МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

Цифровая трансформация бизнеса
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка чат-бота с применением масштабируемой облачной ИТ-инфраструктуры, DevOps, экосистемы k8s, микросервисов, бессерверных вычислений и ИИ-технологий

Обучающийся П.А. Донской
(Инициалы Фамилия) (личная подпись)

Руководитель канд.пед.наук, А.В. Богданова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Тема выпускной квалификационной работы: «Разработка чат-бота с применением масштабируемой облачной IT-инфраструктуры, DevOps, экосистемы k8s, микросервисов, бессерверных вычислений и ИИ-технологий».

Целью данной работы является проведение исследования и реализация программного обеспечения для автоматизации процесса обработки запросов студентов к сотрудникам кафедры.

Актуальность работы обусловлена потребностью в автоматизации рабочего процесса по сбору, хранению и обработке запросов от студентов университета.

По итогу бакалаврской работы была разработана система на базе микросервисного подхода со следующим функционалом:

- взаимодействие с Telegram API для приёма и отправки сообщений;
- сохранение и обработка запросов студентов;
- обработка команд администраторов;
- интеграция с облачными LLM провайдерами.

Пояснительная записка содержит введение, три главы, заключение и список используемой литературы

Abstract

The topic of the graduate qualification work is «Development of a chatbot using scalable cloud IT infrastructure, DevOps, k8s ecosystem, microservices, serverless computing, and AI technologies».

The purpose of this work is to conduct research and implement software for automating the process of processing student requests to department employees.

The relevance of the work is due to the need to automate the workflow for collecting, storing, and processing requests from university students.

As a result of the bachelor's work, a system was developed based on a microservice approach with the following functionality:

– interaction with the Telegram API for receiving and sending messages;
– saving and processing student requests;
– processing administrator commands;
– integration with cloud LLM providers.

The explanatory note contains an introduction, three chapters, a conclusion, and a list of used literature.

Оглавление

Введение

Цель данной работы обусловлена темой «Разработка чат-бота с применением масштабируемой облачной ИТ-инфраструктуры, DevOps, экосистемы k8s, микросервисов, бессерверных вычислений и ИИ-технологий», и заключается в проведении исследования и дальнейшей реализации программного обеспечения (сокр. ПО) для автоматизации процесса обработки запросов, адресованных к сотрудникам кафедры прикладной математики и информатики (сокр. ПМИФ) Тольяттинского государственного университета.

Для достижения этой цели предстоит выполнить следующий перечень задач: провести анализ организации и связанной с ней предметной области; объяснить, чем обусловлена необходимость внедрения автоматизированной системы; проанализировать бизнес-процессы до и после внедрения предлагаемой системы автоматизации; спроектировать и разработать ПО в соответствии с потребностями организации; провести оценку экономической эффективности проекта. В ходе данной работы я планирую освоить следующие компетенции: обработка и анализ материалов, полученных в ходе исследования; проектирование и разработка масштабируемого программного обеспечения с использованием ИИ-технологий, микросервисной архитектуры и методологии DevOps; анализ затрат на реализацию и эксплуатацию проекта, расчёт экономической эффективности проекта; оформление отчетной документации, отражающей ход исследования и проводимых работ. Актуальность проекта будет наглядно продемонстрирована в ходе дальнейшего исследования, направленного на автоматизацию бизнес-процессов кафедры посредством внедрения чат-бота для обработки заявок студентов. Результатом работы, полученным по её завершении, будет отчёт, описывающий проведённое исследование и программный продукт, разработанный на основе этого анализа.

Глава 1 Анализ предметной области

1.1 Характеристика деятельности предприятия

Кафедра ПМИФ, являющаяся объектом исследования этой работы, входит в состав Тольяттинского государственного университета. Рассматриваемая организация ведёт образовательную и исследовательскую деятельность в области информационных технологий и прикладной математики, включающую в себя обучение студентов и проведение исследовательской работы с применением современных информационных технологий. Для лучшего понимания задач и потребностей кафедры, составим структурную схему организации, которая наглядно продемонстрирует иерархию должностей и их взаимоотношение (см. рисунок 1).

Рисунок 1 – Организационная структура предприятия

В рамках данной работы значительное внимание будет уделено взаимодействию учебно-вспомогательного и преподавательского состава со студентами университета. Именно этот процесс и будет рассмотрен в следующем подразделе.

1.2 Построение моделей предметной области

Сопровождение учебного процесса и практической деятельности студентов является одной из ключевых задач организации. В связи с этим стоит рассмотреть, как именно устроен процесс взаимодействия сотрудников организации со студентами. Начнём с создания AS-IS диаграммы, применив нотацию BPMN [4] (см. рисунок 2).

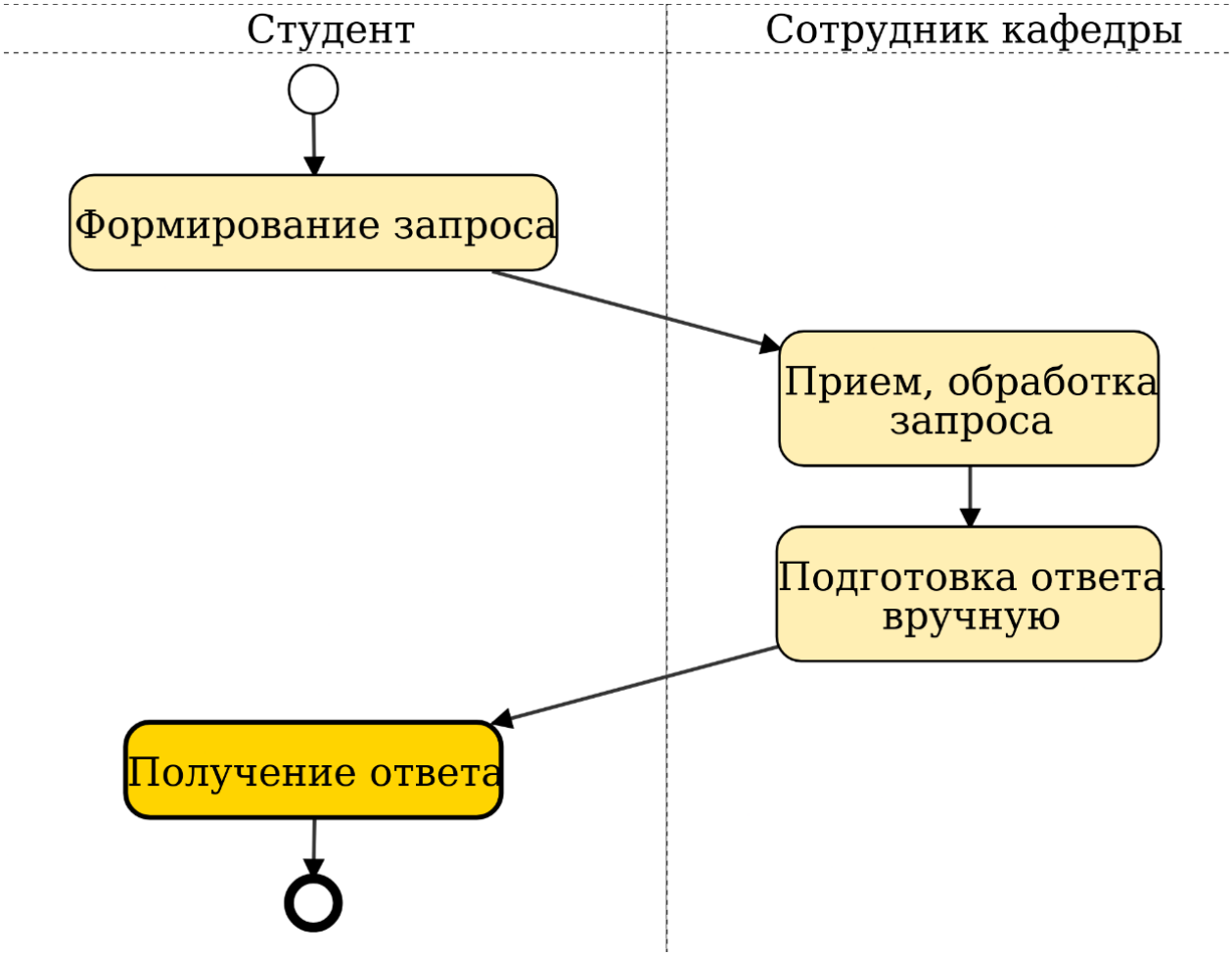

Рисунок 2 – AS-IS BPMN диаграмма процесса обработки запроса студента

Представленная диаграмма процесса по обработке запросов студентов демонстрирует промежуточные этапы до внедрения системы автоматизации. Это включает в себя начальное формирование запроса, его дальнейшую

обработку работником кафедры и получение подготовленного ответа студентом.

Для реинжиниринга рассматриваемого бизнес-процесса была создана TO-BE диаграмма, отображающая процесс обработки заявок после добавления средства автоматизации в виде чат-бота (см. рисунок 3).

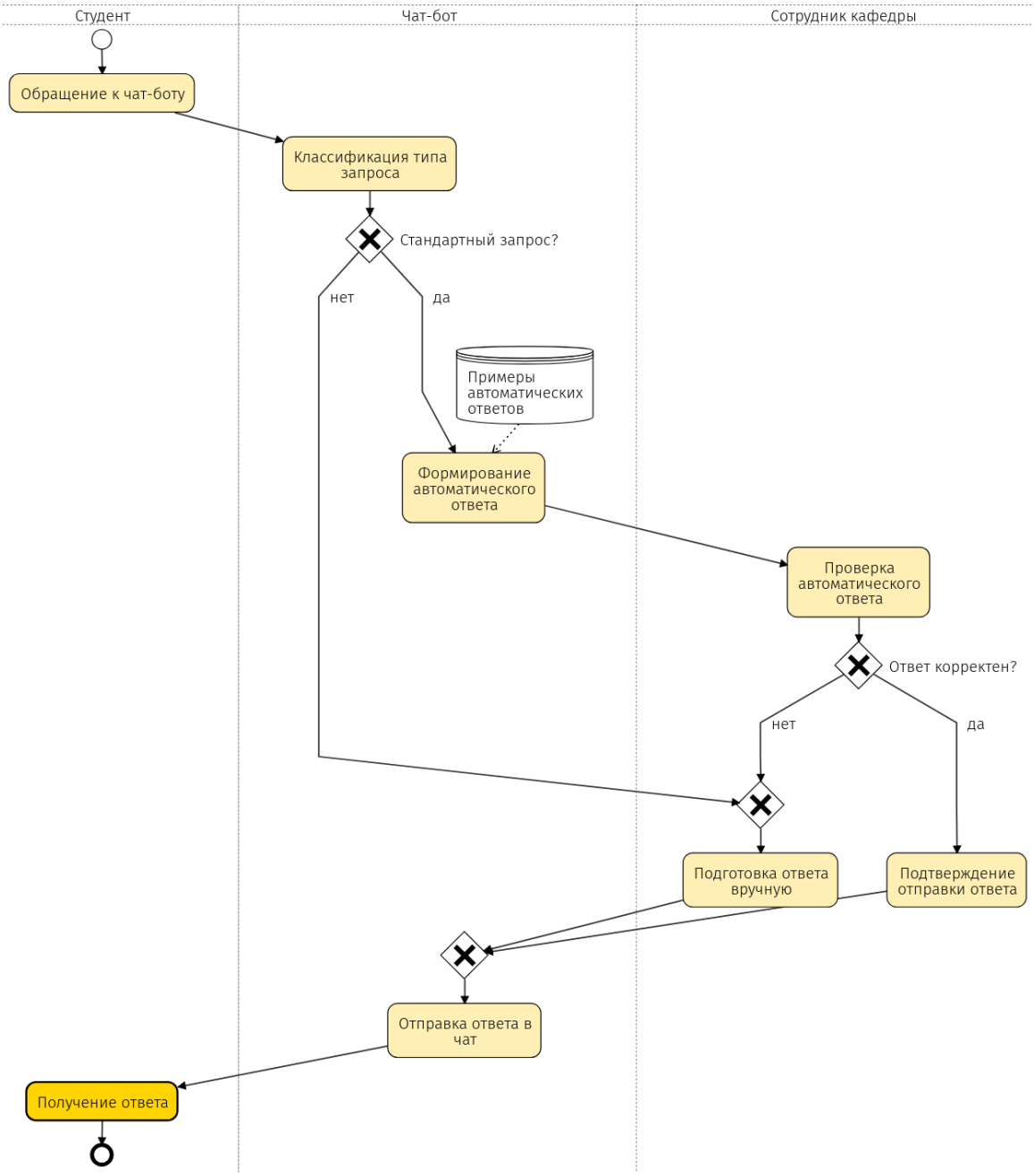

Рисунок 3 – TO-BE BPMN диаграмма процесса обработки запроса студента

Процесс обработки заявок информационной системой начинается с классификации типа запроса и последующей генерации ответа посредством больших языковых моделей (англ. LLM). Если же пользовательский запрос является нетривиальным, сотрудник может вручную составить ответ и переслать его при помощи чат-бота. Для предотвращения ошибок или неточностей, каждый автоматически сгенерированный ответ проходит верификацию работником кафедры.

Теперь перейдём к анализу лучших практик, используемых для автоматизации деятельности организации. В контексте информационных систем, микросервисная архитектура является одним из самых популярных решений для создания масштабируемых приложений, позволяя разбивать сложные системы на независимые модули и изменять объём ресурсов (реплик сервисов) в зависимости от нагрузки [19]. Также, совместно с этой практикой может быть использована методология DevOps, позволяющая объединить две важные функции: разработку ПО (development) и его эксплуатацию (operations), упрощая процесс выпуска и размещения обновлений [12]. Для контейнеризации (создания изолированной среды, в которой происходит компиляция или запуск модулей ПО) может применяться система с открытым исходным кодом Kubernetes (k8s). Помимо основного функционала в виде создания и управления отдельными контейнерами, экосистема инструментов k8s также значительно упрощает управление распределенными системами, что особенно актуально для этого проекта. Также в ходе разработки возможно применение бессерверных вычислений – это ещё один подход к созданию информационных систем, подразумевающий использование серверной инфраструктуры, управление и масштабирование которой полностью берет на себя облачный провайдер. В этом случае необходимые вычисления запускаются в облаке и автоматически масштабируются в зависимости от нагрузки.

1.3 Анализ существующих разработок

В качестве примера существующих разработок для решения обозначенной задачи, можно выделить комплект ПО Microsoft Bot Framework и платформу Dialogflow от компании Google. Начнём с того, что оба продукта весьма востребованы – они применяются в различных сферах, таких как обслуживание клиентов, образование, здравоохранение и так далее. Исходный код Microsoft Bot Framework предоставлен в открытом доступе и активно поддерживается энтузиастами и сотрудниками компании Microsoft [20]. Dialogflow является закрытой платформой, и вместо этого предоставляет доступ к своим инструментам в качестве платного сервиса [2]. Оба решения обладают широким набором инструментов и интеграций с продуктами компании Microsoft или Google, но также имеют свои ограничения и требуют значительных усилий для настройки, размещения и обеспечения совместимости с остальными компонентами информационной системы университета.

1.4 Разработка требований к информационной системе

Отталкиваясь от проведённого анализа, можно составить следующий перечень функциональных требований к представленному решению по автоматизации процесса обработки запросов:

– возможность взаимодействия между студентами и сотрудниками кафедры университета посредством чат-бота;

– классификация типов пользовательских запросов;

– формирование автоматических ответов на стандартные запросы;

– перенаправление нестандартных запросов сотрудникам кафедры с последующей отправкой подготовленного ответа студенту.

Бизнес-цели проекта заключаются в:

– снижении рабочей нагрузки на сотрудников кафедры;

– ускорении процесса обработки запросов студентов, повышении качества обслуживания.

Требования к ИТ-проекту включают в себя:

– разработку масштабируемой и отказоустойчивой системы;

– применение архитектуры, подходящей для дальнейшего расширения функционала;

– обеспечение бесперебойной и круглосуточной работы системы.

Результаты, собранные в ходе этого анализа лягут в основу дальнейшей разработки приложения и разработки перспективных направлений дальнейших исследований.

Глава 2 Проектирование и реализация проекта

2.1 Моделирование логической и физической архитектуры проекта

При реализации проекта было отдано предпочтение микросервисной архитектуре вместо классического монолитного проектирования систем. Данный выбор позволит разделять приложения на мелкие функциональные компоненты с собственным набором кода, программным интерфейсом приложения (англ. API), базой данных и прочими изолированными от внешней среды компонентами (см. рисунок 4). Благодаря этому подходу получаемая на выходе система обладает большей модульностью, масштабируемостью и отказоустойчивостью [12].

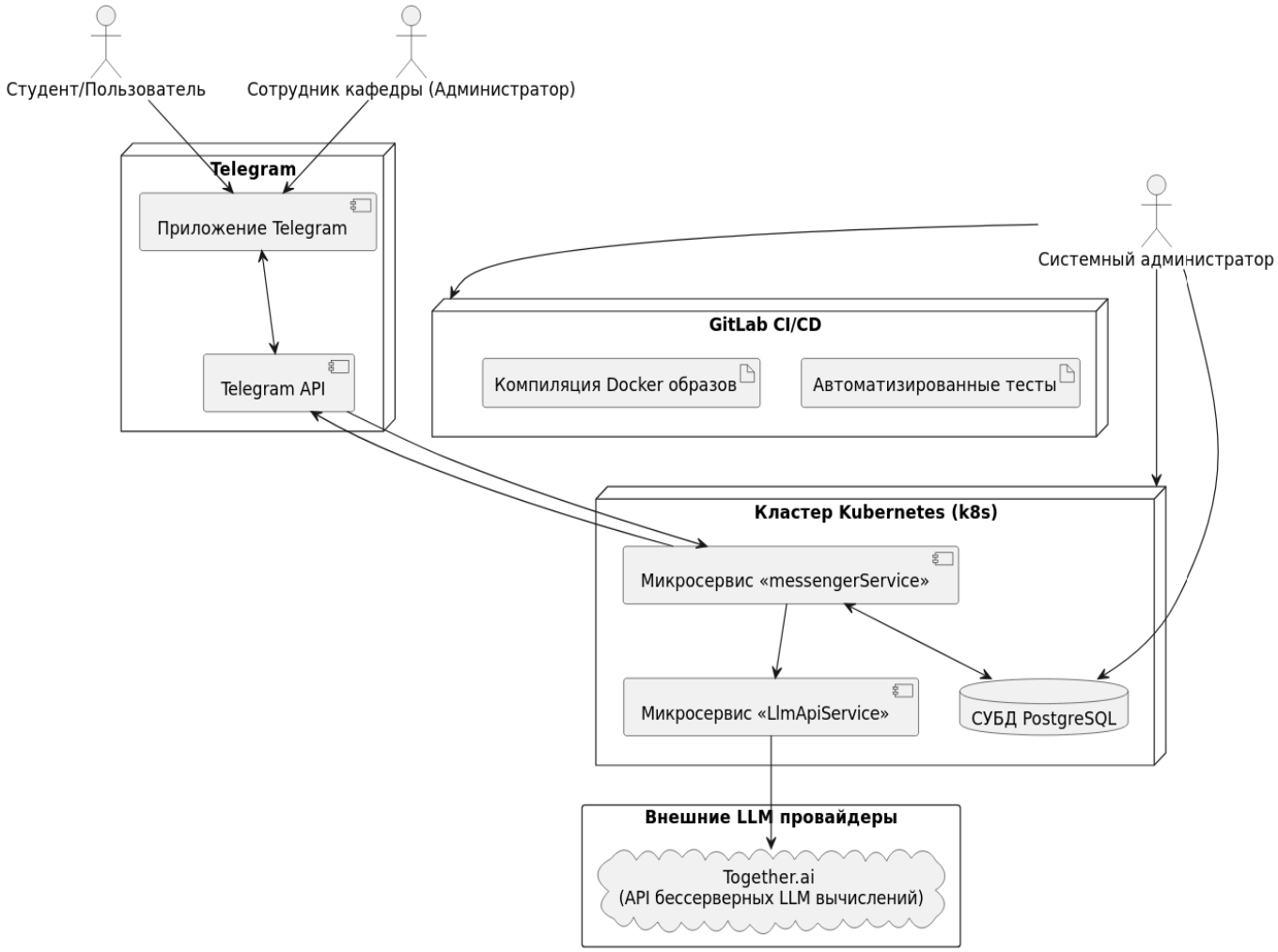

Рисунок 4 – Диаграмма размещения, демонстрирующая системную архитектуру проекта

В качестве основных компонентов, реализованных в рамках данного проекта выступают микросервисы «messengerService» и «llmService». Рассмотрим функционал сервиса «messengerService»:

– взаимодействие с мессенджером Telegram через API для приёма и обработки, отправки сообщений в чатах Telegram;

– обработка полученных команд от сотрудников кафедры с правами администраторов;

– взаимодействие с микросервисом «llmService» для автоматической генерации ответов на пользовательские запросы;

– чтение, запись, обновление и удаление данных в реляционной базе данных PostgreSQL.

Микросервис «llmService» предназначен для выполнения роли посредника между сервисом «messengerService» и облачными LLM провайдерами. Он не содержит сложной бизнес-логики, так как его основная задача заключается в абстрагировании прямого взаимодействия с внешними сервисами, осуществляющими генерацию текста при помощи нейросетей. Такая абстракция облегчает интеграцию и гарантирует, что микросервис «messengerService» не будет содержать лишней бизнес-логики, выполняя лишь ограниченный набор функций связанных с мессенджером, без необходимости реализовывать детали взаимодействия внешними LLM провайдерами.

Физическая архитектура системы построена на основе Kubernetes-кластера, который обеспечивает оркестрацию и управление контейнеризированными компонентами: микросервисами, базами данных, распределителями нагрузки и прочими элементами инфраструктуры, что позволяет масштабировать их независимо друг от друга [5]. Например, при увеличении числа запросов на генерацию текста к «llmService», можно горизонтально масштабировать экземпляры данного микросервиса, размещая их на одном или нескольких серверах (узлах) физической или виртуальной инфраструктуры, не затрагивая работу «messengerService». Нагрузочное тестирование системы на дальнейшем этапе покажет, что запуск всех контейнеров Kubernetes на одном физическом сервере подходит для начальных требований к пропускной способности приложения. Стоит учитывать, что в ходе эксплуатации системы также останется возможность запуска дополнительных контейнеров в кластере для распределения входящего трафика между доступными экземплярами отдельных логических компонентов.

Также для упрощения процесса выпуска и размещения обновлений посредством методологии DevOps используется система GitLab CI/CD. Она позволяет осуществлять сборку образов Docker-контейнеров, содержащих все необходимые для работы программные компоненты для дальнейшего размещения микросервисов. При каждой выгрузке изменений исходного кода в облачную платформу GitLab проводится автоматический запуск тестов, результаты выполнения которых сохраняются на серверах платформы [16] для корректной работы с версиями.

2.2 Разработка логической и физической модели данных

Отталкиваясь от проведённого анализа архитектуры системы, составим концептуальную схему данных приложения, воспользовавшись нотацией «сущность – связь» П. Чена [3] (см. рисунок 5).

Рисунок 5 – Диаграмма «сущность – связь»

На концептуальной схеме представлены три сущности, которые в дальнейшем будут перенесены в базу данных: Пользователи («USERS»), Сообщения «MESSAGES», Примеры ответов на сообщения («RESPONSE_EXAMPLES»). Как мы видим, пользователи могут создавать сообщения, а при наличии прав администратора – проверять запросы других пользователей перед отправкой ответов и сохранять примеры ответов на часто задаваемые вопросы. Также на основе этой модели была создана следующая подсхема таблиц базы данных (см. рисунок 6):

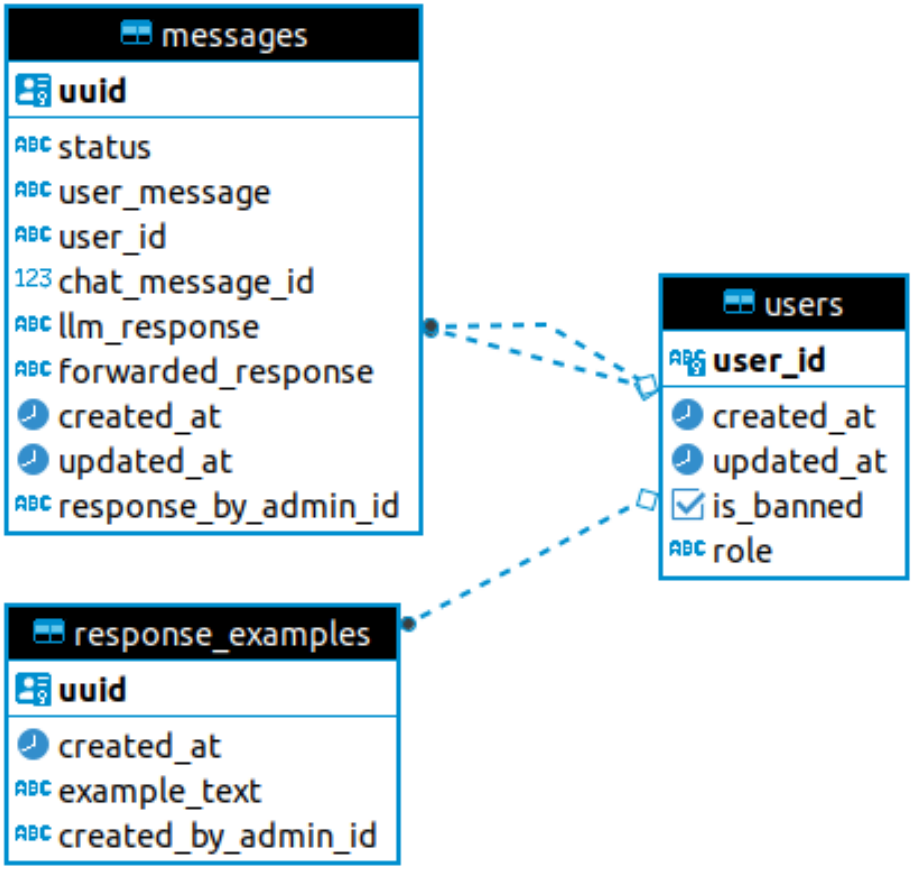

Рисунок 6 – Схема таблиц базы данных

В базе данных, используемой микросервисом «messengerService» также отражены три основные сущности – пользователи, сообщения (запросы студентов) и примеры ответов на сообщения. Сущность пользователей

содержит идентификатор «user_id», получаемый из мессенджера при получении первого сообщения. Столбец «role» может содержать значения «admin» или «user» в зависимости от уровня привелегий пользователя. При нарушении правил использования чат-бота, пользователя можно заблокировать, установив флаг «is_banned». Редактирование этих значений предусмотрено посредством ручной отправки запросов к БД системным администратором, но в дальнейшем данный функционал также может быть добавлен в чат-бот.

Теперь рассмотрим индексы и внешние ключи в таблицах «messages» и «response_examples». Данные, сохраняемые в остальных полях будут рассмотрены при анализе подсхемы программной архитектуры микросервиса «messengerService».

Таблица «messages» (запросы от пользователей) содержит несколько индексов:

– индекс «messages_pkey», оптимизирующий запросы по столбцу «uuid» (первичный ключ, содержит случайно сгенерированное значение размером в 16 байт, представляется в виде 36 текстовых символов [10]);

– индекс «idx_messages_user_id», оптимизирующий запросы по столбцу «user_id», который в свою очередь является внешним ключом, ссылающимся на таблицу «users»;

– индекс «idx_messages_chat_message_id» оптимизирует запросы по столбцу «chat_message_id». Значения идентификаторов, вносимых в БД предоставляются мессенджером Telegram, в виде целочисленных номеров сообщений в чате с отдельным пользователем;

– индекс «idx_messages_response_by_admin_id» оптимизирует запросы по столбцу «response_by_admin_id», который также является внешним ключом, связанным с таблицей «users». Данное значение устанавливается при отправке сотрудником с правами администратора ответа на пользовательский запрос.

В таблице «response_examples» имеется индекс «response_examples_pkey» для столбца «uuid», являющегося первичным ключом. Также в таблице имеется внешний ключ «created_by_admin_id», связанный с таблицей «users», значение которого заполняется при первом внесении записи.

Стоит отметить, что для записей в таблице примеров ответов на вопросы не предусмотрено изменение данных – для автоматической генерации ответов всегда берётся самая последняя запись посредством сортировки по столбцу «created_at» (с соответствующим индексом «idx_response_examples_created_at»). Для таблиц «messages» и «users», помимо даты создания также сохраняется временная метка внесения последнего изменения «updated_at».

Финальная модель данных приложения была спроектирована с учетом требований третьей формы нормализации (3NF) – все неключевые атрибуты в таблицах функционально зависят от всего первичного ключа, а дополнительные зависимости между неключевыми полями отсутствуют. Соблюдение данных требований позволит в дальнейшем минимизировать избыточность данных и обеспечить более эффективное хранение информации.

2.3 Разработка объектно-ориентированной архитектуры системы

Рассмотрим приложения в виде UML диаграммы классов микросервиса «messengerService» (см. рисунок 7):

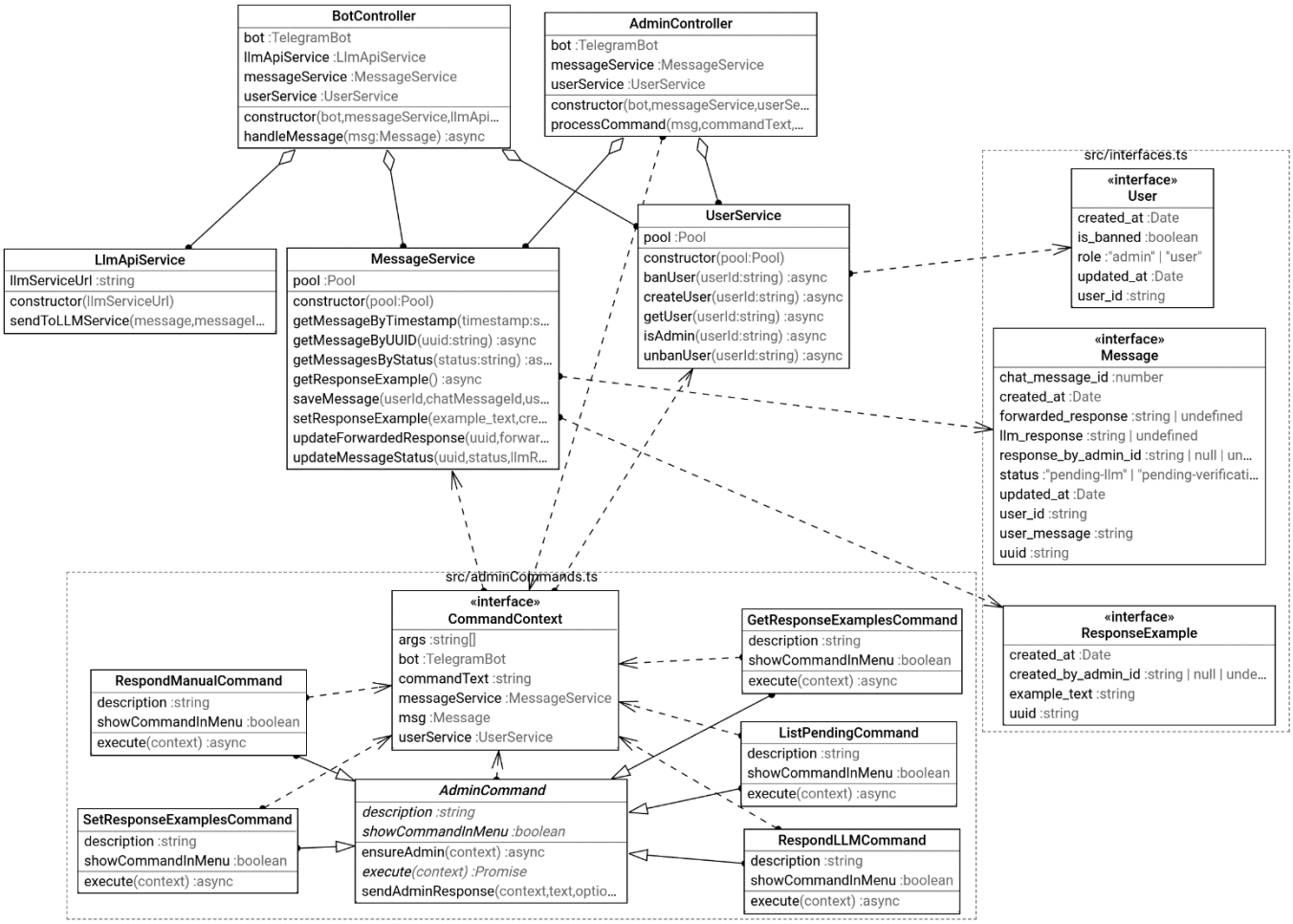

Рисунок 7 – UML диаграмма классов микросервиса «messengerService»

Как было ранее обозначено, главная задача этого микросервиса заключается в приёме, обработке, сохранении и отправке сообщений чат-ботом. Использование многослойной архитектуры при проектировании данного компонента подразумевает следующее разделение классов: контроллеры являются внешним слоем иерархии классов, принимающим запросы (входящие сообщения от Telegram API), а сервисы – внутренним слоем, реализующим основную бизнес-логику и взаимодействующим с внешними системами [7], а именно – базой данных и микросервисом «llmService». Взаимодействие между слоями происходит сверху вниз, в

данном случае – контроллеры взаимодействуют с сервисами, содержащими более низкоуровневую бизнес-логику по взаимодействию с базой данных и сервисом для генерации текста. Например, класс «BotController» обрабатывает пользовательские сообщения с запросами к кафедре, пересылая их в «LlmApiService» для генерации автоматического ответа, а затем сохраняя их посредством «MessageService». Классы из файла «adminCommands.ts» также можно рассматривать как часть внутреннего слоя бизнес-логики для обработки сообщений с командами администратора. Теперь более подробно рассмотрим иерархию этих классов. В качестве абстрактного родительского класса выступает «AdminCommand», от которого наследуются все его реализации, содержащие бизнес-логику, связанную с их выполнением [1]. За счёт этого все классы содержат поле с описанием и флаг «showCommandInMenu», что позволяет скрывать определенные команды из меню администратора. Функционал каждой из этих команд будет более подробно рассмотрен на этапе демонстрации работы приложения. В качестве примера реализации класса «AdminCommand» представлен листинг исходного кода класса «ListPendingCommand», пересылающего администратору список сообщений, ожидающих ответа (см. рисунок 8):

```
export class ListPendingCommand extends AdminCommand {
  description = "Список сообщений, ожидающих ответа";
  showCommandInMenu = true;

  async execute(context: CommandContext): Promise<void> {
    if (!(await this.ensureAdmin(context))) return;

    try {
      const messages = await context.messageService.getMessagesByStatus("pending-verification");
      if (messages.length === 0) {
        await this.sendAdminResponse(context, "Нет сообщений, ожидающих проверки.");
        return;
      }

      for (const message of messages) {
        try {
          await context.bot.forwardMessage(context.msg.chat.id, message.user_id, message
              .chat_message_id);

          if (message.llm_response) {
            const createdAtTimestamp = formatDateToString(message.created_at);
            let responseText = `ID Пользователя: ${message.user_id}`;
            responseText += `\nДата отправки: ${createdAtTimestamp}`;
            responseText +=
              `\nАвтоматически сгенерированный ответ:` + checkLLMResponseLengthValid(message
                  .llm_response)
                ? `<pre>${message.llm_response}</pre>`
                : "Отстутствует";
            responseText += `\nПереслать автоматически сгенерированный ответ (нажмите, чтобы
                скопировать):\n<code>/respondLLM ${message.uuid}</code>`;
            responseText += `\nОтправить вручную составленный ответ (нажмите, чтобы скопировать):\n
                <code>/respondManual ${message.uuid} Текст с ответом...</code>`;
            await this.sendAdminResponse(context, responseText, { parse_mode: "HTML" });
          }
        } catch (forwardError) {
          console.error(`Ошибка при пересылке сообщения UUID:${message.uuid}:`, forwardError);
          await this.sendAdminResponse(context, `Ошибка при пересылке сообщения UUID:${message.uuid}
              : ${forwardError}`);
          continue;
        }
      }

      await this.sendAdminResponse(context, "Сообщения, ожидающие проверки, пересланы.");
    } catch (error) {
      console.error("Ошибка при получении списка сообщений, ожидающих проверки:", error);
      await this.sendAdminResponse(context, "Не удалось получить список сообщений, ожидающих
          проверки.");
    }
  }
}
```

Рисунок 8 – Листинг исходного кода класса «ListPendingCommand»

Для моделирования сущностей базы данных в микросервисе используются интерфейсы «Message», «ResponseExample», «User», каждый из которых имеет соответствующий аналог в базе данных PostgreSQL. Интерфейс «Message» содержит информацию о пользовательских запросах к кафедре, на различных этапах обработки (поле «status»). Возможные значения включают в себя:

– pending-llm: запрос создан, в поле «user_message» записан оригинальный текст пользовательского сообщения. Система ожидает результата автоматической генерации ответа для сохранения значения «llm_response»;

– pending-verification: автоматический ответ сгенерирован, ожидается проверка запроса администратором для отправки сформированного сотрудником или нейросетью ответа;

– message-forwarded: сотрудник кафедры проверил ответ и переслал ответ студенту (на этом этапе сохраняются значения «forwarded_response» и «response_by_admin_id»)

– failed: при обработке запроса произошла ошибка.

Интерфейс «ResponseExample» соответствует таблице «response_examples» из базы данных, и содержит пример ответов на один или несколько вопросов в текстовом поле «example_text». Данный текст пересылается вместе с запросом пользователя для автоматической генерации ответа нейросетью через «LlmApiService».

Поля интерфейса «User» также полностью соответствуют сущности, прежде рассмотренной на этапе анализа схемы базы данных. Взаимодействие с данными пользователей осуществляется посредством сервиса «UserService».

Как было упомянуто ранее, микросервис «llmService» выполняет роль посредника между сервисом «messengerService» и облачными LLM провайдерами, в связи с чем не содержит сложной бизнес-логики или классовых структур.

2.4 Автоматизированное тестирование системы

Теперь рассмотрим систему тестирования в приложении. Нагрузочное тестирование системы осуществляется посредством симуляции отправки 250 запросов от уникальных пользователей Telegram за три минуты посредством

дополнительного скрипта [11]. Асинхронная логика работы системы допускает возможность параллелизации процесса сохранения исходных данных и генерации ответов с помощью LLM. За счёт этого все запросы сохраняются без каких-либо задержек, а ответы поочередно генерируются с неблокирующей задержкой. Выполненный этап нагрузочного тестирования позволил сделать вывод о работоспособности системы даже при высоких нагрузках, без заметного снижения уровня производительности.

Помимо процедуры нагрузочного тестирования, для проверки корректности работы системы в проекте реализован набор тестовых сценариев (unit-тестов) с использованием JavaScript библиотеки «Jest». Одной из её главных функций является создание mock-зависимостей для имитации работы отдельных компонентов системы, отслеживания вызовов, замену возвращаемых данных и эмуляции сбоев на различных этапах обработки, включая подмену ответов на HTTP-запросы к внешним сервисам [15]. В рамках данного проекта были добавлены тесты на этапе от получения и валидации входящего сообщения до финальной записи обработанной заявки в базу данных. В приложение также добавлены автоматические тесты корректности работы класса «LlmApiService» в случае возникновения ошибок на стороне микросервиса «llmService» (см. рисунок 9).

```
it('should successfully send a message to the LLM service and return the response', async () => {
    const mockResponse = { response: 'Mocked LLM response' };
    mockedFetch.mockResolvedValue({
        ok: true,
        json: async () => mockResponse,
    } as any);

    const message = 'Текстовое сообщение';
    const messageId = '123';
    const exampleText = 'Примеры ответов на вопросы для LLM';

    const response = await llmApiService.sendToLLMService(message, messageId, exampleText);

    expect(response).toEqual(mockResponse.response);
    expect(mockedFetch).toHaveBeenCalledTimes(1);
    expect(mockedFetch).toHaveBeenCalledWith(`${llmServiceUrl}/generate`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ message: message, messageId: messageId, exampleText: exampleText }
            ),
    });
});

it('should handle fetch errors when the LLM service is unavailable', async () => {
    mockedFetch.mockRejectedValue(new Error('Network error'));

    const message = 'Текстовое сообщение';
    const messageId = '123';
    const exampleText = 'Примеры ответов на вопросы для LLM';

    await expect(llmApiService.sendToLLMService(message, messageId, exampleText))
        .rejects.toThrow('Network error');
    expect(mockedFetch).toHaveBeenCalledTimes(1);
});
```

Рисунок 9 – Исходный код unit-теста, проверяющего работу класса «LlmApiService»

Как было упомянуто раньше, при каждом внесении изменений в облачный репозиторий, система GitLab CI/CD осуществляет автоматический запуск тестов (см. рисунок 10). Это значительно упрощает процесс отслеживания возникновения ошибок между различными версиями исходного кода [18].

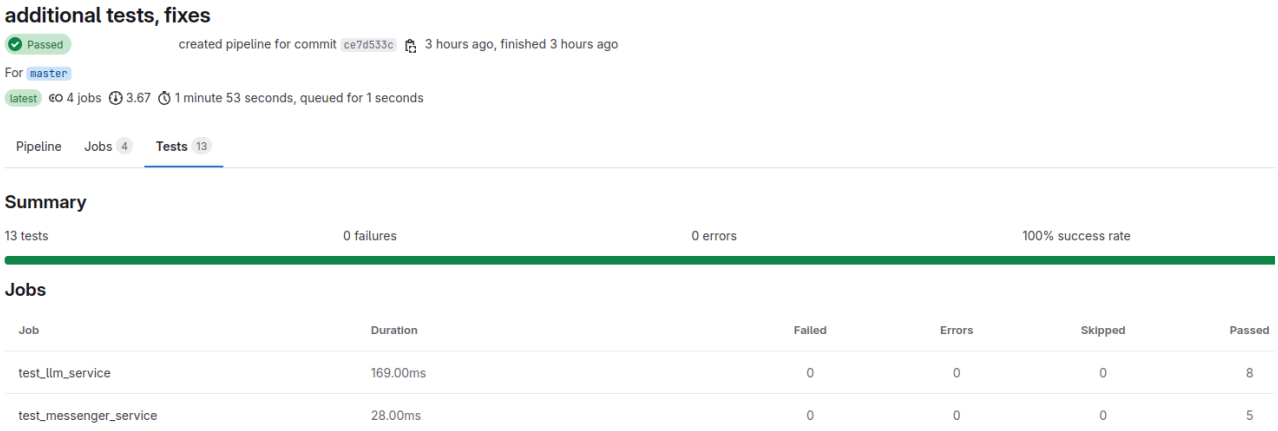

Рисунок 10 – Отчёты о выполнении тестов для микросервисов «messengerService» и «llmService»

При открытии одной из работ, выполненных CI/CD можно увидеть детальный отчёт со списком завершенных тестов (см. рисунок 11):

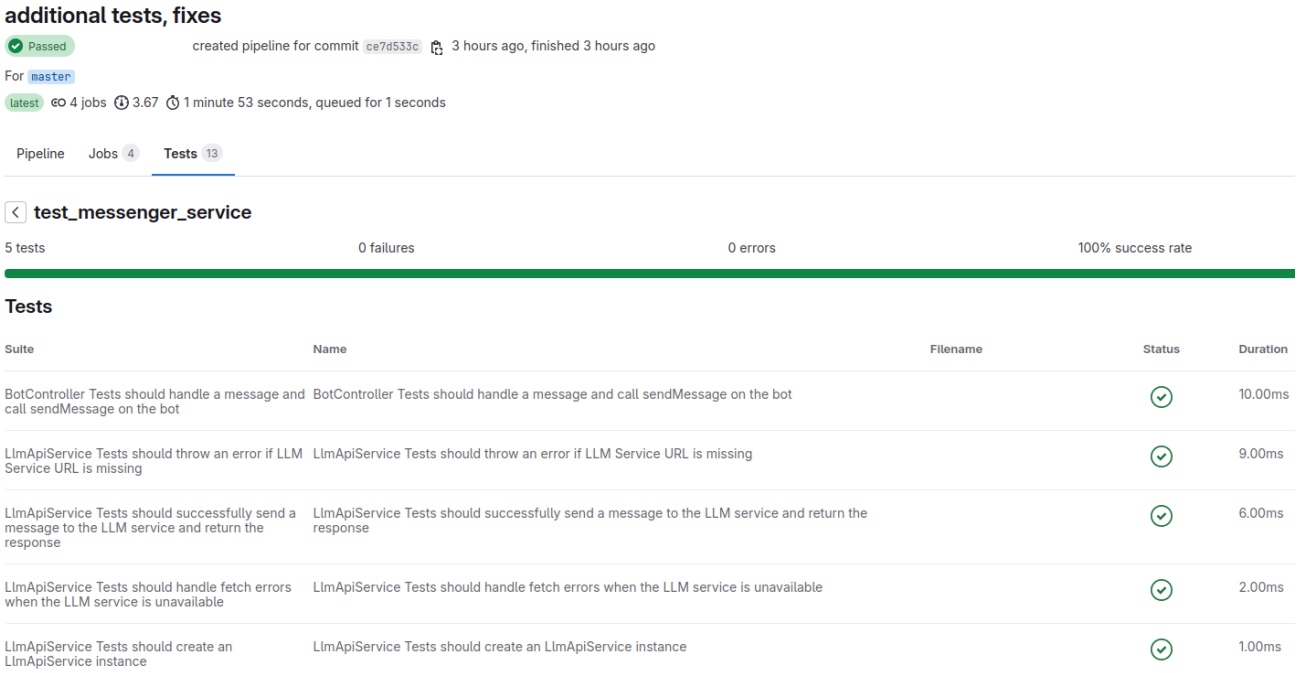

Рисунок 11 – Отчёты о выполнении тестов для класса микросервиса «messengerService»

На снимке экрана перечислены все тесты, прошедшие проверку в микросервисе «messengerService». Помимо автоматизированных тестов, на этапе проектирования приложения были предусмотрены и разработаны

контрольные примеры, обеспечивающие многоцелевое тестирование базы данных и прикладных программ. Они будут более детально рассмотрены после этого раздела с инструкциями для различных категорий пользователей с учётом уровня их технической подготовки.

2.5 Разработка инструкций для всех категорий пользователей

Начнём с составления инструкции для категории «конечный пользователь» (студент, использующий чат-бот для отправки запросов и получения ответов от сотрудников кафедры): Откройте приложение Telegram и введите в поле поиска имя пользователя чат-бота департамента. Начните переписку с чат-ботом. Для этого нужно нажать кнопку «Начать» или отправить команду /start. Свой текстовый запрос или вопрос необходимо ввести в поле чата, а затем нажать кнопку отправки сообщения. Пожалуйста, обратите внимание, что бот может обрабатывать только текстовые сообщения. После того, как текстовый запрос был передан, система отправит автоматическое подтверждение с текстом: «Ваш запрос сохранён, ожидайте ответа. Время создания запроса: …». Пользователям рекомендуется не отправлять дубликаты запросов. Если обнаружены технические ошибки в работе чат-бота, необходимо напрямую обратиться к сотрудникам кафедры. Это следует сделать, отправив по электронной почте снимок экрана и описание возникшей проблемы.

Инструкция для категории пользователей «Сотрудник кафедры»: Для того, чтобы получить административные привилегии, сотрудники кафедры должны отправить официальный запрос через чат-бот со следующим текстом сообщения: «Требуется административный доступ для [Полное имя], должность: [Название должности], ответственный за [Краткое описание обязанностей]». Проверка и выдача прав администратора осуществляется

техническим персоналом. После того как статус пользователя повышен до статуса администратора, становится доступен следующий набор команд:

– /listPending: Отображает все запросы, ожидающие ответа. Информация о запросе включает в себя идентификатор пользователей, время создания и ответ, сгенерированный LLM. Каждый запрос содержит две команды для копирования: /respondLLM [ID_запроса] (пересылка сгенерированного при помощи LLM ответа) или /respondManual [ID_запроса] [Текст с ответом…] (отправка вручную составленного ответа)

– /respondLLM [ID_запроса]: Эта команда передает предварительно сгенерированный нейросетью ответ указанному пользователю. При этом, если длина автоматически составленного ответа менее 10 символов, его отправка будет отклонена.

– /respondManual [ID_запроса] [Текст_ответа]: Эта команда позволяет ввести отредактированный текстовый ответ, который будет перенаправлен пользователю.

– /setResponseExamples [Текст]: Это команда настройки примеров ответов на вопросы. Используется сервисом LLM для генерации ответов.

– /getResponseExamples: Эта команда извлекает текущий набор примеров ответов на вопросы для аудита.

Инструкция для категории пользователей «Технический персонал» (занимается обслуживанием и администрированием системы, имеет прямой доступ к серверу и базе данных PostgreSQL). Для выполнения обслуживания следует использовать следующие команды:

– доступ к базе данных через командный интерфейс: kubectl exec -it postgres-0 – psql -U [имя_пользователя_БД] -d [название_базы_данных];

– открытие портов для подключение к базе данных по протоколу: kubectl port-forward service/postgres 5432:5432;

– предоставление роли администратора: UPDATE users SET role=‘admin’ WHERE user_id=‘[ID_пользователя_Telegram]’;

– блокировка пользователей: UPDATE users SET is_banned=TRUE WHERE user_id=‘[ID_пользователя_Telegram]’;

– удаление конкретного сообщения из базы данных: DELETE FROM messages WHERE uuid = ‘[ID_сообщения]’;

– диагностический анализ: команда для просмотра консольных сообщений микросервисов kubectl logs -f deployment/messenger-deployment –tail=500 или kubectl logs -f deployment/llm-service-deployment –tail=500.

2.6 Контрольный пример реализации проекта

В следующей таблице приведены ключевые сценарии (контрольные примеры) взаимодействия сотрудника кафедры с правами администратора и студента посредством чат-бота. Выполнение этих этапов позволит провести многоцелевое тестирование базы данных и прикладных программ (см. таблицу 1).

Таблица 1 – контрольные примеры реализации проекта, обеспечивающие тестирование базы данных и прикладных программ

№	Тестируемые функции	Среда тестирования	Тестовые данные	Процедура	Ожидаемое поведение системы
1	Сохранение запроса пользователя	Telegram, права пользователя	«Возможно ли пройти производственную практику на кафедре ПМИФ?»	Пользователь отправляет текстовое сообщение	Бот отвечает: «Ваш запрос сохранён, ожидайте ответа. Время создания запроса: …»
2	Просмотр заявок, ожидающих ответов	Telegram, права администратора	Команда /listPending	Администратор отправляет команду чат-боту	Бот пересылает сообщение админу, отображает команды для копирования: «Переслать автоматически сгенерированный ответ: /respondLLM [ID Сообщения] Отправить вручную: /respondManual [ID Сообщения] Текст ответа…»
3	Отправка ручного ответа	Telegram, права администратора	Команда /respondManual [ID Сообщения] «Да, возможно…»	Администратор копирует команду из списка сообщений, ожидающих ответа и передаёт вручную составляет ответ в качестве второго параметра	Бот подтверждает: «Ручной ответ на сообщение ... успешно отправлен». Пользователь получает сообщение: «Ответ на запрос от [дата]: Да, на кафедре ПМИФ возможно…»

Продолжение таблицы 1

4	Управление примерами ответов	Telegram, права администратора	/getResponseExamples -> /setResponseExamples «Вопрос: Когда проходит зимняя и летняя сессия… Ответ: …» -> /getResponseExamples	Администратор последовательно применяет команды: 1) проверка примеров ответов 2) сохранение нового примера 3) проверка примеров ответов	1) Ответ «Примеры ответов не найдены» 2) Ответ «Примеры ответов сохранены» 3) Получение примеров ответов на вопросы: «Вопрос: Когда проходит зимняя и летняя сессия в университете? Ответ: …примерно с 8 по 17 января, для третьего-пятого курсов – с 8 по 19 января…»
5	Сохранение запроса пользователя	Telegram, права пользователя	«Какого числа заканчивается зимняя сессия?»	Пользователь отправляет текстовое сообщение	Бот отвечает: «Ваш запрос сохранён, ожидайте ответа. Время создания запроса: …»
6	Автоматическая генерация ответа	Telegram, права администратора	Команда /respondLLM [ID Сообщения]	Администратор использует команду для отправки автоматически сгенерированного ответа	LLM формирует ответ по сохранённым примерам. Пользователь получает ответ: «Для студентов первого и второго курсов сессия заканчивается 17 января, для студентов третьего-пятого курсов – 19 января».
7	Вывод ошибки при отправке длинного запроса	Telegram, права пользователя	Любое сообщение длинной более 2500 символов	Пользователь отправляет текстовое сообщение длинной более 2500 символов	Бот отвечает: «Не удалось обработать ваше сообщение, так как оно слишком длинное (максимум: 2500 символов)»
8	Вывод ошибки при отправке изображения	Telegram, права пользователя	Сообщение с изображением без текста	Пользователь отправляет изображение без подписи	Бот отвечает: «Не удалось обработать ваше сообщение, так как оно не содержит текст»
9	Вывод ошибки при сбое работы БД	Telegram, права пользователя	Любое сообщение	Пользователь отправляет сообщение при потере соединения с БД на сервере	Бот отвечает: «Не удалось обработать ваше сообщение, повторите попытку позже»

Выполнение каждого из этапов демонстрируется на снимках экрана. Взаимодействие начинается с отправки команды /start и ввода сообщения с текстовым запросом (см. рисунок 12):

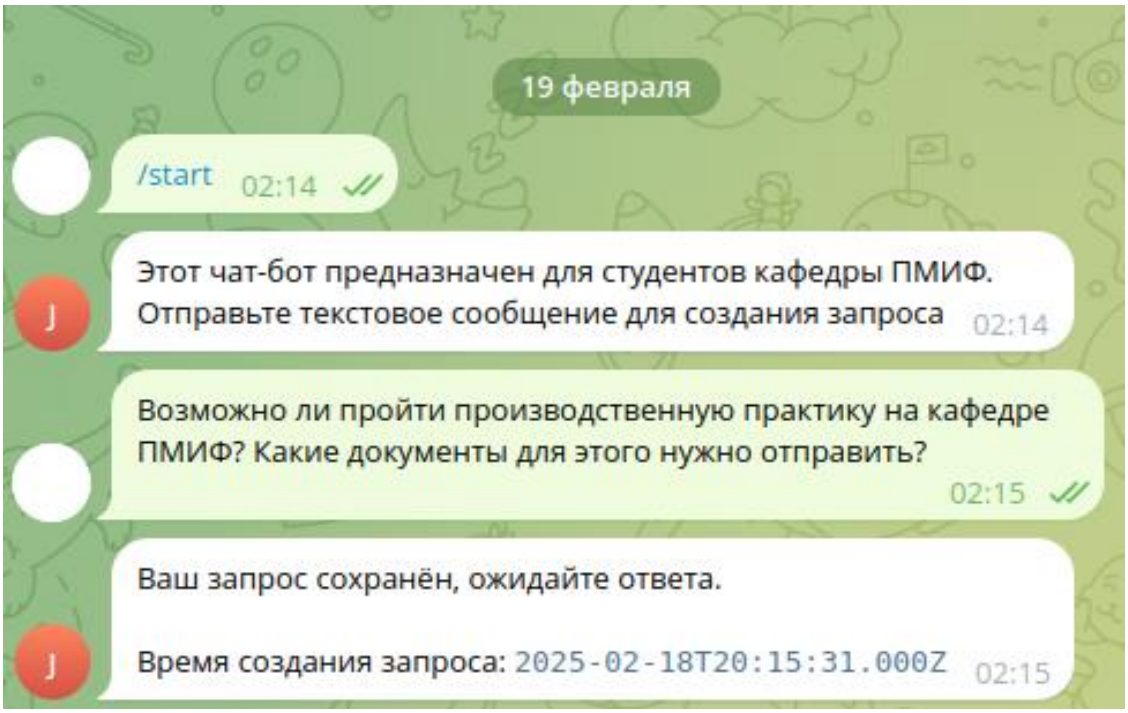

Рисунок 12 – Отправка запроса в чат-бот (вид: конечный пользователь)

Получив доступ к аккаунту администратора, просмотрим список заявок, ожидающих ответов. Скопируем команду, нажав на текст /respondManual [ID Сообщения] и отправим вручную составленный ответ (см. рисунок 13).

Рисунок 13 – Просмотр заявок, отправка вручную составленного ответа (вид: сотрудник кафедры с админ. правами)

Затем конечный пользователь автоматически получает вручную составленный ответ на вопрос (см. рисунок 14):

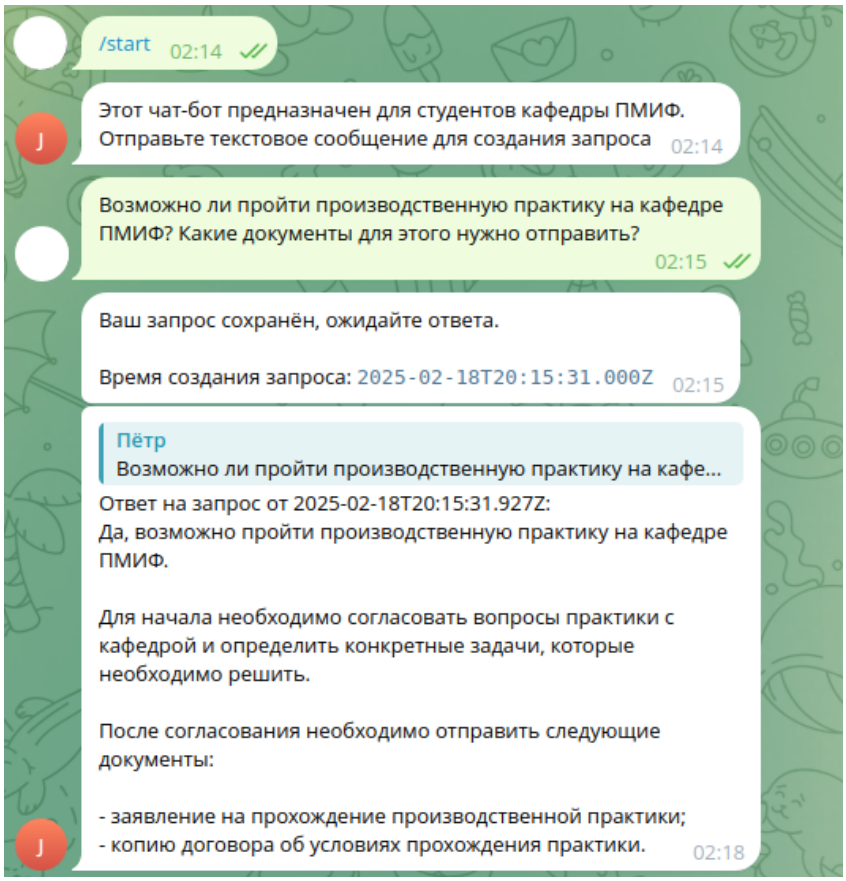

Рисунок 14 – Получение вручную составленного ответа на вопрос (вид: конечный пользователь)

В соответствии с перечнем контрольных примеров (табл. 1), выполним с аккаунта сотрудника кафедры этап «Управление примерами ответов» (см. рисунок 15). Составленный текст о периоде экзаменационной сессии будет пересылаться вместе с пользовательским запросом в облачный сервис генерации текста:

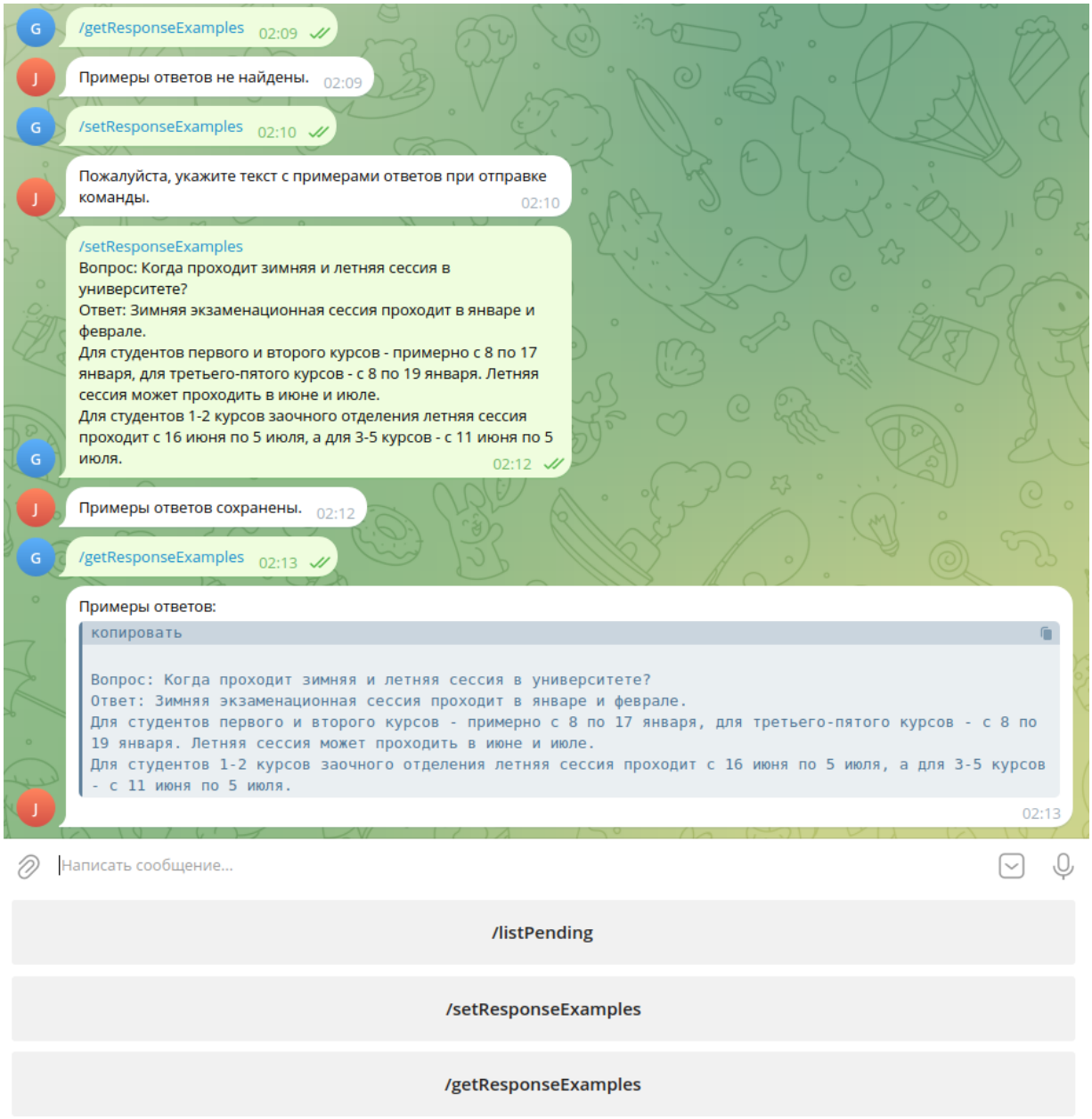

Рисунок 15 – Управление примерами ответов на вопросы для LLM (вид: сотрудник кафедры с админ. правами)

Отправим ещё один запрос для проверки корректности работы генерации ответа на основе сохранённых примеров (см. рисунок 16):

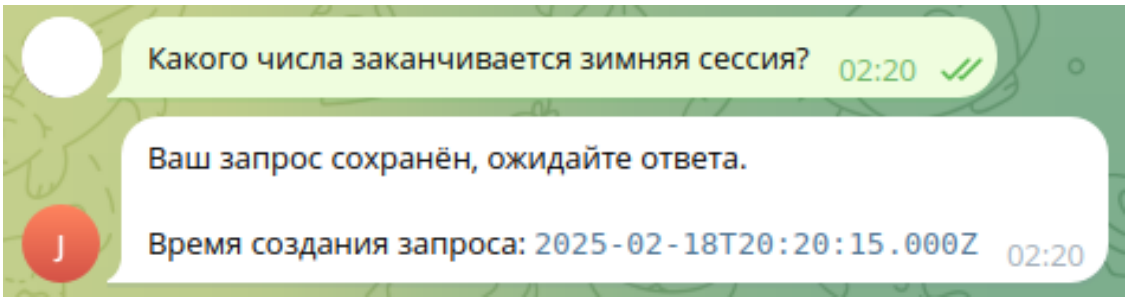

Рисунок 16 – Отправка второго запроса в чат-бот (вид: конечный пользователь)

Так как перед получением пользовательского запроса был указан пример ответа на вопрос, связанный со сроками сессии, LLM корректно сгенерирует ответ на основе указанной информации, а именно: «Для студентов первого и второго курсов сессия заканчивается 17-го января, для студентов третьего-пятого курсов – 19 января» (см. рисунок 17):

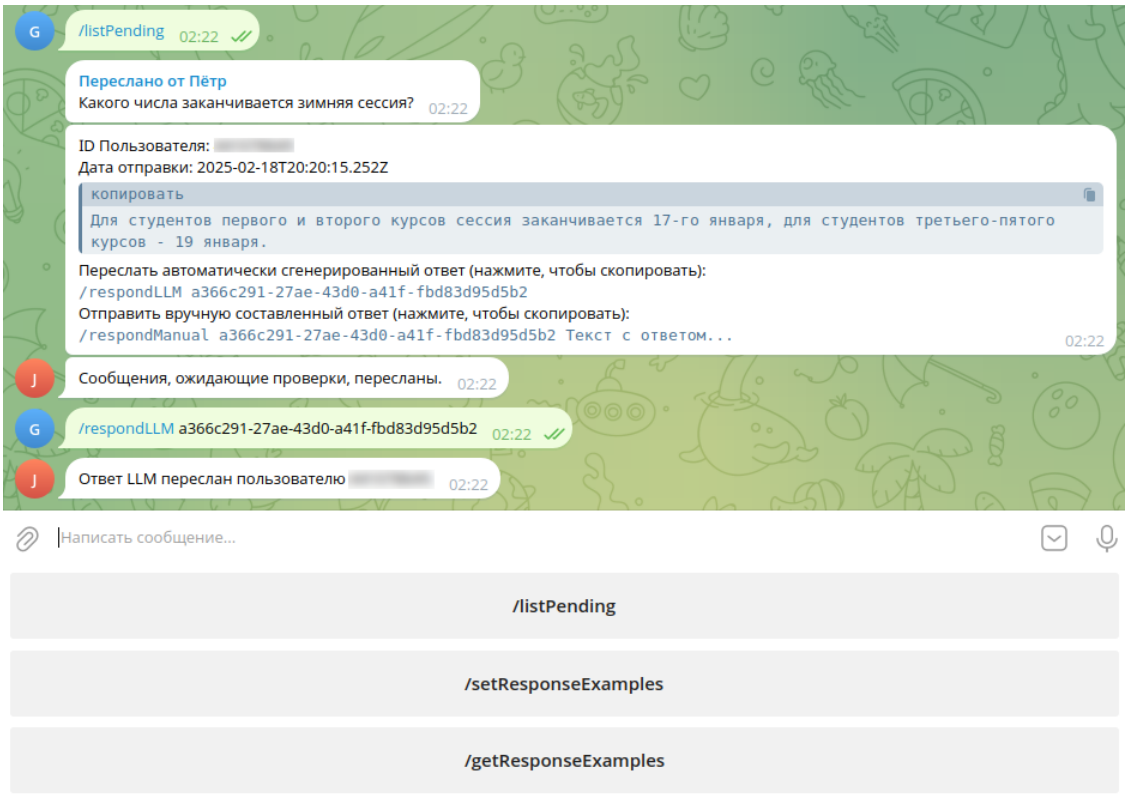

Рисунок 17 – Просмотр заявок, отправка автоматически составленного ответа (вид: сотрудник кафедры с админ. правами)

После подтверждения со стороны сотрудника, автоматически сгенерированный ответ пересылается в чат с конечным пользователем (см. рисунок 18):

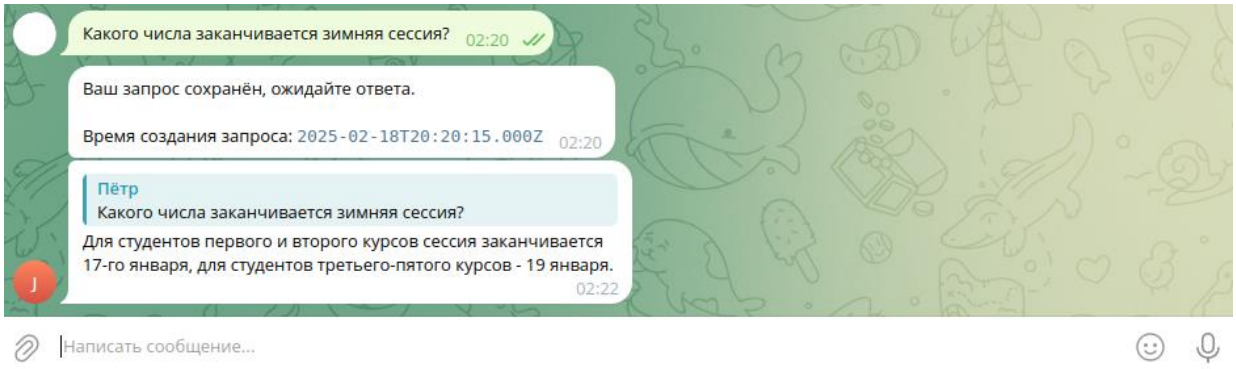

Рисунок 18 – Получение автоматически сгенерированного ответа на вопрос (вид: конечный пользователь)

Также проверим работу приложения в случая возникновения нескольких негативных сценариев. Например, отправка слишком длинного текстового сообщения (см. рисунок 19):

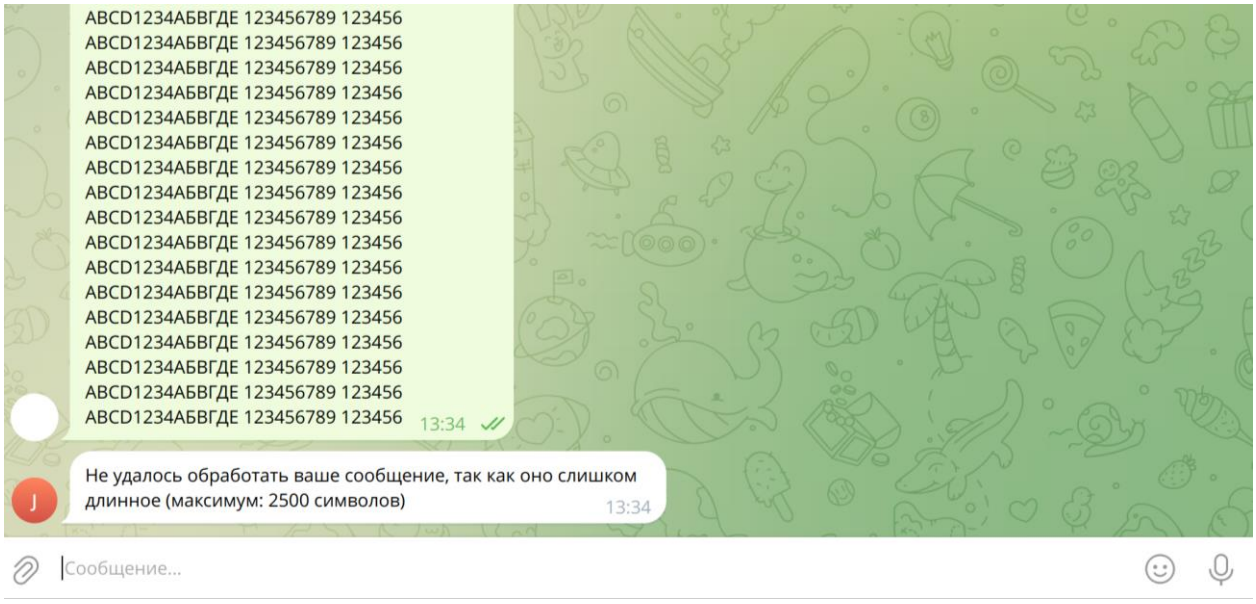

Рисунок 19 – Отправка слишком длинного сообщения в чат-бот (вид: конечный пользователь)

Проверим ответ чат-бота при отправке изображения без подписи (см. рисунок 20):

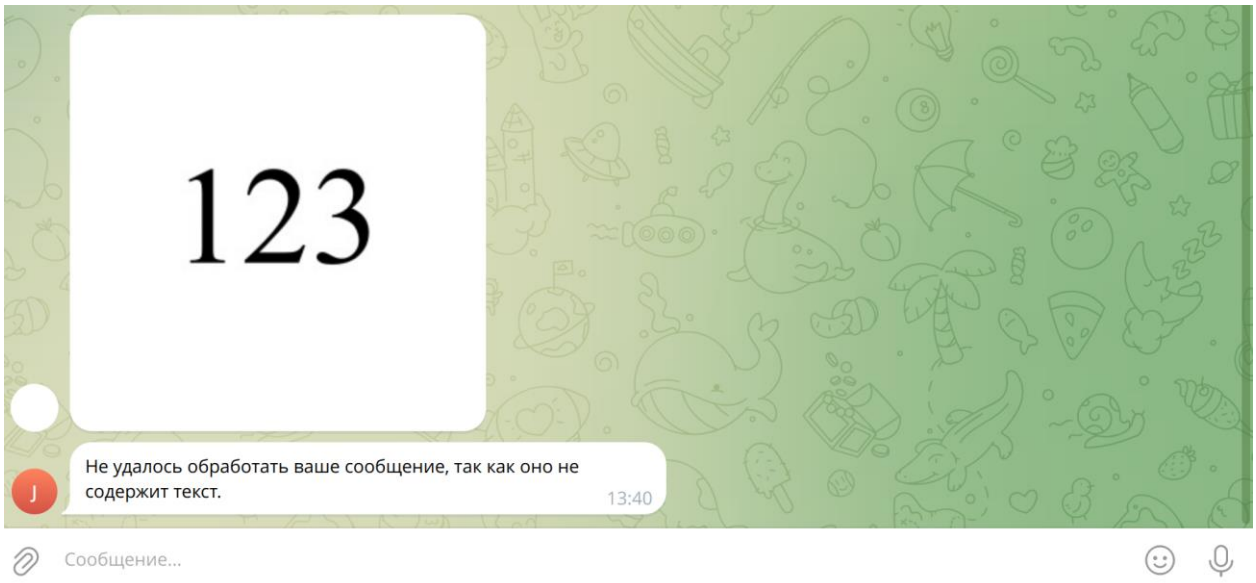

Рисунок 20 – Отправка изображения без подписи (вид: конечный пользователь)

Также в приложении предусмотрена логика обработки запросов при нарушении работы одного из компонентов, например – в случае потери подключения к базе данных. Для проверки данного сценария во время работы системы была остановлена работа БД командой «kubectl delete service postgres», после чего чат-бот начал отвечать на сообщения предупреждением об ошибке (см. рисунок 21).

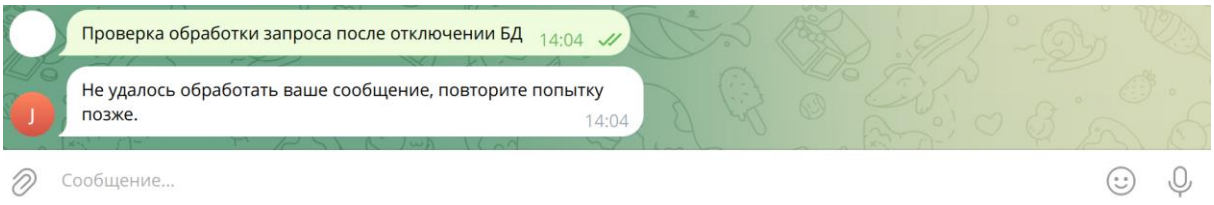

Рисунок 21 – Сообщение об ошибке при неполадках соединения с базой данных (вид: конечный пользователь)

Выполненное тестирование приложения с использованием перечня контрольных примеров позволило убедиться в успешной реализации поставленных задач по разработке чат-бота для кафедры университета.

2.7 Сравнение конечного продукта с готовым решением

Для сравнения представленного решения с готовыми аналогами в качестве примера можно использовать no-code платформу crisp.chat, предназначенную для управления взаимоотношениями с клиентами посредством использования центрального хранилища сообщений из различных каналов коммуникации. Платформа также предоставляет CRM для хранения информации о клиентах, базу знаний, систему тикетов для сложных запросов и интеграцию с Telegram [8], что позволяет получать и отправлять сообщения по аналогии с другими каналами: электронной почтой, виджетом на сайте, и так далее (см. рисунок 22).

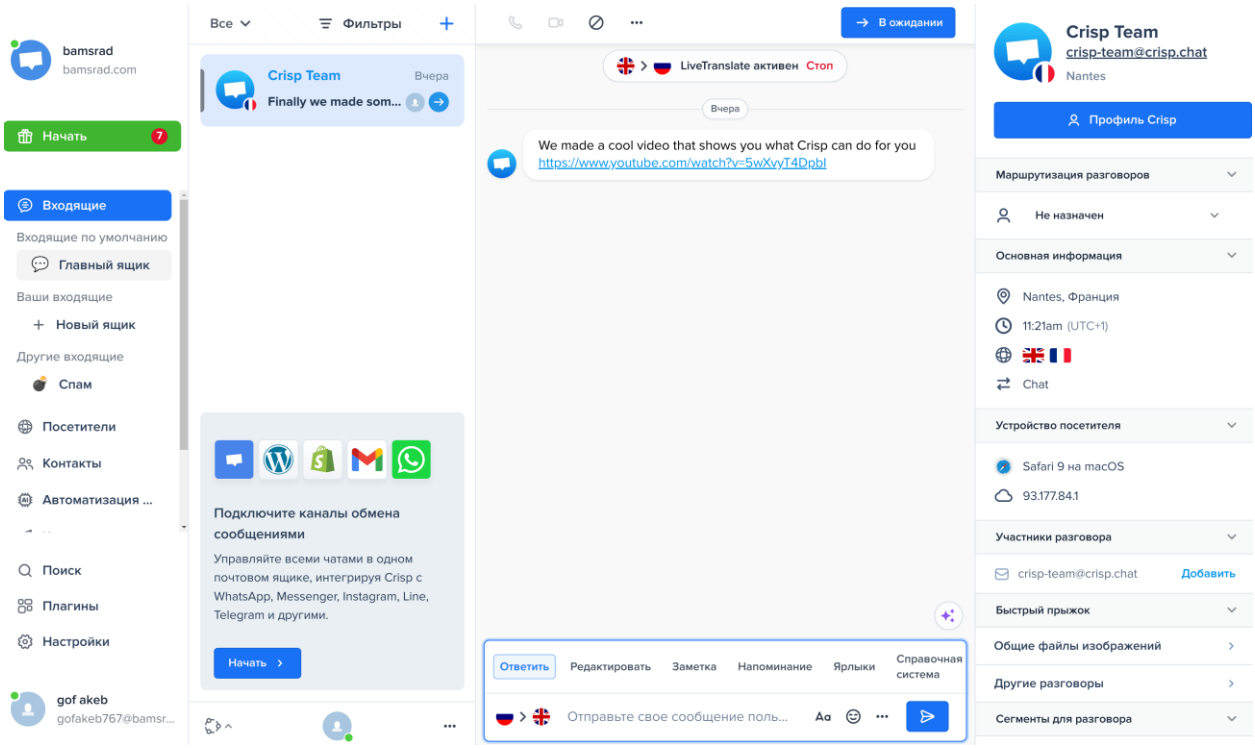

Рисунок 22 – Веб-интерфейс платформы Crisp

Пользователям Crisp также предоставляется доступ к ИИ модели для автоматического составления ответов на базе знаний и истории сообщений. Стоит отметить, что при сравнении разработанной системы с готовой платформой Crisp можно выделить следующий набор ограничений: модель оплаты только по подписке (от $25/месяц и выше), закрытый исходный код, хранение переписок и данных клиентов доступно только посредством облачного хранилища Crisp. Большинство этих недостатков вызвано зависимостью от поставщика, что в дальнейшем может привести к замедлению внедрения доработок и интеграций, увеличению затрат и рисков при переносе данных на другую платформу.

Глава 3 Оценка экономической эффективности проекта

3.1 Оценка затрат на реализацию и эксплуатацию проекта

Теперь перейдём к анализу экономической эффективности реализованного проекта, начав с рассмотрения затрат, связанных с разработкой, внедрением и эксплуатацией представленного решения. Начнём с составления таблицы, демонстрирующей ежемесячную зарплаты персонала, задействованного в разработке, тестировании, развёртывании и поддержке системы (см. таблицу 2):

Таблица 2 – должности сотрудников с ежемесячной зарплатой

Роль	Ежемесячная зарплата (руб)
Сотрудник кафедры	60 000
Разработчик (почасовая оплата)	150 000
Тестировщик (почасовая оплата)	100 000
Руководитель проекта (почасовая оплата)	200 000

Помимо явных первоначальных и ежемесячных затрат, важно учитывать и скрытые расходы, которые могут возникнуть в ходе внедрения и эксплуатации системы:

- разовые расходы на обучение сотрудников для эффективной работы с новой системой, создание вспомогательных инструкций и прочих материалов;
- обновление ПО для внедрения дополнительного функционала и интеграций при возникновении новых требований, исправление ошибок и повышение удобства использования системы;
- резервное копирование данных, принятие мер по обеспечению информационной безопасности.

Эти факторы следует закладывать в долгосрочное планирование, так как их игнорирование может привести к непредвиденным расходам. Например, ежемесячные скрытые затраты на поддержку безопасности, резервного копирования и обновлений могут достигать до 15% от изначального ежемесячного бюджета, а первоначальные инвестиции в обучение персонала – от 10 000 до 50 000 рублей в зависимости от масштаба внедрения [14].

Составим приблизительные расчёты расходов на разработку и запуск первичной версии приложения в виде таблицы 3.

Таблица 3 – сводка первоначальных затрат

Роль	Оценочное время	Почасовая ставка (руб)	Общая стоимость (руб)
Разработчик	120	937.5	112 500
Тестировщик	40	625	25 000
Руководитель проекта	40	1250	50 000
Документация (разработчик)	20	937.5	18 750
Тестирование (тестировщик)	20	625	12 500
Развертывание (разработчик)	10	937.5	9 375
Обучение персонала (разработчик + сотрудник кафедры x2)	~12	937.5+375+375= 1 687.5	20 250
Итого	–	–	248 375

Рассмотрим затраты на использование LLM провайдера. Для расчёта ежемесячных затрат предположим, что изначально чат-бот будет в среднем обрабатывать 500 запросов в месяц (данный показатель может быть значительно увеличен при внедрении чат-бота в другие подразделения университета). Стоимость использования бессерверных вычислений для генерации текста от сервиса Together AI варьируется в зависимости от выбранной модели и количества обрабатываемых токенов [9]. Для расчётов в качестве средних значений для обработки пользовательского запроса возьмём

стоимость обработки 2000 входных токенов и генерацию 320 выходных токенов (см. таблицу 4). В среднем для русского языка 1000 токенов соответствует примерно 375 словам.

Таблица 4 – параметры, используемые для расчёта стоимости генерации текста при помощи LLM

Параметры	Описание	Значение
Общее количество запросов LLM за месяц	Количество запросов, обрабатываемых чат-ботом за месяц	500
Среднее количество входных токенов на запрос	Входные токены	2000
Среднее количество выходных токенов на запрос	Выходные токены	320
Цена входного токена (за 1000 токенов), руб	Цена использования модели (при цене 2000 за млн токенов и курсе 90 руб = $1)	2
Цена выходного токена (за 1000 токенов), руб	Цена использования модели (при цене 7500 за млн токенов и курсе 90 руб = $1)	7.5

Стоит отметить, что в качестве альтернативы использования услуг облачных провайдеров существует возможность самостоятельного хостинга LLM, но данный сценарий выходит за рамки нынешнего анализа. Для того чтобы оценить масштабируемость решения, было рассмотрено возможное увеличение нагрузки в 2, 5 и 10 раз. Основным фактором при этом является количество запросов, обрабатываемых информационной системой. Также масштабируемость осуществляется за счёт регулировки выделенных ресурсов в кластере Kubernetes, на котором развернуты микросервисы и прочие компоненты. Были рассмотрены несколько вариантов конфигураций и их стоимость (см. таблицу 5).

Таблица 5 – таблица подсчёта итоговой ежемесячной стоимости

Уровень масштабирования	Запросов в месяц	Ежемесячная стоимость Kubernetes (руб)	Стоимость генерации LLM (руб)	Итоговая ежемесячная стоимость (руб)
Изначальный	500	7 200	500 × 6.4 = 3 200	7 200 + 3 200 = 10 400
x2	1 000	7 200	1 000 × 6.4 = 6 400	7 200 + 6 400 = 13 600
x5	2 500	7 200	2 500 × 6.4 = 16 000	7 200 + 16 000 = 23 200
x10	5 000	14 400	5 000 × 6.4 = 32 000	14 400 + 32 000 = 46 400

Изначально, как наиболее оптимальный баланс затрат и надёжности, рассматривается конфигурация с тремя мастер-нодами (2 ядра, 4 ГБ ОЗУ) за 7 200 рублей в месяц. В случае десятикратного увеличения нагрузки на чат-бот может понадобиться переход на конфигурацию 3 мастер-нод (4 ядра, 8 ГБ ОЗУ) за 14 400 рублей в месяц [6].

3.2 Расчёт срока окупаемости проекта

Что же касается экономической эффективности проекта, разработанный чат-бот предоставляет ряд существенных преимуществ, ключевым из которых является сокращение времени, в среднем необходимого для обработки заявок и составления ~75% ответов (в зависимости от объёма и сложности запроса). Это в свою очередь снижает нагрузку на сотрудников и повышает качество обслуживания. Анализ взаимодействий с чат-ботом помогает выявлять популярные вопросы, что также поспособствует дальнейшему росту эффективности генератора ответов.

Приступим к расчёту окупаемости, исходя из следующих условий. Экономия времени на одном запросе, для которого был корректно сгенерирован ответ составляет ~15 минут (сокращение с 20 до 5 минут). Изначально обрабатывается 500 запросов в месяц, при этом 75% запросов

автоматизируется чат-ботом. Потраченная зарплата сотрудника кафедры: 375 руб/час (после пересчёта из 60 000 руб/мес). Ежемесячные затраты: 11 960 руб (10 400 руб/мес на систему и 15% скрытых затрат ежемесячно). Первоначальные вложения: 248 375 руб (разработка, запуск, затраты на обучение сотрудников). Исходя из этого, рассчитаем следующие экономические показатели.

Количество автоматизируемых запросов в месяц: 500 запросов × 75% = 375 запросов

Экономия времени в месяц: 375 запросов × 0.25 ч (15 мин) = 93.75 часа

Экономия по зарплате сотрудников: 93.75 часа × 375 руб = 35 156 руб

Рассчитаем чистую экономию за месяц как разницу дохода от автоматизации и ежемесячных затрат: 35 156 − 11 960 = 23 196 руб/мес

Срок окупаемости рассчитаем как частное первоначальных вложений и чистой экономии: 248 375 ÷ 23 196 ≈ 10.7 месяцев ≈ 11 месяцев.

Исходя из расчётов можно сделать вывод о том, что внедрение чат-бота начнёт экономить ресурсы приблизительно через 11 месяцев после запуска [17]. Этот срок включает возврат первоначальных инвестиций (248 375 руб) за счёт чистой экономии (23 196 руб в месяц). При внедрении системы в другие подразделения университета, а также естественном увеличении нагрузки срок окупаемости будет сокращаться за счёт роста дохода от автоматизации растущего количества запросов. При этом стоит отметить, что в дальнейшем расчёты показателей экономической выгоды, эффективности автоматизации и масштабируемости могут быть повторно скорректированы на основе собранных данных о работе системы в реальных условиях.

Заключение

В рамках данной работы была поставлена задача по сбору материалов, проведению анализа существующих бизнес-процессов и последующей разработкой программного решения для организации, предназначенного для автоматизации процесса обработки заявок студентов. На подготовительном этапе, предваряющем непосредственную реализацию программного продукта, был проведён анализ организации и связанной с ней предметной области, а также определены функциональные требования к решению. На следующем этапе была разработана концептуальная схема данных «сущность-связь» и подготовлены UML-диаграммы для моделирования логической и физической архитектуры проекта. Для выполнения поставленных задач была разработана система на базе микросервисного подхода, где ключевыми компонентами стали микросервисы «messengerService» и «llmService» со следующим функционалом: взаимодействие с Telegram API для приёма и отправки сообщений; сохранение и обработка запросов студентов; обработка команд администраторов; интеграция с облачными LLM провайдерами. Следует отметить, что применение методологии DevOps в комбинации с GitLab CI/CD позволило автоматизировать сборку программного продукта и выполнение тестов для каждой новой версии исходного кода, загружаемого в облачный репозиторий проекта. Остальные задачи, поставленные в рамках работы были также успешно выполнены, включая проработку всех обусловленных требований к итоговому программному продукту, разработку архитектуры системы и выбор оптимальных методологических решений для проектирования. По завершению работы был получен обширный набор компетенций и навыков, необходимых для дальнейшей исследовательской и профессиональной деятельности, связанной с разработкой программного обеспечения и автоматизацией деятельности организаций.

Список используемой литературы и используемых источников

1. Command Design Pattern | GeeksforGeeks [Электронный ресурс]. URL: https://www.geeksforgeeks.org/command-pattern/ (дата обращения: 05.01.2025)

2. Dialogflow – Wikipedia [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Dialogflow (дата обращения: 05.01.2025)

3. Entity–relationship model – Wikipedia [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Entity–relationship_model (дата обращения: 09.01.2025)

4. How to Develop As-Is and To-Be Business Process? [Электронный ресурс]. URL: https://www.visual-paradigm.com/tutorials/as-is-to-be-business-process.jsp (дата обращения: 02.01.2025)

5. Kubernetes Scaling: The Comprehensive Guide to Scaling Apps | NetApp [Электронный ресурс]. URL: https://www.netapp.com/learn/cvo-blg-kubernetes-scaling-the-comprehensive-guide-to-scaling-apps/ (дата обращения: 15.01.2025)

6. Kubernetes в облаке (KaaS) | Создание и установка кластера Kubernetes | Рег.ру [Электронный ресурс]. URL: https://www.reg.ru/cloud/managed-kubernetes (дата обращения: 15.01.2025)

7. Multitier architecture – Wikipedia [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Multitier_architecture (дата обращения: 04.01.2025)

8. The Business Messaging Platform For Every Company – Crisp [Электронный ресурс]. URL: https://crisp.chat/en (дата обращения: 10.05.2025)

9. Together AI – The AI Acceleration Cloud [Электронный ресурс]. URL: https://www.together.ai/ (дата обращения: 14.01.2025)

10. UUID – Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/UUID (дата обращения: 10.01.2025)

11. What is Stress Testing in Software Development? | BrowserStack [Электронный ресурс]. URL: https://www.browserstack.com/guide/stress-testing (дата обращения: 07.05.2025)

12. Баланов, А. Н. DevOps: интеграция и автоматизация : учебное пособие для вузов / А. Н. Баланов. – 2-е изд., стер. – Санкт-Петербург : Лань, 2025. 240 с.

13. Баланов, А. Н. Бэкенд-разработка веб-приложений: архитектура, проектирование и управление проектами : учебное пособие для вузов / А. Н. Баланов. – 2-е изд., стер. – Санкт-Петербург : Лань, 2025. 312 с.

14. Как учесть в смете скрытые расходы: методики, анализ и практические рекомендации [Электронный ресурс]. URL: https://смета.москва/stati/kak-uchest-v-smete-skrytye-rashody (дата обращения: 12.05.2025)

15. Моки | JS: Продвинутое тестирование – ru.hexlet.io/ [Электронный ресурс]. URL: https://ru.hexlet.io/courses/js-advanced-testing/lessons/mocking/theory_unit (дата обращения: 11.01.2025)

16. Настройка CI/CD для GitLab-репозитория: работа с микросервисами / Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/850842/ (дата обращения: 07.01.2025)

17. Окупаемость бизнес-проекта: что это, как правильно рассчитать срок [Электронный ресурс]. URL: https://www.reg.ru/blog/okupaemost-biznes-proekta/ (дата обращения: 11.05.2025)

18. Организация автоматического запуска автотестов с использованием Downstream pipelines в GitLab CI / Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/simbirsoft/articles/715028/ (дата обращения: 14.01.2025)

19. Современная микросервисная архитектура: принципы проектирования / Хабр [Электронный ресурс]. URL: https://habr.com/ru/companies/innotech/articles/683550/ (дата обращения: 07.01.2025)

20. Что такое пакет SDK Bot Framework [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/azure/bot-service/bot-service-overview (дата обращения: 05.01.2025)