоит отметить, что все рассмотренные варианты имеют официальную

[1] SaaS – это модель распространения программного обеспечения, при которой приложения размещаются в облаке и предоставляются пользователю как сервис по подписке.

[2] Признана экстремистской и запрещена на территории РФ

поддержку Python и все, кроме FAISS, входят в десятку лучших векторных баз данных по версии рейтинга DB-Engines Ranking на 24 мая 2025 года [9].

2.4.3 Обоснование выбора Qdrant

Для решения задачи была выбрана 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.

2.5 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.

2.5.1 Redis [Ошибка! Источник ссылки не найден.]

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 тыс. 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.

2.5.2 Memcached [Ошибка! Источник ссылки не найден.]

Это 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 тыс. 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.

2.5.3 Dragonfly [Ошибка! Источник ссылки не найден.]

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 тыс. звезд на github.com и занимает 35 место среди key-value хранилищ согласно DB-Engines (24 мая 2025 года).

2.5.4 Обоснование выбора Redis

Таблица 2.5 – Сравнение хранилищ в оперативной памяти

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	Да	Да
Pub/Sub	Да	Нет	Да
Персистентность	Да (RDB, AOF)	Нет	Да

Продолжение таблицы 2.5

Масштабируемость	Кластеры, репликация	Простая горизонтальная (шардирование вручную)	Многопоточность, shared-nothing
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.

2.6 Брокер задач

В выпускной квалификационной работы брокер задач используется для плановой отправки уведомлений. В частности, для уведомления о предстоящих контрольных точка. Это позволяет удобно планировать отправку сообщений по времени, которая не зависит от основного контейнера бота. В таблице 2.6 представлено сравнение популярных решений.

Таблица 2.6 – Сравнение брокеров задач

Параметр	**Arq [2]**	**Celery [4]**	**RabbitMQ [33]**
Поддержка asyncio	Построен на asyncio: полностью асинхронный (неблокирующее выполнение задач).	Синхронная модель по умолчанию; есть поддержка Eventlet/Geven для корутин, но нативно asyncio не использует	Не является Python-библиотекой: сервис на Erlang. Содержит клиентскую библиотеку на Python
Сложность настройки и внедрения	Низкая:	Высокая	Высокая
Масштаби-руемость	Хорошая: за счёт asyncio может выполнять сотни задач одновременноarq-docs.helpmanual.io; масштабируется запуском дополнительных воркеров.	Отличная: поддерживает несколько брокеров и воркеров; обеспечивает высокую доступность и горизонтальное масштабирование	Отличная: ориентирован на кластеризацию (репликация сообщений в кластере) и распределённую обработку.

Продолжение таблицы 2.6

Отложенные и повторяющиеся задачи	Есть: функции enqueue_job поддерживают отложенный запуск, встроенная поддержка cron-задач.	Есть: поддержка отложенного старта	Поддерживается косвенно: сообщения можно задерживать комбинацией TTL и Dead-Letter Exchange
Производи-тельность	Высокая: благодаря asyncio, в простых задачах в несколько раз быстрее аналогов	Очень высокая: один процесс может обрабатывать миллионы задач в минуту	Очень высокая: оптимизирован под высокую нагрузку и throughput; реальная скорость зависит от конфигурации и операций.
GitHub stars (июнь 2025)	2,5 тыс.	26,6 тыс.	12,9 тыс.

Для бота в поддержку абитуриентов, работающего на Python и часто выполняющего неблокирующие задачи (запросы к БД, сетевые вызовы, отправку уведомлений), преимущество имеет легковесный асинхронный брокер Arq. Специально построен на asyncio для неблокирующих операций. и оптимизирован под простые задачи (документация подчёркивает, что он «мал и прост» (около 700 строк кода). При этом он поддерживает отложенный запуск и cron-планировщик без дополнительной настройки.

2.7 Среда исполнения и развертывание приложения

2.7.1 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.

2.7.2 Docker в качестве решения проблемы [11]

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.

2.8 Выбор лингвистической модели векторизации

Существует огромное количество моделей, которые используются для семантического поиска. Наиболее важные требования для нашей системы:

- обязательно требуется поддержка русского языка. На данном этапе в других языках нет необходимости, так как все данные на сайте на русском языке;

- хороший уровень решение задачи семантического свойства для определения схожих текстов. Для этого модели сравним по корреляции Спирмена [10]. В задачах STS эта метрика позволяет сравнивать порядок близости текстовых пар, предсказанный моделью, с порядком, заданным аннотаторами.

- модель должна быть небольшой, чтобы не нагружать систему;

- модель должна быть быстрой для отзывчивого поведения бота и должна запускаться на CPU.

В таблице 2.7 приведено сравнение нескольких популярных моделей по ключевым параметрам.

Таблица 2.7 – Сравнение моделей векторизации

Модель (ссылка)	**Параметры**	**Языки**	**Spearman (STS)**	**Время инференса на CPU (среднее, с)**
sergeyzh/LaBSE-ru-turbo [**Ошибка! Источник ссылки не найден.**]	128M	Русский	0,864	120,40

Продолжение таблицы 2.7

sergeyzh/LaBSE-ru-sts [**Ошибка! Источник ссылки не найден.**]	129M	Русский	0,845	42,84
sergeyzh/rubert-mini-sts [37]	32,4M	Русский	0,815	6,42
sergeyzh/rubert-tiny-sts [**Ошибка! Источник ссылки не найден.**]	29,4M	Русский	0,797	3,21
cointegrated/LaBSE-en-ru [6]	129M	Русский, Английский	0,794	42,87
Tochka-AI/ruRoPEBert-e5-base-512 [50]	139M	Русский	0,793	43,31
cointegrated/rubert-tiny2 [**Ошибка! Источник ссылки не найден.**]	29,4M	Русский	0,750	3,21

Все данные взяты из карточек моделей на Hugging Face. Тесты проводились на датасете encodechka Корреляция Спирмена измерялась на датасете Encodechka [13].

Модель sergeyzh/rubert-mini-sts обеспечивает наиболее сбалансированное сочетание качества и скорости. Она даёт хорошую корреляцию (0,815) при очень быстром выводе (6,4 с). Модели уровня LaBSE-ru-turbo дают чуть более высокое качество (0,864), но оказываются значительно медленнее (120 с). Модели на основе rubert-tiny2 (rubert-tiny-sts, rubert-tiny2) работают ещё быстрее (3,2 с), но с заметным снижением качества (0,797-0,750). Таким образом, rubert-mini-sts обладает оптимальным компромиссом для

русскоязычного семантического поиска: высокая точность при малом размере и быстродействии.

2.9 Итог по техническим решений

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;

- основная 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.

3. Разработка чат-бота в поддержку абитуриента ВятГУ

3.1 Архитектура системы

3.1.1 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 описание компонентов, 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.

3.1.2 Процесс развертывания

Сначала рассмотрим рисунок 3.1, наглядно демонстрирующий взаимодействие компонентов между собой.

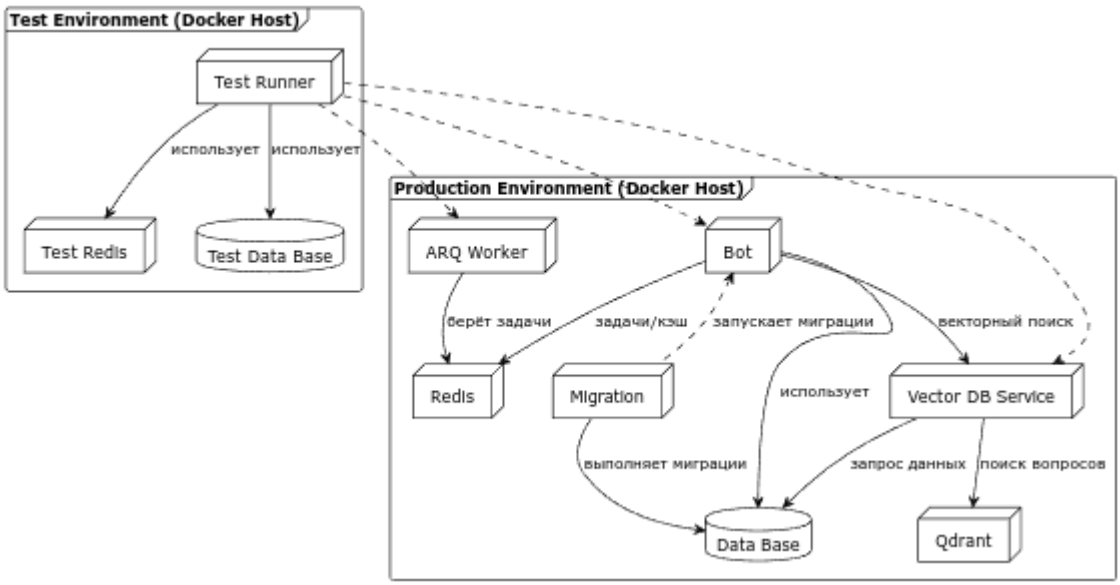


Рисунок 3.1 – Процесс развертывания

Подробнее разберем процесс развертывания, изображенного на рисунке 3.1:

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

- 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;

- одновременно с этим запускается контейнер ARQ Worker;

- в конце запускается Bot, который так или иначе использует все компоненты в Prodaction окружении.

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

3.1.3 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. На рисунке 3.2 показан данный процесс.

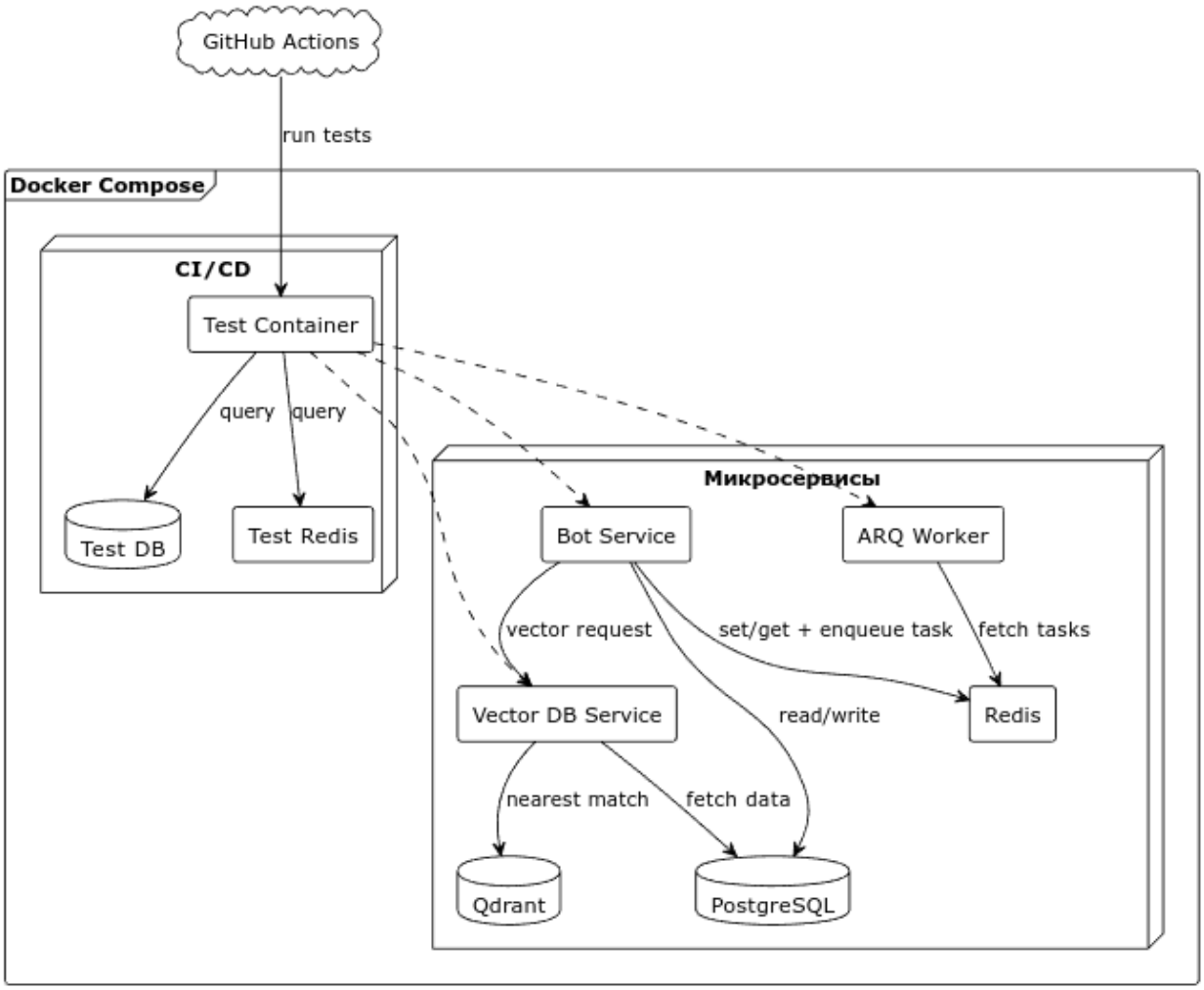


Рисунок 3.2 – Взаимодействие с GitHub Actions

3.1.4 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 3.3.

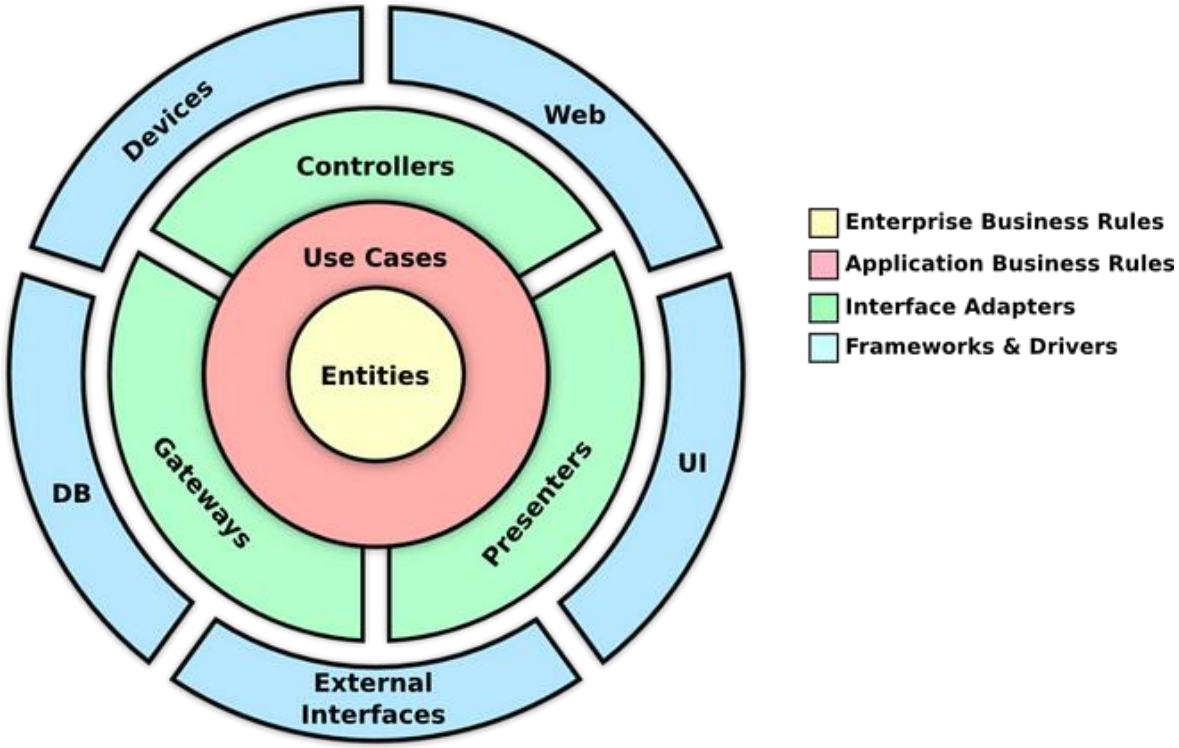


Рисунок 3.3 – Визуализации чистой архитектуры [5]

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 [18].

3.1.5 Структура базы данных

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

На рисунке 3.4 проиллюстрировано взаимодействие сущностей базы данных.

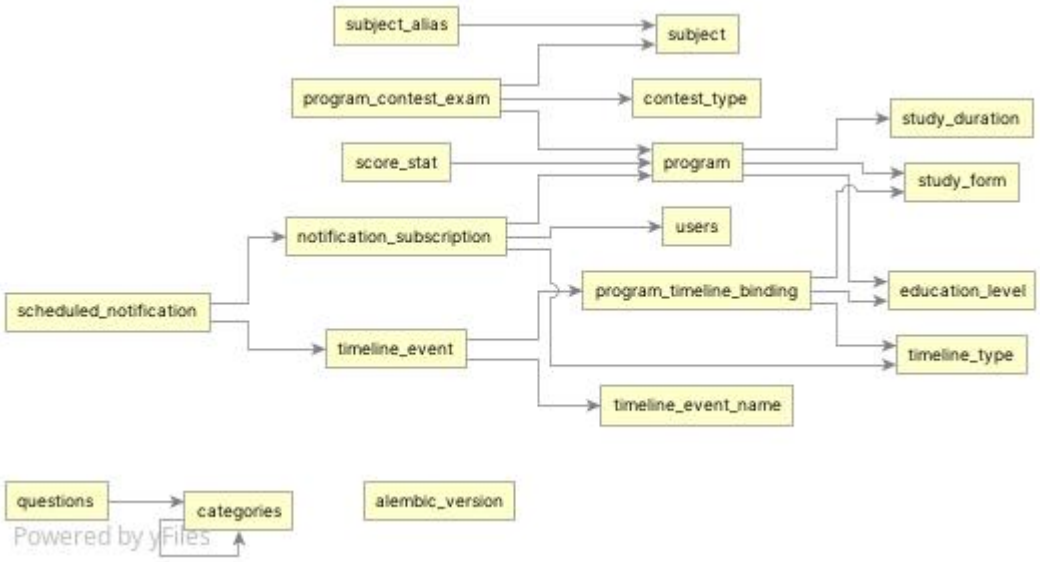


Рисунок 3.4 – Визуализация связей реляционной базы данных

Краткое описание сформированных таблиц:

- users содержит информацию о пользователях. На текущем этапе идентификатор пользователя 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 – справочник названий событий в графике поступления. Используется для унификации и повторного использования событий в разных графиках;

- notification_subscription – представляет собой подписку пользователя на уведомления о событиях программы. Содержит program_id программы и timeline_type_id и id пользователя.

На рисунках 3.5-3.8 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.

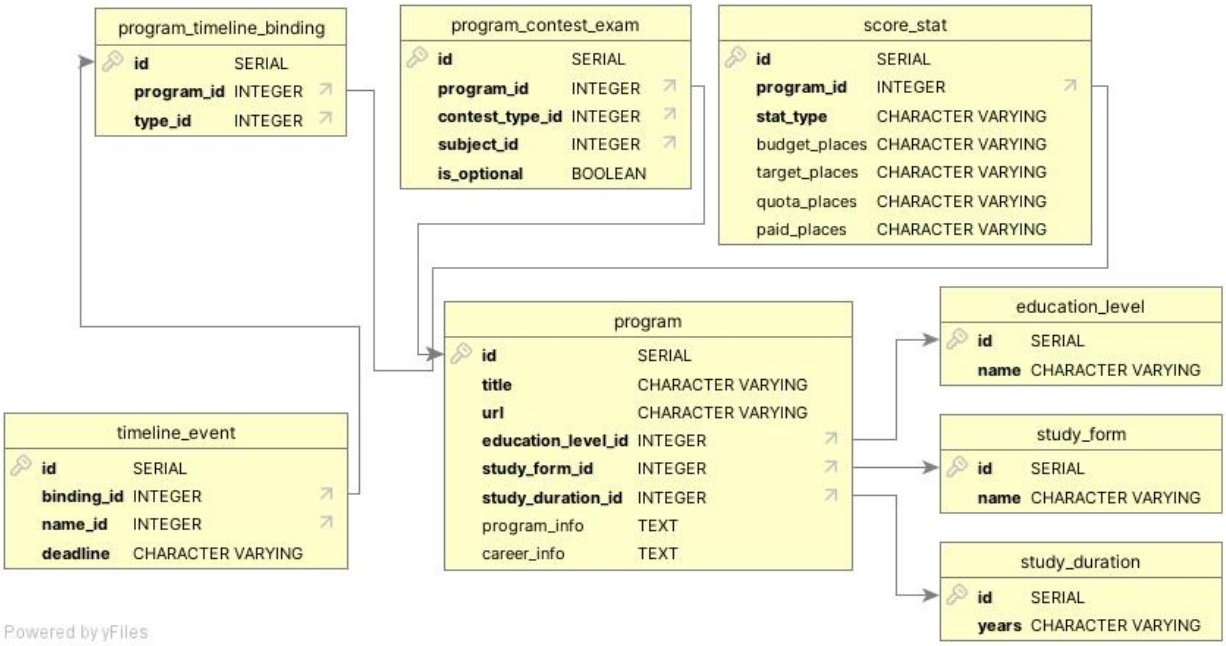


Рисунок 3.5 – Таблица program и все связанные с ней сущности

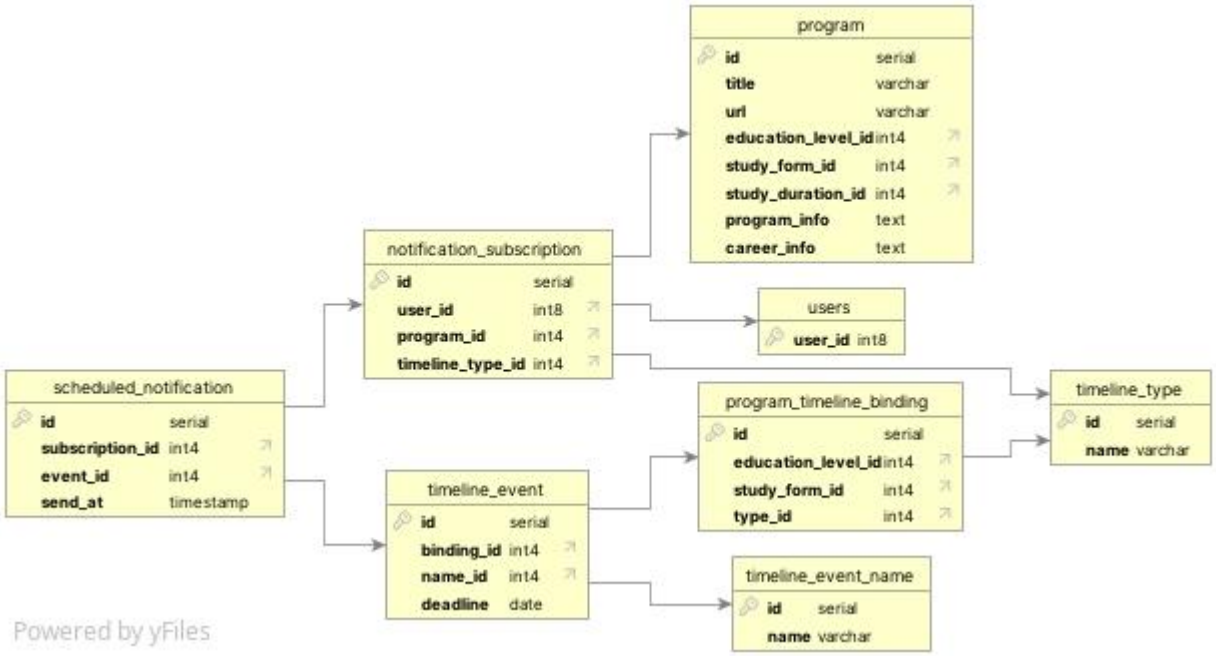


Рисунок 3.6 – Timeline_event и связанные с ней сущности

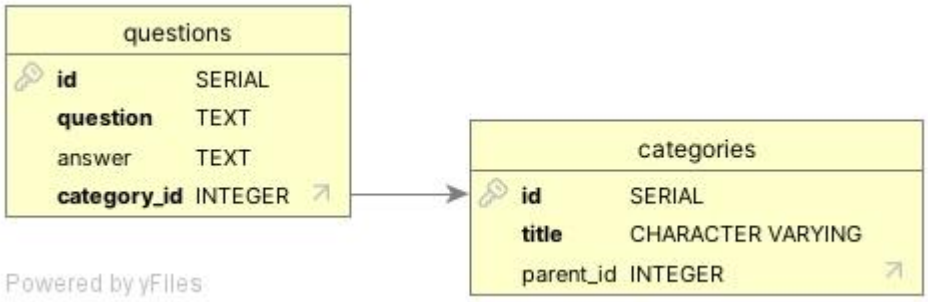


Рисунок 3.7 – Взаимосвязь questions и categories

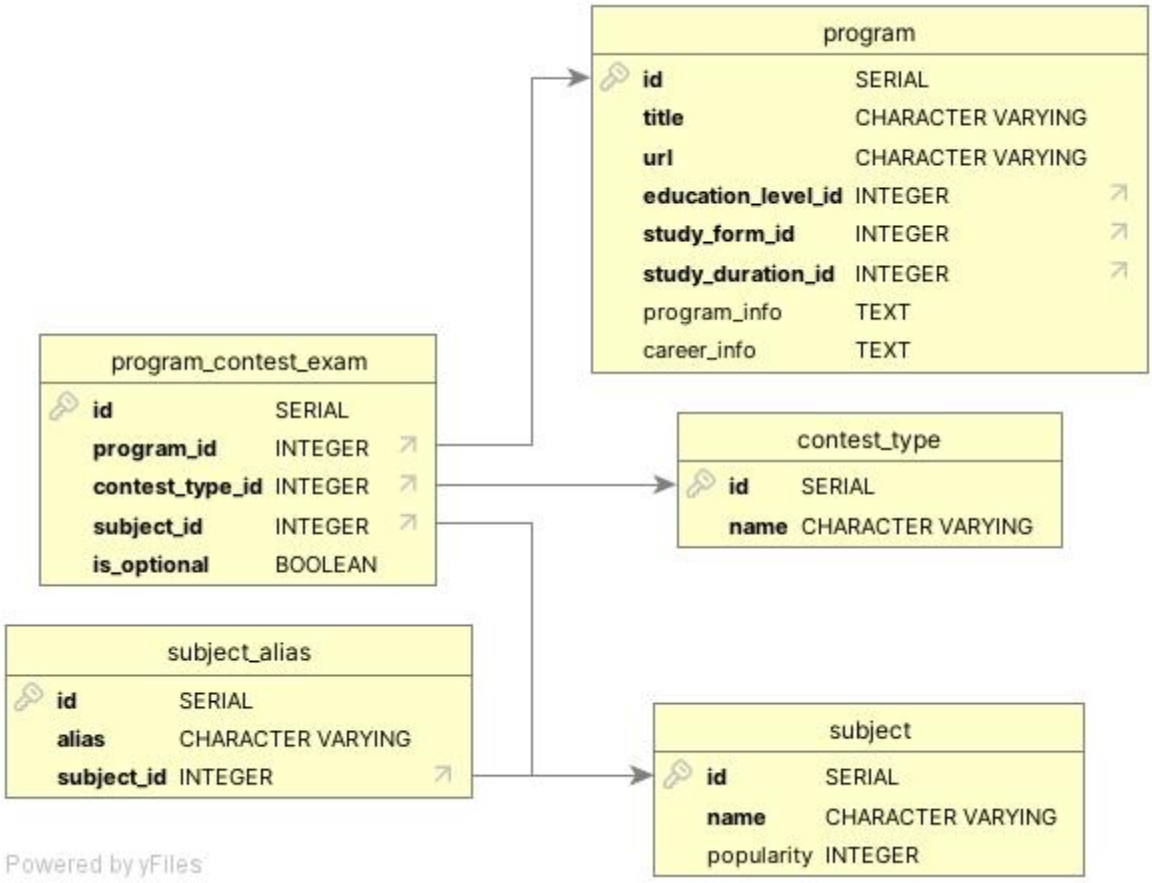


Рисунок 3.8 – Program_context_exam и связанные с ней сущности

3.2 Сценарии взаимодействия с пользователем

В этом разделе рассмотрим основные взаимодействия с пользователем. На рисунке 3.9 изображена диаграмма использования, которая показывает, какие функции включены в бота.

Рисунок 3.9 – Диаграмма использования бота

3.2.1 Вопрос о поступлении

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

Диаграмма состояний на рисунке 3.10 демонстрирует, как пользователь может взаимодействовать с системой.

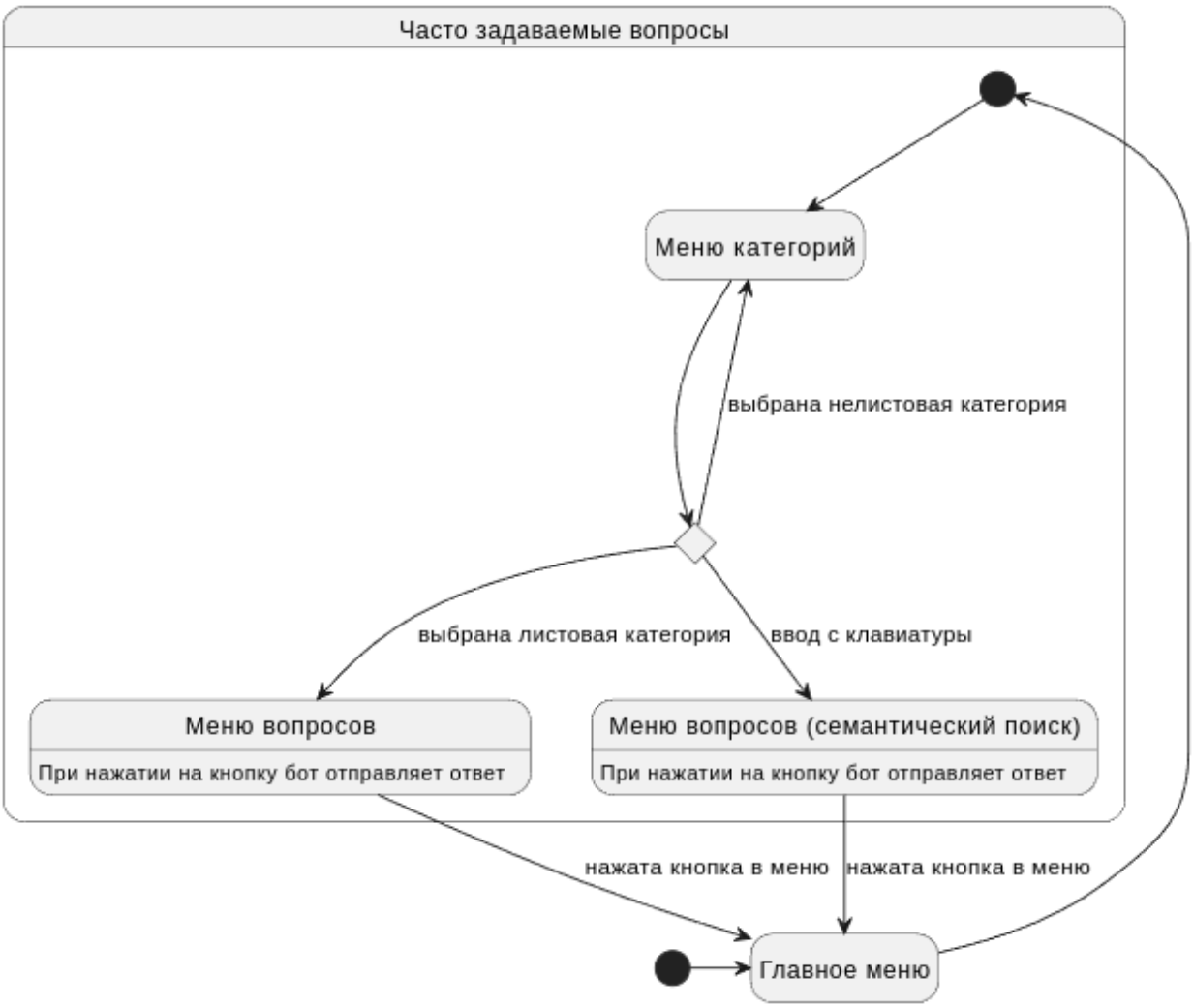


Рисунок 3.10 – Диаграмма состояний пункта «задать вопрос о поступлении»

Для рассмотрения процесса с точки зрения взаимодействия компонентов системы на рисунке 3.11 приведена диаграмма последовательности, иллюстрирующая сценарий выбора категорий пользователем. На ней продемонстрировано взаимодействие с хранилищем Redis и PostreSQL.

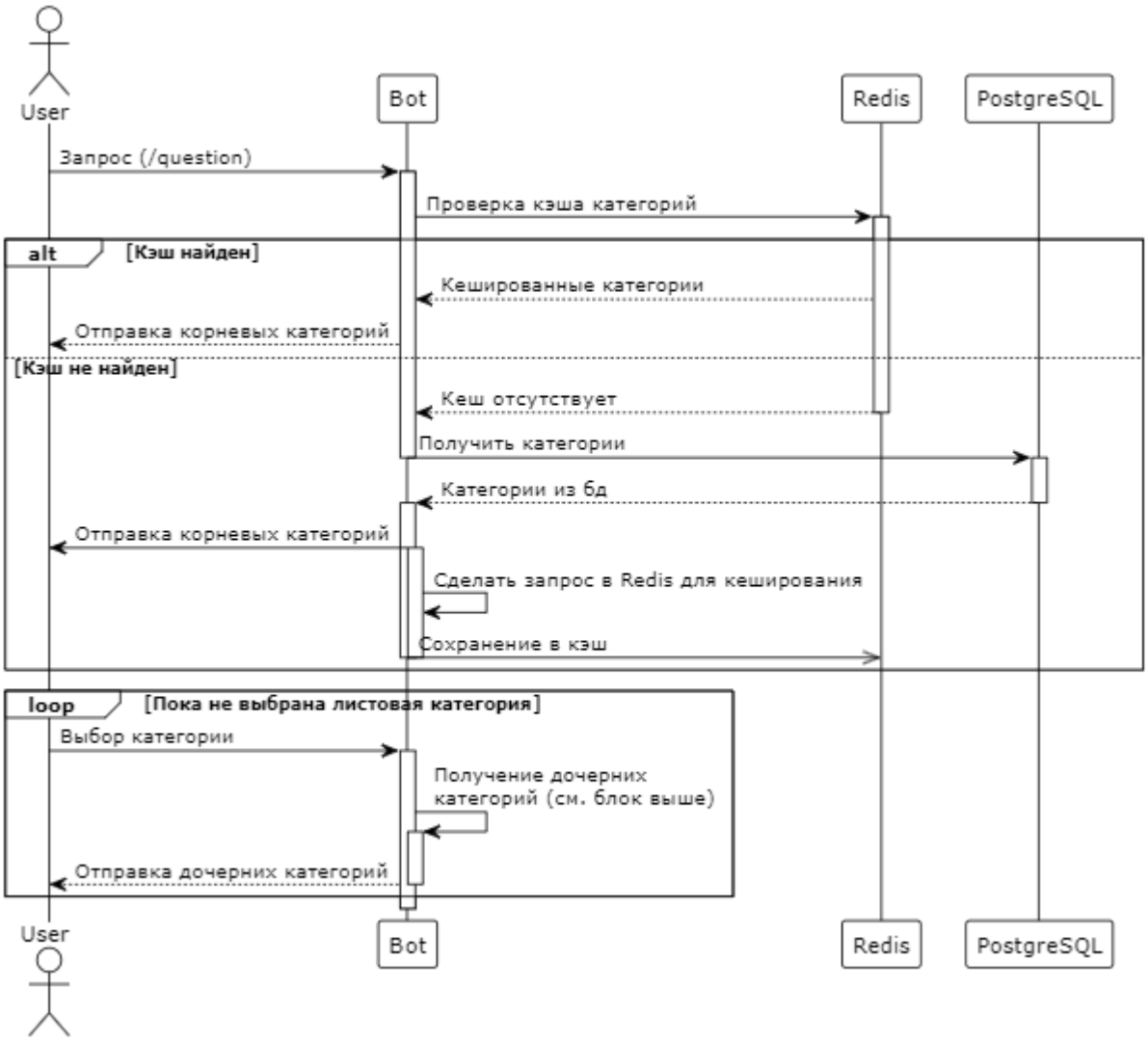


Рисунок 3.11 – Диаграмма последовательности (категориальный ввод)

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.

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

Для рассмотрения процесса с точки зрения взаимодействия компонентов системы на рисунке 3.12 приведена диаграмма последовательности, иллюстрирующая сценарий задания вопроса пользователем в свободной форме. На ней продемонстрировано взаимодействие с хранилищем Vector DB Service и Qdrant.

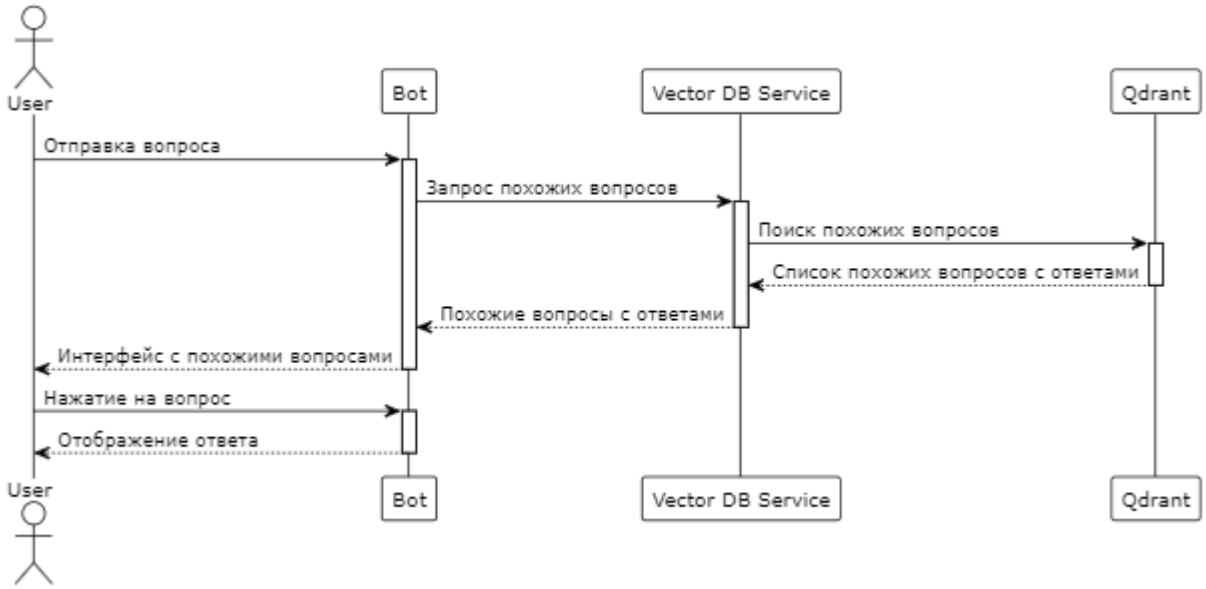


Рисунок 3.12 – Диаграмма последовательности (свободный ввод)

Для лучшего понимания логики взаимодействия с пользователем в мессенджере Telegram на рисунках 3.13-3.17 представлены скриншоты, иллюстрирующие пошаговую работу сценария «Вопрос о поступлении».

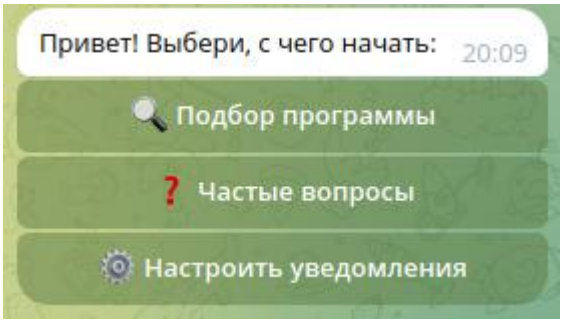


Рисунок 3.13 – Главное меню

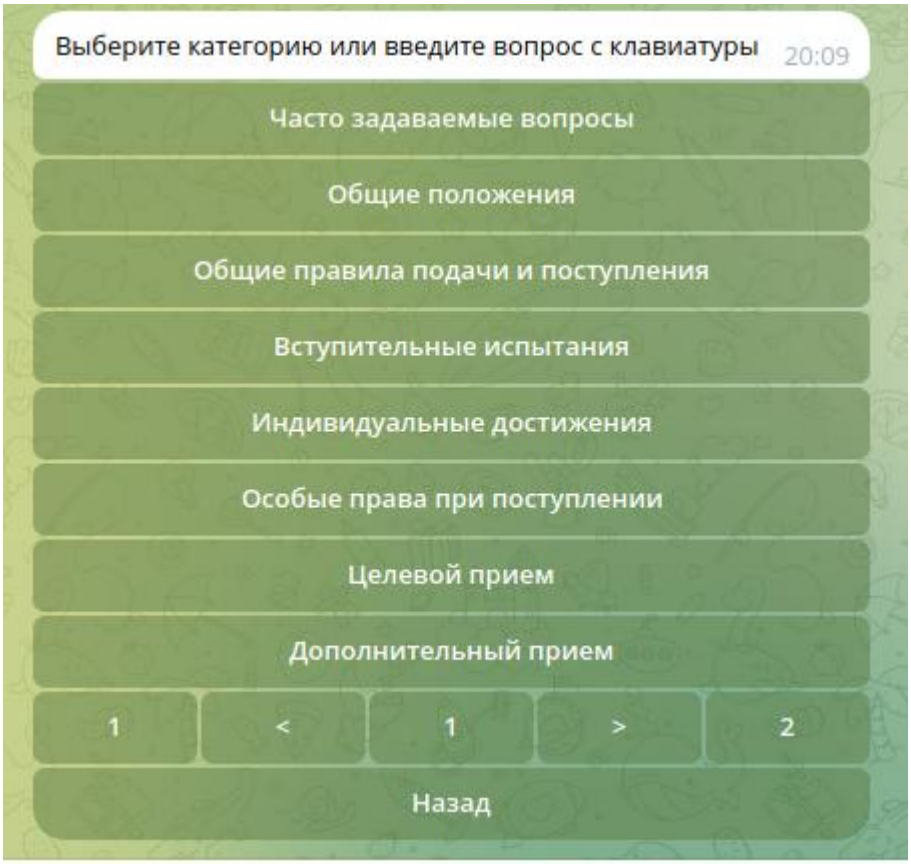


Рисунок 3.14 – Начало диалога задания вопроса боту

Пользователь нажал на кнопку «Общие положения». Бот изменил меню, как показано на рисунке 3.15.

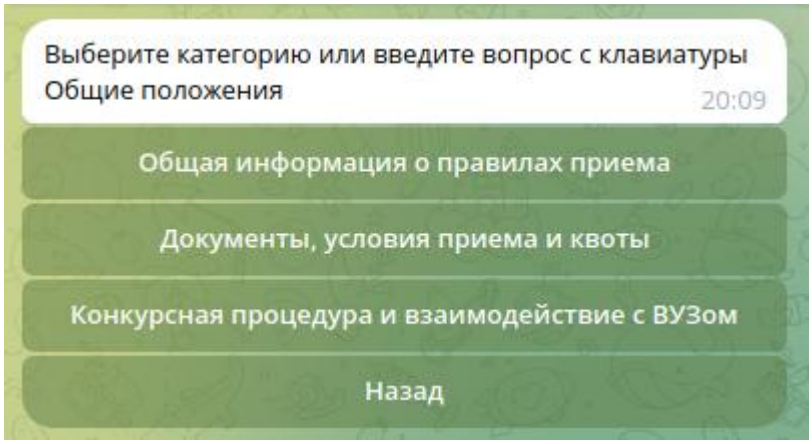


Рисунок 3.15 – Подкатегории вопросов «Общие положения»

Пользователь ввел запрос с клавиатуры «Бюджетные места в ВятГУ» и отправил сообщение. Бот изменил меню, как показано на рисунке 3.16.

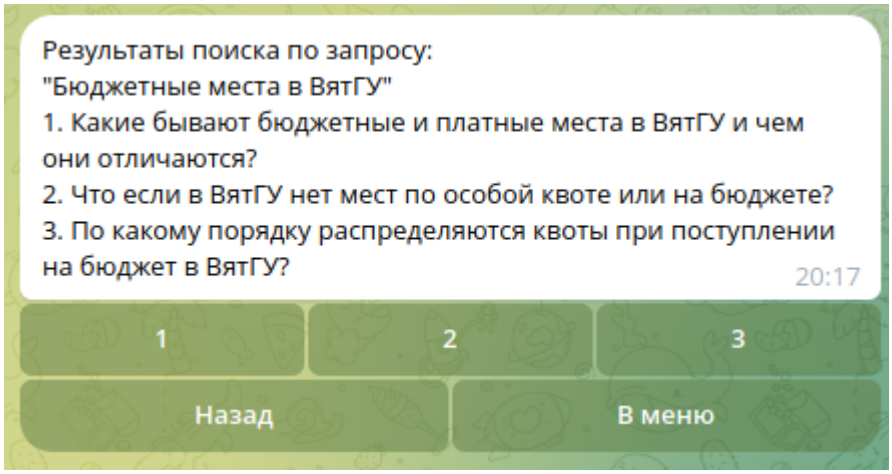


Рисунок 3.16 – Окно вопросов по запросу «Бюджетные места в ВятГУ»

Пользователь нажал на кнопку «2». Бот отправил сообщение, которое изображено на рисунке 3.17.

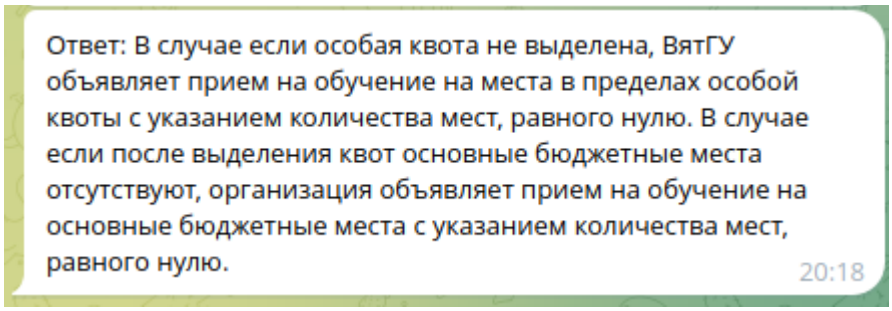


Рисунок 3.17 – Сообщение с ответом на вопрос под номером 2

3.2.2 Получение рекомендованных направлений

Данный сценарий начинается с команды /recommend или с выбора соответствующего пункта в главном меню, после чего бот запускает пошаговый процесс сбора информации о пользователе. На рисунке 3.18 диаграмма состояний для этого сценария.

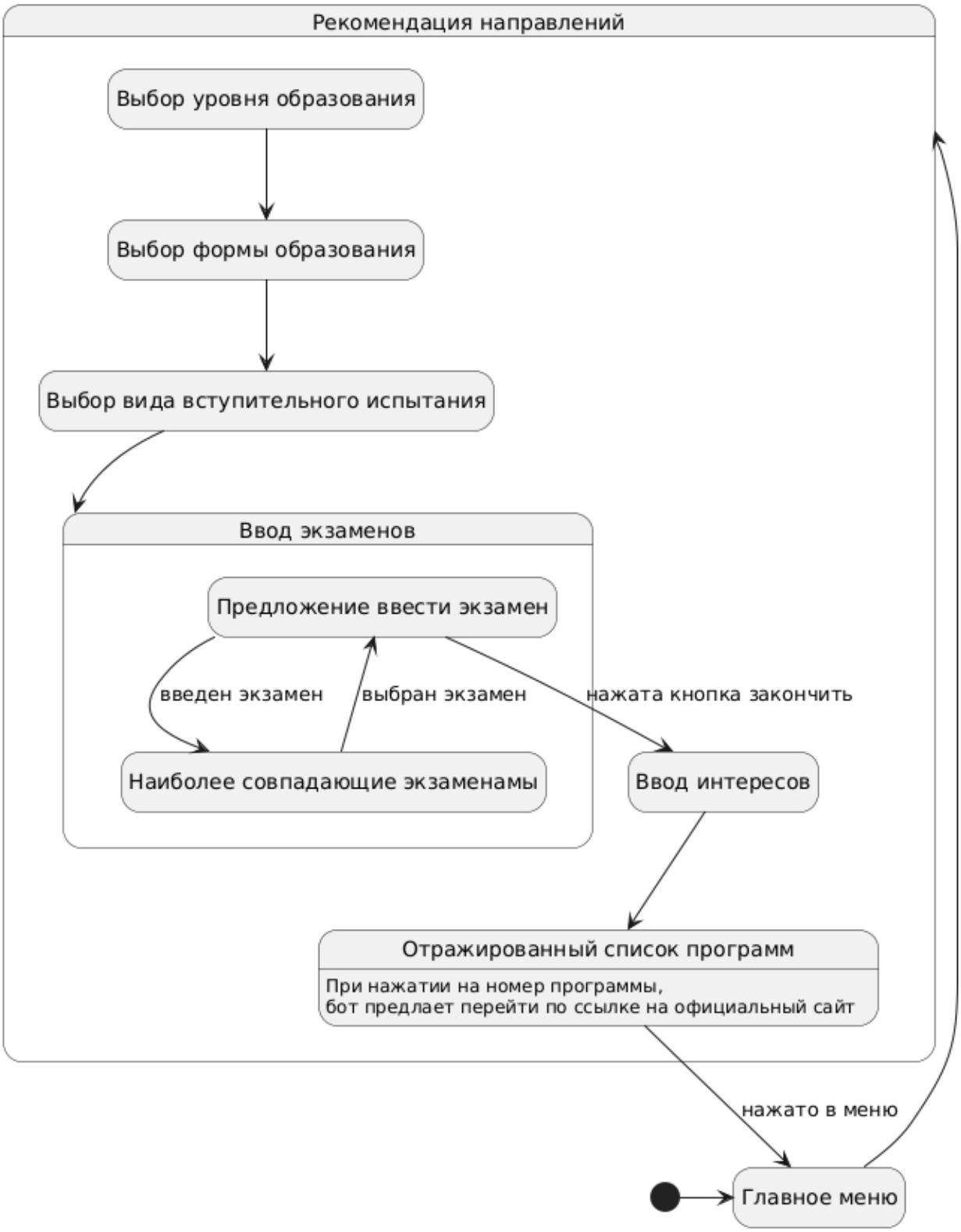


Рисунок 3.18 – Диаграмма состояния сценария получения рекомендованных направлений

Для рассмотрения процесса с точки зрения взаимодействия компонентов системы на рисунке 3.19 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.

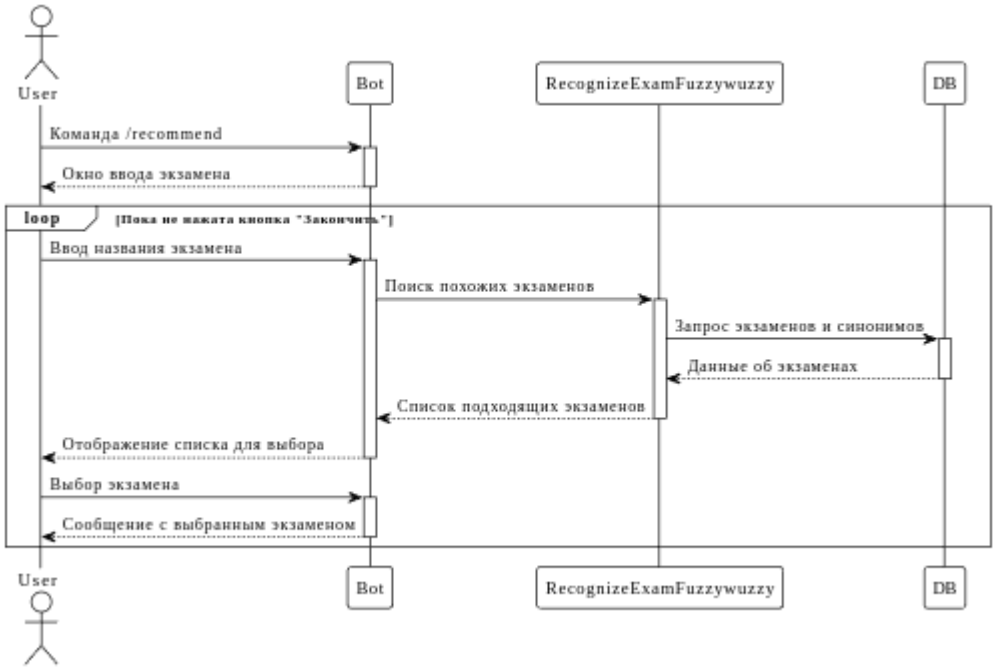


Рисунок 3.19 – Диаграмма последовательности выбора экзаменов.

Для лучшего понимания логики взаимодействия с пользователем в мессенджере Telegram на рисунках 3.20-3.27 представлены скриншоты, иллюстрирующие пошаговую работу сценария «Получение рекомендованных направлений».

Рисунок 3.20 – Окно ввода уровня образования.

Пользователь ввел на клавиатуре «Бакалавриат/Специалитет», открылось окно, показанное на рисунке 3.21.

Рисунок 3.21 – Окно выбора формы обучения

Пользователь нажал на кнопку «Очная». Открылось окно, продемонстрированное на рисунке 3.22.

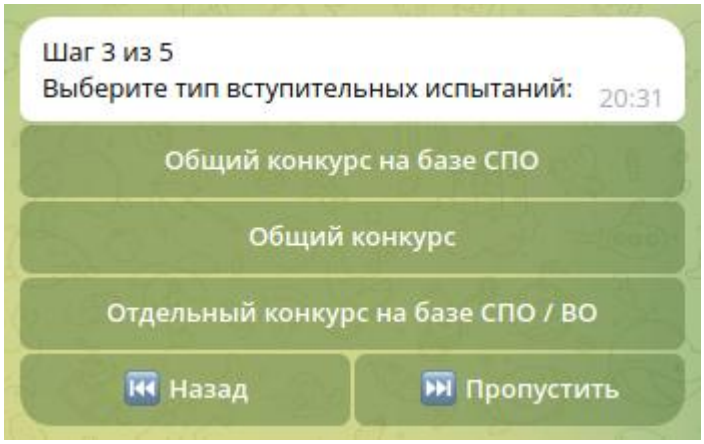


Рисунок 3.22 – Окно выбора тип вступительных испытаний

Пользователь нажал на кнопку «Общий конкурс». Открылось окно, изображенное на рисунке 3.23.

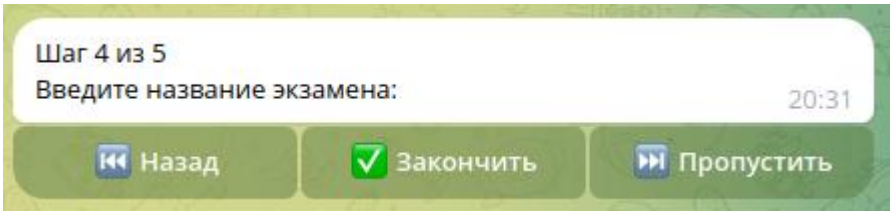


Рисунок 3.23 – Окно ввода экзамена

Пользователь ввел с клавиатуры сообщение «русский», бот распознал один вариант и отправил окно подтверждения ввода (рисунок 3.24).

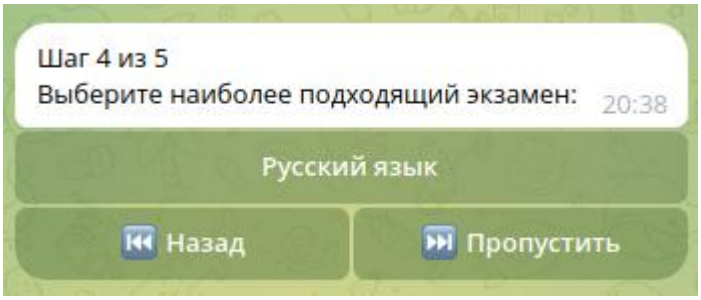


Рисунок 3.24 – Подтверждение ввода экзамена

Пользователь дополнительно ввел экзамены: математика и информатика, после чего нажал на кнопку «Закончить» (рисунок 3.24). Бот отправил окно с предложением ввести интересы пользователя (рисунок 3.25).

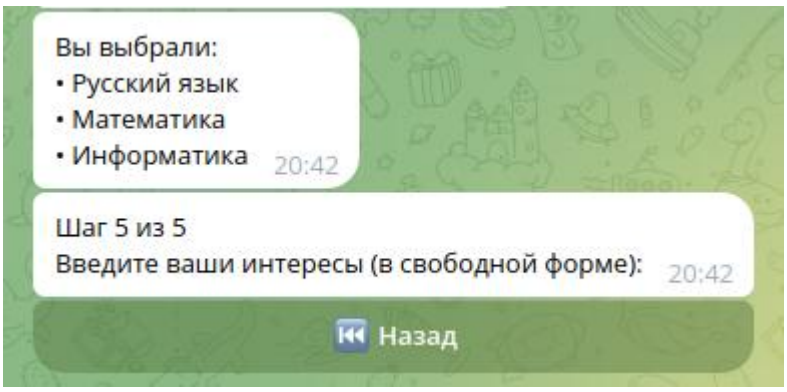


Рисунок 3.25 – Окно ввода интересов пользователя

Пользователь отправил сообщение «Я люблю программировать и хотел бы стать разработчиком по». Бот отправил окно с ранжированными и отфильтрованными направлениями подготовки (рисунок 3.26).

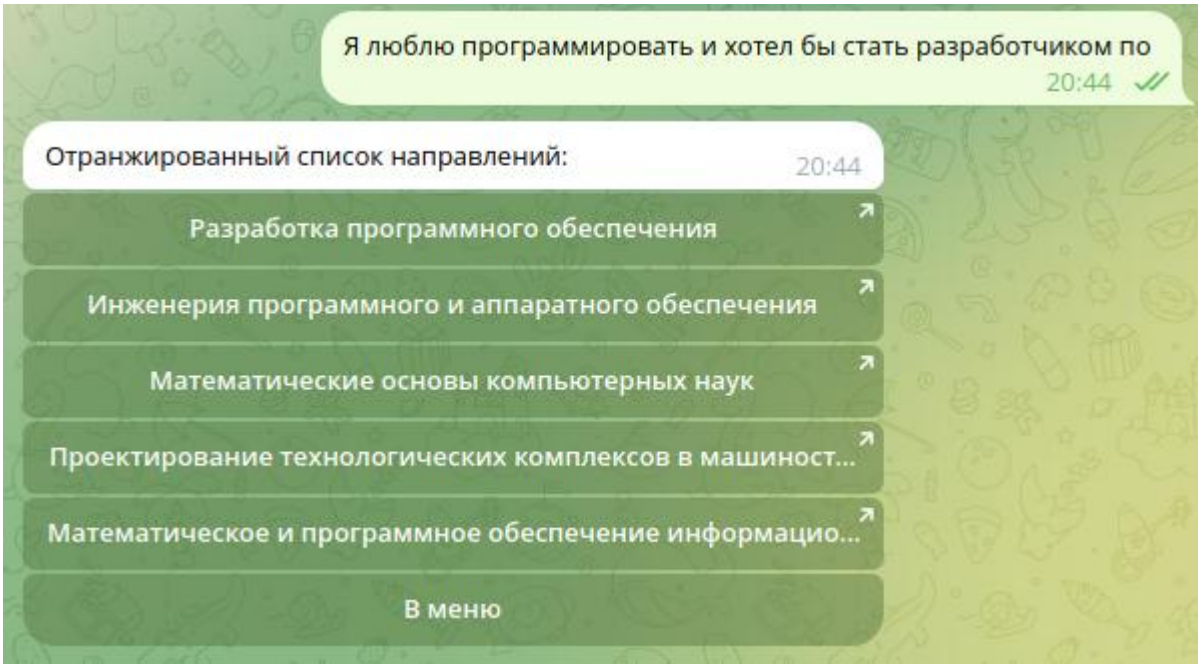


Рисунок 3.26 – Окно выбора направлений подготовки

Пользователь нажал на первую кнопку. Telegram вывел меню с предложением открыть ссылку на указанную программу (рисунок 3.27).

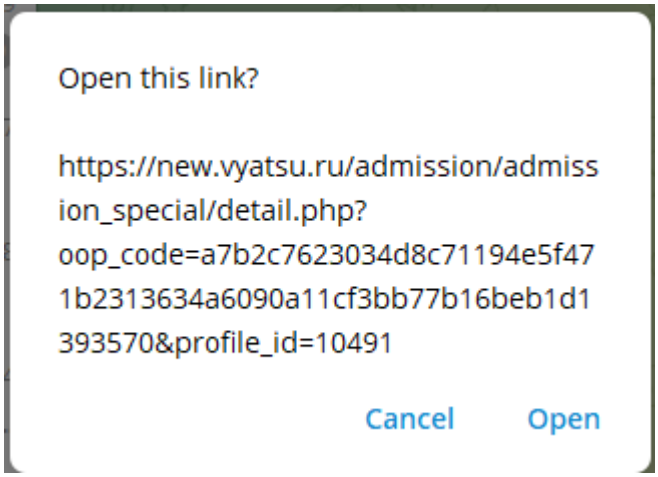


Рисунок 3.27 – Предложение перейти по ссылке.

3.3 Уведомления

Данный сценарий начинается с команды /notification или с выбора соответствующего пункта в главном меню. Отправка сообщений по времени осуществлена с помощью брокера задач arq [2].

На рисунке 3.28 проиллюстрирована диаграмма последовательности, которая демонстрирует процесс подписки на программы и отправки уведомлений.

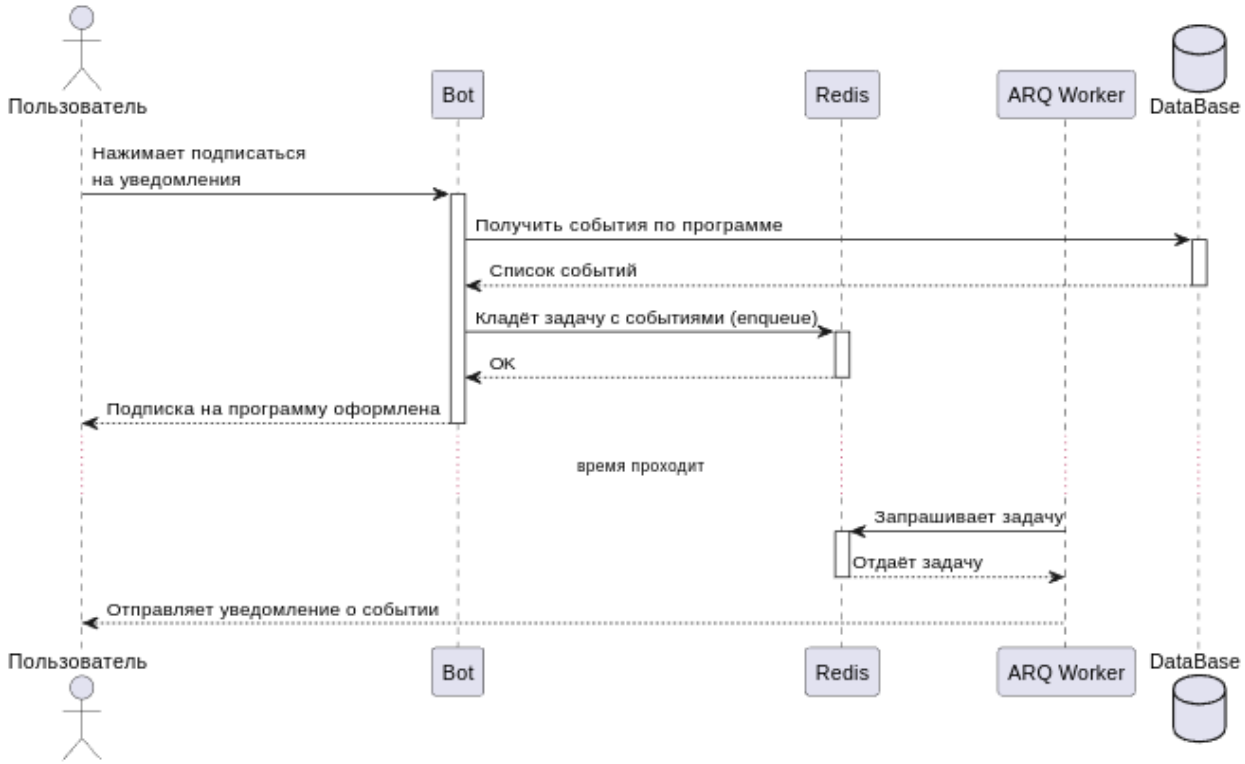


Рисунок 3.28 – Процесс подписки на программы и отправки уведомлений

Для наглядного представления взаимодействия пользователя с интерфейсом бота на рисунке 3.29 продемонстрирована диаграмма состояний пункта настроек подписок.

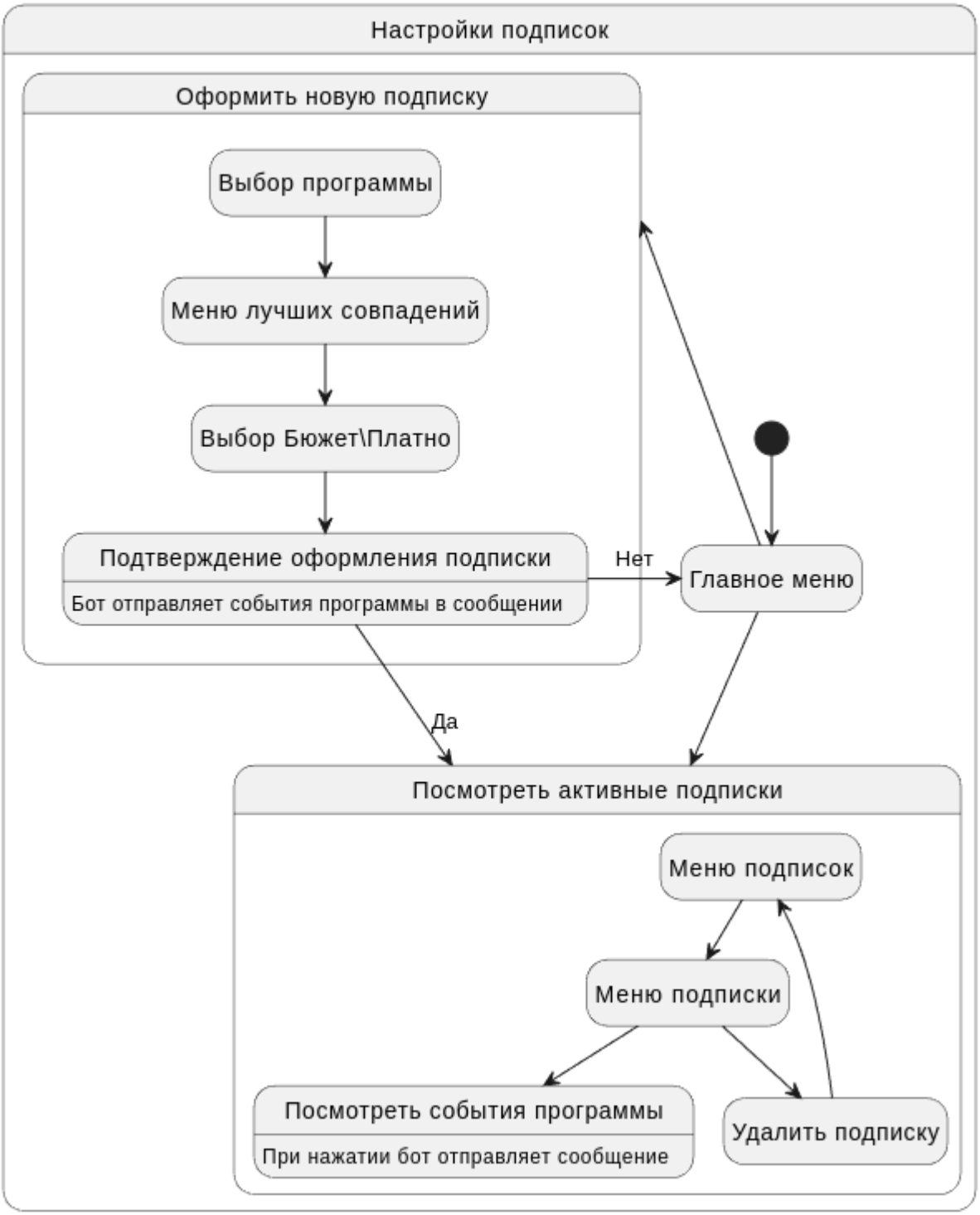


Рисунок 3.29 – Диаграмма состояний пункта настройки подписок

Для большей наглядности на рисунках 3.30-3.38 представлены скриншоты, иллюстрирующие пошаговую работу сценария подписки на уведомления.

Пользователь переходит в диалог настройки уведомлений (через меню или /notification). Бот отправляет окно, изображенное на рисунке 3.30.

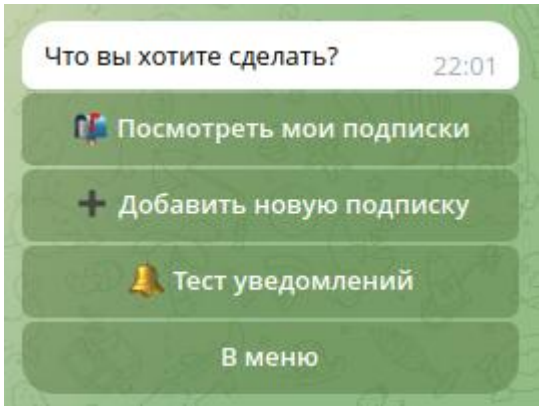


Рисунок 3.30 – Окно действиями при переходе в диалог настройки подписок

Пользователь нажимает «Добавить новую подписку». Появляется окно, предлагающие ввести название программы (рисунок 3.31).

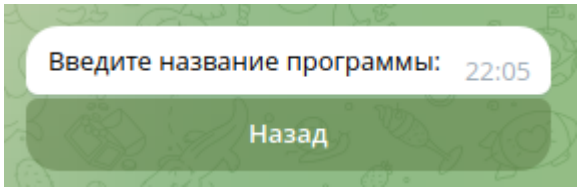


Рисунок 3.31 – Окно с предложением ввести название программы

Пользователь отправляет «разработка программного обеспечения». Бот отправляет окно с предложением выбрать наиболее совпадающую программу по названию (рисунок 3.32).

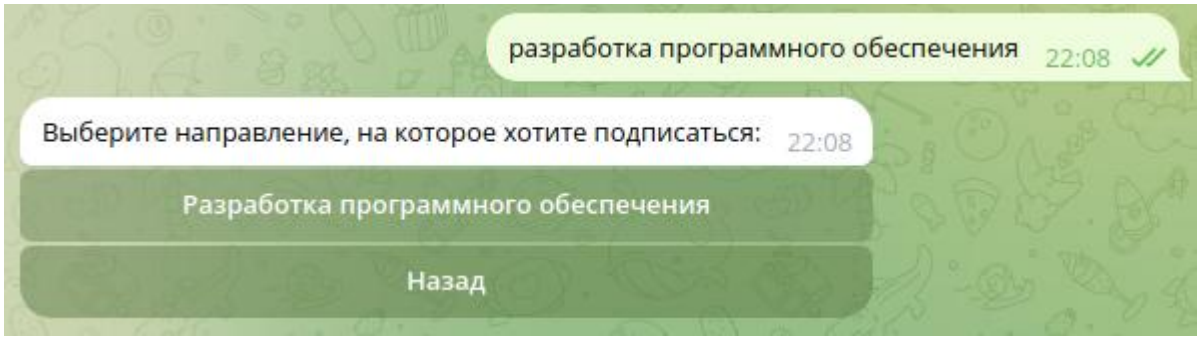


Рисунок 3.32 – Окно с предложением выбрать наиболее совпадающую программу по названию

Пользователь нажимает на «Разработка программного обеспечения». Бот отправляет окно с выбором условия поступления (рисунок 3.33).

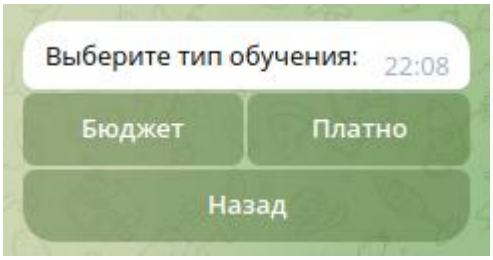


Рисунок 3.33 – Окно с предложением выбрать условия поступления (Бюджет\Платно)

Пользователь нажимает кнопку «Бюджет». Бот отправляет окно, запрашивающие подтверждение оформления подписки на уведомления (рисунок 3.34).

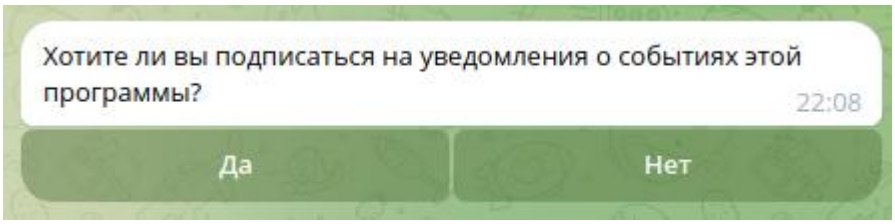


Рисунок 3.34 – Окно подтверждение оформление подписки на уведомления важных событий программы.

Пользователь нажимает «Да». Бот отправляет окно со всеми подписками (рисунок 3.35).

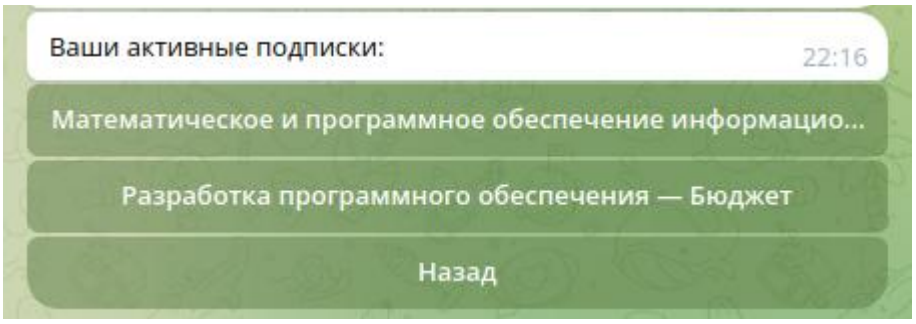


Рисунок 3.35 – Окно с выбором подписки

Пользователь нажал на вторую кнопку. Бот отправил окно с действиями над этой подпиской (рисунок 3.36).

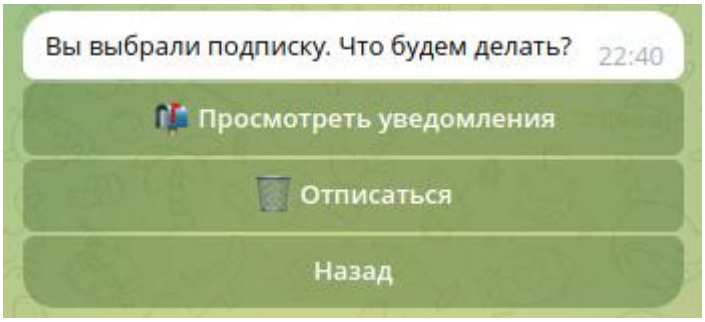


Рисунок 3.36 – Окно с действиями над подпиской

Пользователь нажал на кнопку «Просмотреть уведомления». Бот отправил список всех событий этой программы обучения (рисунок 3.37).

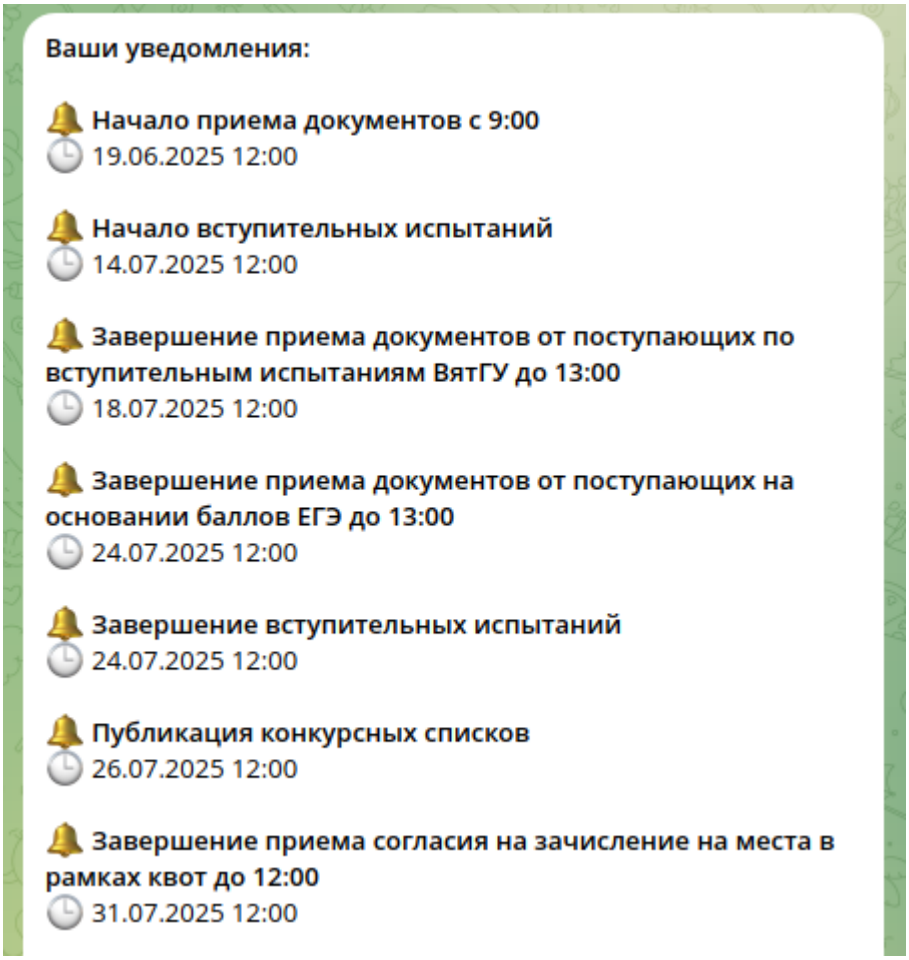


Рисунок 3.37 – Сообщение с событиями выбранной программы

Пользователь нажал кнопку «Отписаться». Бот отправил окно со всеми подписками (рисунок 3.38).

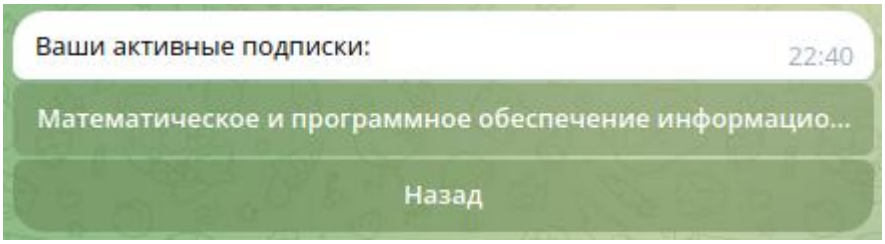


Рисунок 3.38 – Окно со всеми подписками пользователя (Разработка программного обеспечения удалена)

3.4 Тестирование

Разработка приложения велась с соблюдением чистой архитектуры [18], 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.

Далее пример юнит-теста, который проверяет работоспособность

GetScheduledNotificationsBySubscriptionUseCase:

```
NotificationRepo = Annotated[ScheduledNotificationRepository, AsyncMock]
EventRepo = Annotated[TimelineEventRepository, AsyncMock]
NameRepo = Annotated[TimelineEventNameRepository, AsyncMock]

@pytest.fixture
def notification_repo() -> NotificationRepo:
    return AsyncMock(spec=ScheduledNotificationRepository)


@pytest.fixture
def event_repo() -> EventRepo:
    return AsyncMock(spec=TimelineEventRepository)


@pytest.fixture
def name_repo() -> NameRepo:
    return AsyncMock(spec=TimelineEventNameRepository)


@pytest.fixture
def usecase(
    notification_repo: NotificationRepo,
    event_repo: EventRepo,
    name_repo: NameRepo,
) -> GetScheduledNotificationsBySubscriptionUseCase:
    return GetScheduledNotificationsBySubscriptionUseCase(
        notification_repo, event_repo, name_repo
    )

@pytest.mark.asyncio
async def test_returns_dto_for_single_notification(
    usecase, notification_repo, event_repo, name_repo
):
    # Setup test data
    notification = ScheduledNotificationDomain(
        id=1,
        subscription_id=1,
        event_id=10,
        send_at=datetime.today()+ timedelta(days=1),
    )
    event = TimelineEventDomain(
        id=10,
        name_id=100,
        binding_id=-1,
        deadline=date.today() + timedelta(days=2),
    )
    name = TimelineEventNameDomain(id=100, name="Подача документов")

    # Setup mocks
    notification_repo.filter.return_value = [notification]
    event_repo.get_many.return_value = [event]
    name_repo.get_many.return_value = [name]

    # Run usecase
    result = await usecase(subscription_id=1)

    # Assertions
    assert len(result) == 1
    dto = result[0]
    assert dto.id == notification.id
    assert dto.event_name == name.name
    assert dto.send_at == notification.send_at
```

Далее пример интеграционного теста, который проверяет взаимодействия базы данных и базового репозитория:

```
@pytest.mark.asyncio
async def test_add_and_get(session_with_drop_after: AsyncSession):
    repo = BaseRepository[CategoryDomain, Category, CreateCategoryDomain](
        session_with_drop_after, CategoryDomain, Category,
CreateCategoryDomain
    )
    category_domain = CreateCategoryDomain(title="Alice", parent_id=None)

    added = await repo.add(category_domain)
    assert added.title == category_domain.title

    fetched = await repo.get(1)
    assert fetched.title if fetched else '' == category_domain.title
```

После каждого подобного теста все данные в СУБД сбрасываются для изоляции зависимостей. Это делается при помощи фикстуры ниже:

```
@pytest.fixture
async def session_with_drop_after():
    async with async_engine.begin() as conn:
        await conn.run_sync(Base.metadata.drop_all)
        await conn.run_sync(Base.metadata.create_all)

    async with async_session() as session:
        yield session

    async with async_engine.begin() as conn:
        await conn.run_sync(Base.metadata.drop_all)
        await conn.run_sync(Base.metadata.create_all)
```

Далее пример нагрузочного теста, который проверяет соблюдения ограничений скорости отправки сообщений телеграм ботом:

```
token = "1234567890:TEST_TOKEN_FAKE1234567890abcdef"


@pytest.mark.asyncio
async def test_rate_limited_bot_does_not_exceed_global_limit():
    # Устанавливаем лимит: не больше 30 сообщений в секунду
    global_limit = 30
    message_count = 60  # Попробуем отправить 60 сообщений
    chat_id = 123456

    bot = RateLimitedBot(
        token=token,
        redis_url=test_redis_settings.get_async_connection_string(),
        global_rate=global_limit,
        global_per=1.0,
        chat_rate=1000,  # не ограничиваем по chat_id, только global
        chat_per=1.0,
    )

    # Мокаем настоящий send_message
    with patch.object(
        Bot, "send_message", new=AsyncMock(return_value="Mocked Telegram
Message")
    ):
```

```
        start = time.perf_counter()

        # Отправляем сообщения параллельно
        await asyncio.gather(
            *[
                bot.send_message(chat_id=chat_id, text=f"Test {i}")
                for i in range(message_count)
            ]
        )

        end = time.perf_counter()
        elapsed = end - start

        # Бот не мог отправить быстрее, чем message_count / global_limit
        expected_min_time = (message_count - global_limit) / global_limit

        print(f"Elapsed: {elapsed:.2f} sec, Expected ≥
{expected_min_time:.2f} sec")
        assert elapsed >= expected_min_time - 0.2  # небольшой допуск на
погрешность
```

Тесты находятся в директории /tests репозитория, QR-код на который находится в приложении Б.

На рисунке 3.39 демострация процесса выполнения тестов в github actions.

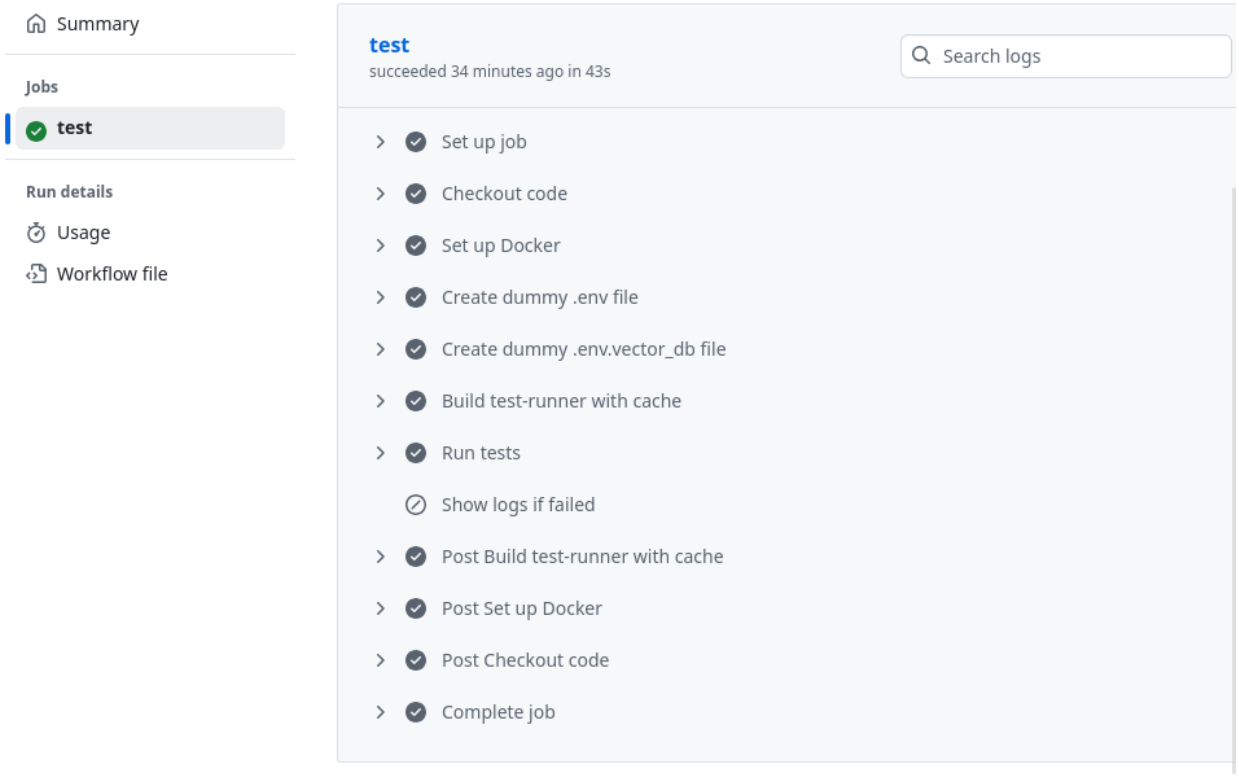


Рисунок 3.39 – Демонстрация процесса выполнения тестов в github actions

3.5 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:

- 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.

3.6 Итог разработки чат-бота в поддержку абитуриентов ВятГУ

В рамках данной главы реализован Telegram-бот, направленный на информационную поддержку абитуриентов ВятГУ. На основе ранее сформулированных требований была построена архитектура системы и определены ключевые сценарии взаимодействия с пользователем.

Бот успешно реализует три основные функции:

- рекомендации образовательных программ в соответствии с интересами пользователя;
- поиск ответов на вопросы по базе знаний;
- отправку уведомлений о предстоящих событиях.

Проведённое тестирование показало стабильную работу и соответствие реализованного функционала поставленным задачам. Также были обозначены направления для дальнейшего развития системы.

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, отправлять уведомления о событиях программы, 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.

Приложение Б содержит QR-коды на бота и на исходный код проекта. Краткое руководство эксплуатации представлено в приложении В и в README.md репозитория github.

Библиографический список

1. Aiogram [Electronic resource] / Aiogram contributors. URL: https://github.com/aiogram/aiogram (дата обращения: 26.05.2025).

2. ARQ: официальная документация [Electronic resource]. URL: https://arq-docs.helpmanual.io/ (дата обращения: 16.06.2025).

3. C# – .NET Languages [Electronic resource]. URL: https://dotnet.microsoft.com/en-us/languages/csharp (дата обращения: 26.05.2025).

4. Celery: официальный репозиторий [Electronic resource] / GitHub, The Celery Project. URL: https://github.com/celery/celery (дата обращения: 16.06.2025).

5. Clean Architecture: A Little Introduction [Electronic resource] // Medium. URL: https://medium.com/swlh/clean-architecture-a-little-introduction-be3eac94c5d1 (дата обращения: 26.05.2025).

6. cointegrated / LaBSE-en-ru [Electronic resource] / Hugging Face. URL: https://huggingface.co/cointegrated/LaBSE-en-ru (дата обращения: 26.05.2025).

7. cointegrated / rubert-tiny2 [Electronic resource] / Hugging Face. URL: https://huggingface.co/cointegrated/rubert-tiny2 (дата обращения: 26.05.2025).

8. DB-Engines Ranking [Electronic resource]. URL: https://db-engines.com/en/ranking (дата обращения: 26.05.2025).

9. DB-Engines Ranking: Vector DBMS [Electronic resource]. URL: https://db-engines.com/en/ranking/vector+dbms (дата обращения: 26.05.2025).

10. De Winter J. C. F., Gosling S. D., Potter J. Comparing the Pearson and Spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data [Text] //Psychological methods. – 2016. – Т. 21. – №. 3. – С. 273.

11. Docker: официальный сайт [Electronic resource]. URL: https://www.docker.com/ (дата обращения: 26.05.2025).

12. DragonflyDB [Electronic resource]. URL:

https://github.com/dragonflydb/dragonfly (дата обращения: 26.05.2025).

13. Encodechka: официальный репозиторий [Electronic resource]. URL: https://github.com/avidale/encodechka (дата обращения: 17.06.2025).

14. FAISS [Electronic resource] / Facebook AI Research. URL: https://github.com/facebookresearch/faiss (дата обращения: 26.05.2025).

15. GitHub Octoverse 2024 [Electronic resource]. URL: https://github.blog/news-insights/octoverse/octoverse-2024/ (дата обращения: 26.05.2025).

16. Java: официальный сайт [Electronic resource]. URL: https://www.java.com/ru/ (дата обращения: 26.05.2025).

17. KPFU Admissions Bot [Электронный ресурс]. URL: https://t.me/kpfu_admissions_bot (дата обращения: 26.05.2025).

18. Martin R. C. Clean architecture [Electronic resource].

19. Memcached [Electronic resource]. URL: https://github.com/memcached/memcached (дата обращения: 26.05.2025).

20. Milvus [Electronic resource]. URL: https://github.com/milvus-io/milvus (дата обращения: 26.05.2025).

21. Moscow Polytech Bot [Электронный ресурс]. URL: https://t.me/MoscowPolytechBot (дата обращения: 26.05.2025).

22. MTProto vs Bot API [Electronic resource] // Pyrogram Docs. URL: https://docs.pyrogram.org/topics/mtproto-vs-botapi (дата обращения: 26.05.2025).

23. MySQL Server [Electronic resource]. URL: https://github.com/mysql/mysql-server (дата обращения: 26.05.2025).

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

25. Pinecone [Electronic resource]. URL: https://www.pinecone.io/ (дата обращения: 26.05.2025).

26. PostgreSQL [Electronic resource]. URL: https://github.com/postgres/postgres (дата обращения: 26.05.2025).

27. Push technology – Wikipedia [Electronic resource]. URL:

https://en.wikipedia.org/wiki/Push_technology#Long_polling (дата обращения: 26.05.2025).

28. PyPI Stats [Electronic resource]. URL: https://pypistats.org/ (дата обращения: 26.05.2025).

29. Pyrogram [Electronic resource]. URL: https://github.com/pyrogram/pyrogram (дата обращения: 26.05.2025).

30. Python Software Foundation: официальный сайт [Electronic resource]. URL: https://www.python.org/ (дата обращения: 26.05.2025).

31. Python Telegram Bot [Electronic resource] / python-telegram-bot contributes. – URL: https://github.com/python-telegram-bot/python-telegram-bot (дата обращения: 26.05.2025).

32. Qdrant [Electronic resource]. URL: https://github.com/qdrant/qdrant (дата обращения: 26.05.2025).

33. RabbitMQ Server: официальный репозиторий [Electronic resource]. URL: https://github.com/rabbitmq/rabbitmq-server (дата обращения: 16.06.2025).

34. Redis [Electronic resource]. URL: https://github.com/redis/redis (дата обращения: 26.05.2025).

35. sergeyzh / LaBSE-ru-sts [Electronic resource] // Hugging Face. URL: https://huggingface.co/sergeyzh/LaBSE-ru-sts (дата обращения: 26.05.2025).

36. sergeyzh / LaBSE-ru-turbo [Electronic resource] // Hugging Face. URL: https://huggingface.co/sergeyzh/LaBSE-ru-turbo (дата обращения: 26.05.2025).

37. sergeyzh / rubert-mini-sts [Electronic resource] // Hugging Face. URL: https://huggingface.co/sergeyzh/rubert-mini-sts (дата обращения: 26.05.2025).

38. sergeyzh / rubert-tiny-sts [Electronic resource] // Hugging Face. URL: https://huggingface.co/sergeyzh/rubert-tiny-sts (дата обращения: 26.05.2025).

39. SPbPU Bot [Электронный ресурс]. URL: https://t.me/SPbPUbot (дата обращения: 26.05.2025).

40. SQLite [Electronic resource]. URL: https://github.com/sqlite/sqlite (дата обращения: 26.05.2025).

41. Stack Overflow Developer Survey 2024 [Electronic resource]. URL: https://survey.stackoverflow.co/2024/ (дата обращения: 26.05.2025).

42. Telegram API: messages.editMessage [Electronic resource] / Telegram Messenger Inc.. URL: https://core.telegram.org/method/messages.editMessage (дата обращения: 26.05.2025).

43. Telegram Bot API [Electronic resource]. URL: https://core.telegram.org/bots/api (дата обращения: 26.05.2025).

44. Telegram Bot API FAQ [Electronic resource]. URL: https://core.telegram.org/bots/faq#my-bot-is-hitting-limits-how-do-i-avoid-this (дата обращения: 26.05.2025).

45. Telegram Bot API: Paid Broadcasts [Electronic resource]. URL: https://core.telegram.org/bots/api#paid-broadcasts (дата обращения: 26.05.2025).

46. Telegram Web Apps [Electronic resource]. URL: https://core.telegram.org/bots/webapps (дата обращения: 26.05.2025).

47. Telegram Webhooks [Electronic resource]. URL: https://core.telegram.org/bots/webhooks (дата обращения: 26.05.2025).

48. Telegram обогнал YouTube по ежедневной аудитории в России [Электронный ресурс] // Adindex. 2024. 13 нояб. URL: https://adindex.ru/news/digital/2024/11/13/327316.phtml (дата обращения: 26.05.2025).

49. TIOBE Index [Electronic resource]. URL: https://www.tiobe.com/tiobe-index/ (дата обращения: 26.05.2025).

50. Tochka-AI / ruRoPEBert-e5-base-512 [Electronic resource] // Hugging Face. URL: https://huggingface.co/Tochka-AI/ruRoPEBert-e5-base-512 (дата обращения: 26.05.2025).

51. Weaviate [Electronic resource]. URL: https://github.com/weaviate/weaviate (дата обращения: 26.05.2025).

52. What is NuGet [Electronic resource] // Microsoft Learn. URL: https://learn.microsoft.com/en-us/nuget/what-is-nuget (дата обращения:

26.05.2025).

53. Более 60 млн россиян ежедневно пользуются Telegram [Электронный ресурс] // Ведомости. 2024. 11 июля. URL: https://www.vedomosti.ru/technology/articles/2024/07/11/1049269-bolee-60-mln-rossiyan-ezhednevno-polzuyutsya-telegram (дата обращения: 26.05.2025).

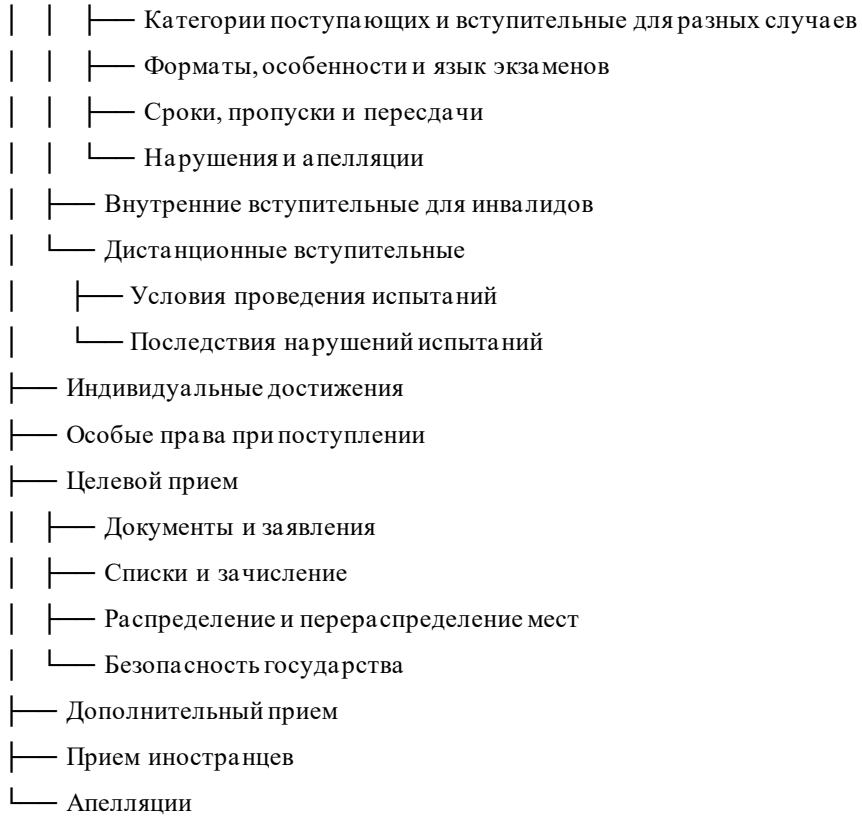
54. Вятский государственный университет. Правила приёма в ФГБОУ ВО «Вятский государственный университет» на 2025/2026 учебный год: бакалавриат, специалитет, магистратура [Электронный ресурс]. URL: https://new.vyatsu.ru/admission/admission_info/rules-of-admission/pravila-priema-v-fgbou-vo-vyatskiy-gosudarstvennyy-universitet-bakalavr-magistr-2025.php (дата обращения: 16.06.2025).

55. Вятский государственный университет: официальный сайт [Electronic resource]. URL: https://new.vyatsu.ru/ (дата обращения: 26.05.2025).

Приложения

Приложение А

├── Часто задаваемые вопросы
│ ├── Подача документов и поступление
│ │ ├── Способы подачи документов
│ │ ├── Общие вопросы
│ │ ├── Перевод и восстановление
│ │ ├── ЕГЭ и вступительные испытания
│ │ └── Целевой прием и платное обучение
│ ├── Общежитие
│ └── Направление «Педагогическое образование (с двумя профилями подготовки)»
│ ├── Поступление
│ ├── Выбор траекторий и профилей
│ ├── Содержание и организация обучения
│ ├── Управление учебной программой
│ └── Форматы и детали учебного процесса
├── Общие положения
│ ├── Общая информация о правилах приема
│ ├── Документы, условия приема и квоты
│ └── Конкурсная процедура и взаимодействие с ВУЗом
├── Общие правила подачи и поступления
│ ├── Одновременное поступление в организации
│ ├── Подача документов
│ │ ├── Подача заявления и документов
│ │ ├── Сроки и изменение заявления
│ │ ├── Конкурсные группы и приоритеты
│ │ └── Особые условия и подтверждение намерений
│ ├── Списки поступающих
│ ├── Зачисление, согласие и договор
│ │ ├── Порядок и условия зачисления
│ │ ├── Согласие на зачисление
│ │ ├── Зачисление на платные места (договор)
│ │ └── Отзыв заявления и отказ
│ ├── Зачисление на бюджет
│ │ ├── Этапы зачисления и приоритеты
│ │ └── Перераспределение и участие в нескольких этапах
│ ├── Информирование о приеме
│ └── Сроки приема
├── Вступительные испытания
│ ├── Перечень и формы вступительных
│ │ ├── Формы поступления и выбор предметов
│ │ ├── Оценивание, приоритет и конкуренция


Категории поступающих и вступительные для разных случаев
Форматы, особенности и язык экзаменов
Сроки, пропуски и пересдачи
Нарушения и апелляции
Внутренние вступительные для инвалидов
Дистанционные вступительные
Условия проведения испытаний
Последствия нарушений испытаний
Индивидуальные достижения
Особые права при поступлении
Целевой прием
Документы и заявления
Списки и зачисление
Распределение и перераспределение мест
Безопасность государства
Дополнительный прием
Прием иностранцев
Апелляции

Приложение Б

QR-коды на исходный код проекта и на бота

Рисунок 1 – QR-код на бота

Рисунок 2 – QR-код на исходный код проекта

Приложение В

Руководство эксплуатации бота

1. Установка

1.1 Установка необходимых компонентов

Для корректной работы программного комплекса необходимо установить:

- docker;
- docker-compose.

1.2 Клонирование репозитория

Склонируйте репозиторий в локальную директорию, выполнив следующие команды:

- git clone https://github.com/DimaOshchepkov/vyatsu_applicant_bot.git;
- cd vyatsu_applicant_bot.

1.3 Настройка конфигурационных файлов

Создайте два файла: .env и .env.vector_db в корневой директории проекта.

Пример содержимого файла .env:

API_TOKEN=<токен Telegram-бота>

POSTGRES_USER=<пользователь PostgreSQL>

POSTGRES_PASSWORD=<пароль>

POSTGRES_DB=<имя базы данных>

DB_HOST=db

DB_PORT=5432

DB_NAME=<имя базы данных>

DB_USER=<пользователь>

DB_PASS=<пароль>

THRESHOLD=70

REDIS_HOST=bot_redis

REDIS_PORT=6379

Параметр THRESHOLD указывает пороговое значение семантической схожести для распознавания программ и экзаменов.

Пример содержимого файла .env.vector_db:

QDRANT_QUESTION_COLLECTION=<название коллекции вопросов>

QDRANT_PROGRAM_COLLECTION=<название коллекции программ>

QDRANT_HOST_NAME=qdrant

QDRANT_PORT=6333

EMBEDDED_MODEL=<имя модели, например: sergeyzh/rubert-mini-sts>

HUB_EMBEDDED_MODEL=./src/vector_db_service/sergeyzh_rubert-mini-sts

EMBEDDED_SIZE=312

VECTOR_DB_SERVICE_CONTAINER_NAME=vector_db_service

VECTOR_DB_SERVICE_PORT=8000

1.4 Подготовка модели эмбеддингов

Модель эмбеддингов необходимо загрузить в директорию vector_db_service. Пример структуры:

```
vector_db_service/
└── sergeyzh_rubert-mini-sts/
    ├── config.json
    ├── pytorch_model.bin
    └── ...
```

1.5 Сборка и запуск проекта

Для сборки и запуска всех компонентов выполните команду docker-compose up --build.

При первом запуске контейнер migration выполнит миграции и загрузит данные в базу из файлов, размещённых в src/tactic/infrastructure/db/migrations/upload_data.

Для остановки и удаления контейнеров выполните команду docker-compose down -v.

2. Архитектура системы

Программный комплекс построен по принципам чистой архитектуры (Clean Architecture). Логика разделена на слои, каждый из которых имеет свою

область ответственности.

2.1 Слой domain

Расположен в папке src/tactic/domain.

Содержит:

- бизнес-сущности (см. src/tactic/domain/entities);
- value objects.

Данный слой не зависит от внешних библиотек и реализует предметную область.

2.2 Слой application

Расположен в папке src/tactic/application.

Содержит:

- интерфейсы для взаимодействия с внешними слоями (common/repositories.py, services/);
- сценарии использования (use_cases/), отвечающие за координацию бизнес-логики.

2.3 Слой presentation

Расположен в src/tactic/presentation.

Отвечает за взаимодействие с внешними системами. Включает в себя:

- точку входа в приложение – bot.py;
- обработку пользовательских сценариев – telegram/;
- внедрение зависимостей – interactor_factory.py, ioc.py.

2.4 Слой infrastructure

Расположен в src/tactic/infrastructure.

Отвечает за реализацию внешних зависимостей. Содержит следующие компоненты:

- репозитории (repositories/);
- сервисы (recognize_*.py, notificaton_message_sheduling_service.py, telegram_message_sender.py, rate_limited_bot.py);
- мидлвари для ограничения спама (middlewares/antiflood_middlewares.py).

3. Контейнер vector_db_service

Контейнер vector_db_service реализует HTTP API (на базе FastAPI) для доступа к векторной базе данных Qdrant. Сервис работает на CPU и предоставляется другим сервисам через внутреннюю сеть.

Ключевые компоненты:

- API-интерфейс – src/vector_db_service/app/api.py;
- скрипт проверки состояния – src/vector_db_service/healthcheck.sh;
- загрузка данных о программах – src/vector_db_service/app/load_program_collection.py;
- загрузка вопросов – src/vector_db_service/app/load_question_collection.py.

4. Контейнер arq (фоновая обработка)

Контейнер arq отвечает за выполнение фоновых задач (например, отложенных уведомлений). Это позволяет не блокировать основной поток исполнения Telegram-бота.

Основные модули:

- воркер – src/tactic/presentation/notification/worker.py;
- задача отправки сообщений – src/tactic/presentation/notification/send_delayed_message.py.

5. Контейнер migrations

Контейнер migrations предназначен для:

- выполнения миграций базы данных (на базе Alembic);
- загрузки подготовленных данных из JSON-файлов (src/tactic/infrastructure/db/migrations/upload_data).

6. Дополнительная информация

Файл конфигурации контейнеров – docker-compose.yml.

ORM-модели вынесены в src/shared/models.py.

Тесты размещены в src/tests. Они запускаются в отдельном контейнере test, использующем тестовую базу test-db для обеспечения полной изоляции и поддержки CI/CD.