**«Санкт-Петербургский государственный электротехнический университет**

**«ЛЭТИ» им. В.И.Ульянова (Ленина)»**

**(СПбГЭТУ «ЛЭТИ»)**

|  |  |  |
| --- | --- | --- |
| **Направление** | 09.03.01 – Информатика и вычислительная техника | |
| **Профиль** | Системы автоматизированного проектирования | |
| **Факультет** | Компьютерных технологий и информатики | |
| **Кафедра** | Систем автоматизированного проектирования | |
| *К защите допустить* |  | |
| Зав. кафедрой, к.т.н., доцент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Бутусов Д.Н. |

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

БАКАЛАВРА

Тема: Разработка приложения для отслеживания истории болезни

|  |  |  |
| --- | --- | --- |
| Студент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Семейкин С.А. |
|  |
| Руководитель к.т.н., доцент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Бутусов Д.Н. |
|  |  |  |
| Консультант Ассистент каф. САПР | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Кулагин М.В. |
|  |  |  |
| Консультант Ассистент каф. ПЭ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
|  |  |  |
| Консультант по нормоконтролю | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |

Санкт-Петербург

2025

**ЗАДАНИЕ**

**на выпускную квалификационную работу**

|  |  |
| --- | --- |
|  | Утверждаю |
|  | Зав. кафедрой САПР, к.т.н., доцент |
|  | \_\_\_\_\_\_\_\_\_\_\_\_ Бутусов Д.Н. |
|  | «\_\_»\_апреля\_2025 г. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент: | Семейкин С.А. |  | Группа | 1301 |
| Тема работы: Разработка приложения для отслеживания истории болезни. | | | | |
| Место выполнения ВКР: кафедра САПР, СПбГЭТУ «ЛЭТИ» | | | | |

Исходные данные (технические требования):

В ходе разработки приложения для отслеживания истории болезни должны быть учтены следующие требования, которые позволят программе быть полезной и простой в использовании:

* Интерфейс должен быть интуитивно понятным и должен обеспечивать простоту и удобство использования
* Система должна быть устойчива к возможным нагрузкам при увеличении количества пользователей и скорости их запросов
* Данные пользователей должны быть конфиденциальными и не должны попасть третьим лицам, в том числе злоумышленникам
* Система должна иметь возможность автоматической проверки качества работы, для обеспечения скорости доставки обновлений без участия дополнительных затрат на тестирование
* Система должна быть протестирована на предмет непродуманных сценариев использования

Цель разработки – реализовать приложение, способное упростить процесс отслеживания истории болезни пользователя, в том числе включая возможность взаимодействия с врачом.

Функциональные требования к приложению:

1. Возможность добавления болезней в историю, включая произвольный период болезни, описание и симптомы.
2. Возможность просмотра медицинской карты, которая включает данные о пользователе и историю его болезней, в трёх форматах: веб-страница, текстовое описание и документ Microsoft Word.
3. Возможность добавить уведомления о приеме лекарств, указывая периодичность и продолжительность приема.
4. Возможность выбора языка интерфейса для пользователя.
5. Возможность редактирования данных пользователя после регистрации, в том числе изменение языка интерфейса.
6. Возможность добавления «текущих» болезней, с последующей возможностью «завершения» болезни.
7. Меню доктора для взаимодействия с выбранными пациентами, а так же добавление пациентам болезней, уведомлений о предстоящем приеме и просмотр карты пациента.

Список используемых технологий:

1. Python 3.12. Интерпретируемый язык с динамической строгой типизацией, поддерживающий возможности как функционального, так и объектно-ориентированного подхода к разработке, имеющий автоматическое управление память. Поддерживается компанией Python Software Foundation с 1991 года.
2. FastAPI. Библиотека, которая позволяет создавать пользовательские клиент-серверные системы с поддержкой асинхронного выполнения кода, что дает значительный прирост в скорости обработки запросов. В том числе предоставляет возможность использования принципа внедрения зависимостей, автоматическую генерацию документации.
3. AIOgram. Библиотека, упрощающая взаимодействие с Telegram Bot API, написанная на языке Python и поддерживающая асинхронное выполнение кода.
4. PostgreSQL. Система управления базами данных, предоставляющая инструменты для работы со структурированным языком запросов (SQL).
5. SQLAlchemy 2.0. Библиотека, позволяющая синхронизировать записи базы данных с объектами языка Python.
6. Nginx. Веб-сервер, поддерживающий HTTP запросы, в том числе SSL протокол.

Содержание ВКР: введение, основные функции приложения, проектирование архитектуры приложения, разработка серверной части приложения, разработка пользовательского интерфейса приложения, экономическое обоснование, заключение, список использованной литературы.

Перечень отчетных материалов: пояснительная записка, иллюстративный материал, материалы к презентации ВКР, иные отчетные материалы.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дополнительный раздел – Экономическое обоснование ВКР. | | | | |
| Дата выдачи задания | | Дата представления к защите | | |
| «\_\_»\_\_\_\_\_\_\_\_2025г. | | «\_\_»\_\_\_\_\_\_\_\_\_\_2025г. | | |
|  | |  | | |
| Студент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Семейкин С.А. |
|  | |
| Руководитель к.т.н., доцент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Бутусов Д.Н. |
|  |  | |  |
| Консультант  Ассистент каф. САПР | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Кулагин М.В. |
|  |  | |  |

|  |  |  |
| --- | --- | --- |
| Консультант  Ассистент каф. ПЭ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | ФИО |

**календарный план выполнения**

**выпускной квалификационной работы**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | Утверждаю | | | | | | | |
|  | | | Зав. кафедрой САПР, к.т.н., доцент | | | | | | | |
|  | | | \_\_\_\_\_\_\_\_\_\_\_\_Бутусов Д.Н. | | | | | | | |
|  | | | «\_\_»\_\_\_\_2025\_ г. | | | | | | | |
| Студент: | | Семейкин С.А. | | | |  | | Группа | 1301 | |
| Тема работы: Разработка приложения для отслеживания истории болезни. | | | | | | | | | | |
|  | | | | | | | | | | |
| № п/п | Наименование работ | | | | | | Срок выполнения | | |
| 1 | Обзор литературы по теме работы. | | | | | | 04.04 – 07.04 | | |
| 2 | Определение основного функционала приложения. | | | | | | 08.04 – 11.04 | | |
| 3 | Проектирование архитектуры приложения. | | | | | | 12.04 – 14.04 | | |
| 4 | Проектирование базы данных для хранения данных о болезнях и пользователях. | | | | | | 15.04 – 16.04 | | |
| 5 | Разработка серверной части приложения | | | | | | 17.04 – 22.04 | | |
| 6 | Разработка пользовательского интерфейса (Telegram bot). | | | | | | 23.04 – 25.04 | | |
| 7 | Реализация взаимодействия работы частей приложения, в том числе веб-сервера. | | | | | | 26.04 – 27.04 | | |
| 8 | Выполнение дополнительного раздела «экономическое обоснование ВКР». | | | | | | 01.05 – 06.05 | | |
| 9 | Тестирование приложения для отслеживания истории болезни | | | | | | 07.05 – 11.05 | | |
| 9 | Оформление пояснительной записки. | | | | | | 12.05 – 22.05 | | |
| 10 | Оформление иллюстративного материала. | | | | | | 23.05 – 25.05 | | |
| 11 | Прохождение предварительной зашиты на кафедре. | | | | | |  | | |
| 12 | Представление ВКР для проверки степени оригинальности пояснительной записки. | | | | | |  | | |
| 13 | Представление ВКР к защите. | | | | | |  | | |
|  | | | |  |  | | | | | |

|  |  |  |
| --- | --- | --- |
| Студент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Семейкин С.А. |
|  |
| Руководитель к.т.н., доцент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Бутусов Д.Н. |
|  |  |  |
| Консультант  Ассистент каф. САПР | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Кулагин М.В. |
|  |  |  |
| Консультант  Ассистент каф. ПЭ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | ФИО |

РЕФЕРАТ

Пояснительная записка включает в себя 59 стр., 32 рис., 8 табл., 13 ист.

УЧЁТ ЗАБОЛЕВАНИЙ, УВЕДОМЛЕНИЯ О ПРИЁМЕ ЛЕКАРСТВ, ВЗАИМОДЕЙСТВИЕ ПАЦИЕНТА И ВРАЧА, TELEGRAM-БОТ, FASTAPI, POSTGRESQL, SQLALCHEMY, AIOGRAM, DOCKER, PYTEST, ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС.

Цель данной работы – реализовать приложение, способное упростить процесс отслеживания истории болезни пользователя.

Разработка приложения выполняется с использованием языка Python 3.12 и его библиотеки FastAPI. Фреймворк SQLAlchemy дает возможность хранить данные в реляционной базе данных, с использованием СУБД PostgreSQL. Взаимодействие с пользователем происходит посредством Telegram бота, который реализован с помощью фреймворка Aiogram. Контейнеризация приложения выполнена при помощи Docker и docker-compose.

В процессе работы были спроектированы архитектурные решения, обеспечивающие масштабируемость и отказоустойчивость системы. Подробно описаны пользовательские сценарии, включая регистрацию, добавление болезней, настройку уведомлений и взаимодействие врача с пациентами.

Разработанная система обеспечивает возможность удобной регистрации пользователей, ведения истории болезней, назначения приёмов и отправки напоминаний. Пользователи получают доступ к функционалу напрямую через мессенджер Telegram, без необходимости устанавливать отдельные приложения. Врачи, в свою очередь, получают доступ к медицинским картам пациентов, могут назначать приёмы и контролировать процесс лечения.

Система ориентирована на простоту использования и возможность дальнейшего масштабирования за счёт модульной архитектуры и открытости к горизонтальному масштабированию.

ABSTRACT

The topic of this project is the development of an application that simplifies the process of tracking a user's medical history.

The application is built using Python 3.12 and the FastAPI framework. SQLAlchemy is used to interact with a relational database powered by PostgreSQL. User interaction is implemented through a Telegram bot developed using the Aiogram framework. The application is containerized using Docker and managed with docker-compose.

During the development process, architectural solutions were designed to ensure the system's scalability and fault tolerance. User scenarios are described in detail, including registration, adding diseases, setting up notifications, and doctor-patient interaction.

The developed system provides a convenient way for users to register, manage their medical history, schedule appointments, and receive notifications. Users access all functionality directly via Telegram without the need to install additional applications. Doctors, in turn, have access to patients' medical records, can schedule appointments, and monitor treatment progress.

The system focuses on ease of use and is designed for future scalability through modular architecture and support for horizontal scaling.

**ОГЛАВЛЕНИЕ**

[ПЕРЕЧЕНЬ ОСНОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ 12](#_Toc193986687)

[ВВЕДЕНИЕ 13](#_Toc193986688)

[Глава 1. Основные функции приложения 14](#_Toc193986689)

[1.1. Пользователи 14](#_Toc193986690)

[1.1.1. Врачи 15](#_Toc193986691)

[1.2. Болезни 15](#_Toc193986692)

[1.3. Уведомления 16](#_Toc193986693)

[1.3.1. Прием лекарств 16](#_Toc193986694)

[1.3.2. Прием у врача 17](#_Toc193986695)

[1.4. Карта болезней пациента 17](#_Toc193986696)

[1.5. Выводы 18](#_Toc193986697)

[Глава 2. Проектирование архитектуры приложения 19](#_Toc193986698)

[2.1. Описание выбранных технологий 19](#_Toc193986699)

[2.1.1. Кеширование и асинхронность 19](#_Toc193986700)

[2.1.2. Горизонтальное масштабирование и микросервисная архитектура 20](#_Toc193986701)

[2.1.3. Высокая доступность и отказоустойчивость 20](#_Toc193986702)

[2.1.4. Мониторинг и отладка 21](#_Toc193986703)

[2.2. Описание частей приложения 21](#_Toc193986704)

[2.3. Выводы 23](#_Toc193986705)

[Глава 3 Разработка серверной части приложения 24](#_Toc193986706)

[3.1. Проектирование базы данных 24](#_Toc193986707)

[3.1.1. Таблица users 24](#_Toc193986708)

[3.1.2. Таблица doctors 26](#_Toc193986709)

[3.1.3. Таблица diseases 26](#_Toc193986710)

[3.1.4. Таблица notifications 28](#_Toc193986711)

[3.1.5. Таблица notification\_times 29](#_Toc193986712)

[3.2. Сервис API 31](#_Toc193986713)

[3.2.1. Архитектура сервиса 31](#_Toc193986714)

[3.2.2. Router’ы 33](#_Toc193986715)

[3.2.2.1. UserRouter 33](#_Toc193986716)

[3.2.2.2. DoctorRouter 34](#_Toc193986717)

[3.2.2.3. DiseasesRouter 34](#_Toc193986718)

[3.2.2.4. NotificationsRouter 35](#_Toc193986719)

[3.2.3. Сервисы 35](#_Toc193986720)

[3.2.3.1. Сервис пользователей 36](#_Toc193986721)

[3.2.3.2. Сервис докторов 36](#_Toc193986722)

[3.2.3.3. Сервис болезней 36](#_Toc193986723)

[3.2.3.4. Сервис уведомлений 37](#_Toc193986724)

[3.2.4. Слой доступа к данным 38](#_Toc193986725)

[3.3. Тестирование 40](#_Toc193986726)

[Глава 4 Разработка пользовательского интерфейса приложения 43](#_Toc193986727)

[4.1. Регистрация пользователя 44](#_Toc193986728)

[4.2. Сценарий добавления болезни 46](#_Toc193986729)

[4.3. Просмотр активных болезней 47](#_Toc193986730)

[4.4. Сценарий просмотра истории болезней 48](#_Toc193986731)

[4.5. Меню уведомлений 49](#_Toc193986732)

[4.5.1. Сценарий добавления уведомления о приеме лекарств 50](#_Toc193986733)

[4.5.2. Просмотр списка уведомлений 50](#_Toc193986734)

[4.6. Просмотр данных пользователя 51](#_Toc193986735)

[4.7. Меню доктора 51](#_Toc193986736)

[4.7.1. Список свободных пациентов 52](#_Toc193986737)

[4.7.2. Список пациентов доктора 53](#_Toc193986738)

[4.7.3. Меню работы с пациентом 53](#_Toc193986739)

[4.7.4. Просмотр расписания доктора 54](#_Toc193986740)

[Глава 5 Экономическое обоснование ВКР 55](#_Toc193986741)

[Заключение 56](#_Toc193986742)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 58](#_Toc193986743)

# ПЕРЕЧЕНЬ ОСНОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

**БД** – это совокупность данных, которая хранится с определенной пользователем схемой, и используется для удовлетворения информационных потребностей пользователей (база данных).

**Бэкенд** – это логика приложения, которая реализована на стороне сервера.

**Фреймворк** – библиотека языка, предоставляющая высокоуровневый доступ к инструментам компонентов системы.

**HTML** – язык разметки, необходимый для структурирования контента и отображении его на веб-странице приложения.

**API (Application Programming Interface)** - описание способов и правил взаимодействия различных компьютерных программ между собой, например, клиента с сервером.

**Врач** – в рамках приложения, пользователь, наделенный администратором приложения возможностью взаимодействия с другими пользователями и просмотра их данных.

**MVP (Minimum viable product) –** Минимально жизнеспособный продукт. Другими словами – продукт, обладающий минимальными, но достаточными для удовлетворения первых потребителей функциями, в том числе отвечающий на заданные функциональные требования.

# ВВЕДЕНИЕ

На сегодняшний день автоматизация охватывает всё больше сфер жизни человека. Современные пользователи хотят иметь под рукой удобные инструменты для отслеживания состояния своего здоровья, при этом не тратя время на установку дополнительных приложений и сложные процедуры регистрации. Особенно актуальным становится упрощение коммуникации между врачом и пациентом, ведь все больше пользователей предпочитают взаимодействие в текстовой форме.

Именно эти требования легли в основу разработки Telegram-бота, предназначенного для удобной записи истории болезни. Выбор платформы объясняется её широкой распространённостью — мессенджером Telegram ежедневно пользуются миллионы людей, а значит, бот не доставит тягот в начале использования. Кроме того, благодаря этому подходу отпадает необходимость создавать отдельное мобильное приложение, что существенно ускоряет процесс разработки и снижает требования к устройствам пользователей.

Целью данной работы является проектирование и реализация удобного, функционального и масштабируемого сервиса, позволяющего пользователям вести учёт своих заболеваний и получать уведомления о приёмах, а врачам — работать с пациентами в рамках централизованной системы.

Объектом разработки является приложение для отслеживания пользователями своей истории болезней.

Предмет разработки — программная реализация Telegram-бота и серверной части, обеспечивающей работу с данными пользователей.

Таким образом, тема разработки является актуальной и востребованной, а созданный продукт потенциально может быть применён как в рамках личного использования, так и в небольших медицинских организациях.

# Глава 1. Основные функции приложения

Основная задача приложения – предоставить пользователям возможность удобно записывать информацию об их болезнях и визитах к врачу, а также врачам взаимодействовать с их пациентами.

Рассмотрим подробно все структурные единицы приложения.

## Пользователи

Большая часть пользователей разрабатываемого приложения – пользователи пациенты. Для получения доступа к функциям приложения любой пользователь, в том числе и врач, должен пройти регистрацию. В ходе регистрации он должен заполнить следующие поля своего профиля:

* Язык общения с ботом из предложенных вариантов: Русский и Английский
* Имя и фамилия. Поле не проходит дополнительных проверок, поэтому пользователь может указать любой текст, который будет идентифицировать его. Эти данные используются исключительно для отображения в карте пациента и для его идентификации доктором.
* Пол из предложенных вариантов: Мужчина и Женщина. Используется только для отображения в карте пациента.
* Вес. Данные в этом поле должны быть числом, не более 1000. Используется для отображения в карте пациента
* Рост. Данные в этом поле должны быть числом, не более 1000. Также используется для отображения в карте.

Данные пользователя может просмотреть только он и его врач. Во время использования приложения у пользователя есть возможность редактировать свои данные, для поддержания актуальности.

## Врачи

Врач по своей сути является обычным пользователем, но при этом обладает дополнительным функционалом, в рамках которого может устанавливать другим пользователям (пациентам) болезни и назначать приемы. Врач также может быть пациентом, несмотря на свою должность.

В рамках приложения список врачей устанавливается администратором и не может быть изменен напрямую из пользовательского интерфейса, что позволяет обеспечить безопасность данных пользователей и ограничить доступ к конфиденциальным данным о болезнях.

## Болезни

Болезнь может быть добавлена как пациентом для самого себя, так и врачом для своего пациента. В ходе добавления болезни пользователю необходимо предоставить следующую информацию:

* Название болезни. Пользователю будет предоставлен выбор из часто встречающихся болезней: ОРВИ, грипп, аллергия, перелом, сердечно сосудистые заболевания, но по своему усмотрению он может ввести название самостоятельно
* Симптомы и осложнения. В этом поле пользователю предлагается ввести любое описание болезни, которое ему может пригодиться в будущем, например, на приеме у врача.
* План лечения. Если есть, то пользователь может указать назначенные ему рекомендации от врача, либо самостоятельно продуманное лечение.
* Дата начала болезни. Дата, в которую пользователь «заболел», в формате ДД.ММ.ГГГГ.
* Дата окончания болезни. Если болезнь завершилась (то есть пользователь выздоровел), то он может ввести дату окончания болезни, в формате аналогичном дате начала. Иначе дату можно не указывать и после выздоровления отметить болезнь оконченной и указать дату.

## Уведомления

В приложении реализовано два вида уведомлений: о приеме лекарств и о назначенном приеме у врача. Первый вид уведомлений пользователь может добавлять самостоятельно. Второй же вид может добавить только врач, при назначении приема.

## Прием лекарств

Прием лекарств является одним из способов установки уведомлений. Для добавления этого вида уведомлений пользователь должен заполнить необходимые поля, для того чтобы приложение могло уведомлять его в нужное время и на протяжении необходимого числа дней:

* Название лекарства. В этом поле пользователь может указать любое словесное описание лекарства, которое будет отображено в сообщении-уведомлении от приложения.
* Продолжительность приема в днях. Пользователю будет предоставлен список из стандартных длительностей (3, 5, 7, 10, 14), но также у него есть возможность ввода произвольного числа дней, в течение которого приложение будет уведомлять пользователя.
* Частота приема, раз в день. Пользователю так же будет предоставлен список из стандартных частот приема (1,2,3,4) и у него будет возможность ввода числа самостоятельно. От этого значения зависит количество следующих полей.
* Время приема. В зависимости от частоты приема, пользователю нужно будет ввести N времен, в формате ЧЧ:ММ. Времена нужно вводить последовательно по возрастанию.

После заполнения всех необходимых полей пользователю будет отправлено сообщение об успешном добавлении уведомления с данными, которые он ввел.

## Прием у врача

Возможность добавлять прием есть только у врача. Этот вид уведомлений подразумевает два уведомления для пациента и одно опциональное уведомление для врача. Для создания такого типа уведомлений нужно заполнить следующие поля:

* Дата приема. Дата в формате ДД.ММ.ГГГГ, на которую планируется прием
* Время приема. Время в формате ЧЧ: ММ – на которое планируется прием.
* Выбор уведомлять ли себя (врача) о предстоящем приеме. Варианты – да или нет.

После заполнения всех полей врачу будет отправлено соответствующее сообщение об успешном добавлении. Если уведомление для себя было указано как нужное – врач сможет увидеть этот прием в списке своих приемов. Иначе там не появится уведомление.

## Карта болезней пациента

Карта болезней пациента формируется по его запросу и может быть получена либо пациентом для самого себя, либо врачом для любого из его пациентов. Она содержит данные, которые пользователь указал в своем профиле, а также список всех его болезней, включая даты и статус болезней.

Для удобства использования реализованы три варианта просмотра карты болезней – HTML страница, Microsoft Word документ и текст. Их содержимое не отличается, за исключением того, что в тексте перечисляются все болезни, без указания информации о пациенте, так как текстовый формат подразумевался как быстрый и короткий вариант просмотра карты, для предоставления карты на печать удобен формат MS Word, а для просмотра с ПК – веб-страница.

## Выводы

В данной главе были описаны все структурные единицы приложения, с помощью которых оно обеспечит пользователя требуемым функционалом. Заранее продуманная структура полей для заполнения позволила избежать ошибок и доработок по мере написания кода, что значительно ускорило процесс разработки

# Глава 2. Проектирование архитектуры приложения

# 

## Описание выбранных технологий

В первую очередь архитектура приложения будет зависеть от выбранных технологий. В настоящее время существует огромный выбор инструментов, которые предоставляют возможности быстрой разработки, и в то же время быструю и безотказную работу полученного продукта.

В рамках разрабатываемого проекта основными факторами, на которые стоит ориентироваться являются скорость разработки, возможность в будущем поддерживать программный код и масштабировать его. Одним из самых быстрых языков для реализации MVP является вышеупомянутый Python. Возможность поддержки обеспечивается высокой популярностью и распространенностью языка, согласно рейтингу TIOBE [1], Python занимает первую строчку по популярности с 2023 года.

Согласно статье «Архитектура высоконагруженных приложений» [2], ключевыми признаками высоконагруженных приложений являются: горизонтальное масштабирование, высокая доступность и отказоустойчивость, распределенные системы и микросервисная архитектура, кеширование и асинхронность, мониторинг и отладка.

Рассмотрим каждый из признаков подробнее, с указанием технологий для реализации.

## 2.1.1. Кеширование и асинхронность

Для реализации асинхронного API используется фреймворк FastAPI, который поддерживает кеширование с помощью сторонних библиотек, в том числе позволяя выносить хранение кеша в отдельное хранилище, реализуемое обособленным сервисом.

## 2.1.2. Горизонтальное масштабирование и микросервисная архитектура

Технология контейнеризации Docker [3] позволяет упаковывать приложения в контейнер, предоставляя необходимое окружение и зависимости, и затем запускать полученный его на целевой системе. С помощью инструмента docker-compose [3] каждый контейнер может быть отмасштабирован горизонтально, иными словами продублирован выбранное количество раз. Это позволит обеспечить быстрое и своевременное увеличение пропускной способности приложения, не занимая большого времени на разработку.

В том числе важной деталью является то, что все сервисы должны поддерживать горизонтальное масштабирование. Именно для этого и были выбраны библиотеки для разработки FastAPI и AIOgram, а так же СУБД PostgreSQL, ведь они имеют эту возможность без реализации дополнительного функционала.

## 2.1.3. Высокая доступность и отказоустойчивость

Вышеупомянутый инструмент docker-compose в комбинации c веб-сервером Nginx позволяют контролировать количество запущенных сервисов и в случае отказов некоторых из них – перезапускать их в автоматическом режиме.

Кроме того, Nginx предоставляет возможность изолировать пользователя от обращения к сервисам напрямую, с помощью технологии обратного прокси, транслируя запросы из внешней сети (от пользователя) во внутреннюю (серверную) сеть. Это позволяет в случае отказа одного из экземпляров сервиса перенаправить запрос пользователя на аналогичный сервис, при этом для человека это пройдет незаметно.

## 2.1.4. Мониторинг и отладка

Язык разработки Python предоставляет огромный выбор библиотек для отладки и мониторинга программных продуктов. В их числе инструменты для тестирования, профилирования, мониторинга в реальном времени и множество других технологий. В рамках приложения используется библиотека Pytest, которая позволяет реализовывать различные виды тестов, обеспечивая корректную работу приложения при разработке, и сводит к нулю возможность попадания кода с ошибкой на продуктовый сервер, где ошибки могут встретить пользователи.

Мониторинг приложения выполняется с помощью встроенных средств Docker для отслеживания состояния контейнеров, нагрузки на сервер и сеть.

Таким образом описанная выше архитектура с применением конкретных технологий отвечает всем поставленным требованиям.

## Описание частей приложения

Для более наглядного описания взаимодействия частей приложения обратимся к Рисунку 2.1

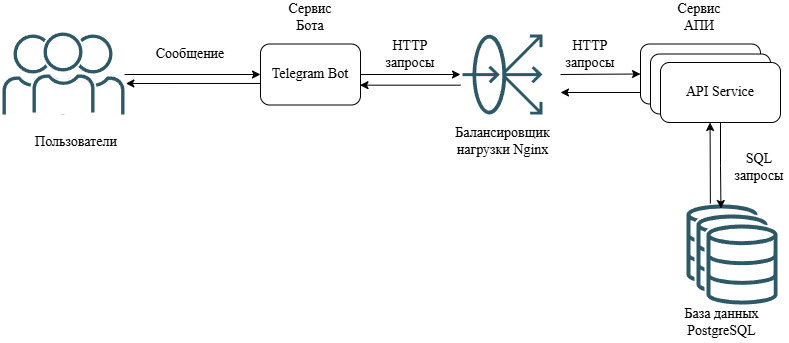


Рисунок 2.1 – диаграмма взаимодействия компонентов приложения

На Рисунке 2.1 изображен путь запроса пользователя по всей системе приложения. Он состоит из следующих стадий:

1. Взаимодействие человека с пользовательским интерфейсом (Telegram ботом). С помощью него происходит взаимодействие со всеми функциями приложения.
2. Сервис бота обрабатывает запросы пользователей и направляет их в балансировщик нагрузки.
3. Балансировщик нагрузки контролирует работоспособность реплик сервиса API, и перенаправляет запросы от бота напрямую к ним.
4. Сервис API выполняет всю обработку данных, собирает данные из базы данных или, в зависимости от запроса, записывает их в нее, и затем возвращает ответ на запрос.

Каждая из стадий ожидает ответа от следующего элемента, а затем возвращает свой ответ. Таким образом запрос пользователя проходит через все этапы проверки и обработки и возвращается обратно в сервис бота для ответа пользователю.

Представленная диаграмма демонстрирует возможность горизонтального масштабирования, которое подразумевалось ранее, благодаря использованию балансировщика нагрузки можно увеличивать количество реплик сервиса API, тем самым увеличивая его пропускную способность. Выбранная СУБД PostgreSQL также поддерживает горизонтальное масштабирование с помощью шардирования и репликации, поэтому в случае достижения лимита нагрузки на базу данных никаких доработок со стороны кода для увеличения пропускной способности базы данных не потребуется.

В том числе поддерживает возможность масштабирования и пользовательский интерфейс, выполненный на основе Telegram бота. Библиотека AIOgram позволяет увеличить количество одновременно работающих обработчиков запросов, что прямо пропорционально увеличит количество одновременных ответов пользователям.

В качестве будущих улучшений представленная система может быть расширена с помощью дополнительных точек взаимодействия пользователя с приложением, например, путем добавления веб-сайта или мобильного приложения, которые будут так же взаимодействовать с API посредством HTTP запросов.

## Выводы

В данном разделе были рассмотрены основные критерии выбора архитектуры разрабатываемого приложения, исходя из них были описаны необходимые критерии для технологий, а затем приведены используемые в проекте библиотеки и компоненты.

Так же была представлена диаграмма взаимодействия пользователя с системой, которая отражает внутреннюю структуру приложения.

В том числе были продуманы возможные пути улучшения системы путем горизонтального масштабирования и добавления новых точек взаимодействия с пользователем.

# Глава 3 Разработка серверной части приложения

## 3.1. Проектирование базы данных

Для реализации всех функциональных требований будет достаточно одной базы данных HealthDB, хранящей в себе данные о пользователях, болезнях, врачах, а также уведомлениях пользователей.

Для взаимодействия с базой данных, а также ее создания используется декларативных подход с использованием библиотеки языка Python – SQLAlchemy 2.0. Она позволяет описывать объекты базы данных в виде классов языка, что упрощает последующее взаимодействие с ними. Библиотека преобразует код объектов в SQL запросы и создает необходимые таблицы.

Для доставки обновлений в будущие версии приложения был использован механизм миграций. С помощью библиотеки Alembic можно генерировать файлы-миграции для моделей SQLAlchemy, что облегчает процесс обновления существующих баз данных под новые версии кода.

В следующих пунктах будет представлен именно SQL код полученных преобразований для универсальности и простоты понимания.

## 3.1.1. Таблица users

На рисунке 3.1 представлено строение таблицы users.

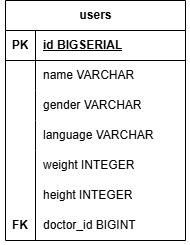


Рисунок 3.1 структура таблицы users

Описание таблицы:

1. id – это первичный ключ таблицы. Значение этого поля приходит в запросе создания пользователя от API сервиса, который в свою очередь получает его от бота. В качестве id берутся уникальные идентификаторы пользователей Telegram, что упрощает проверку и обработку значений. Тип данных для этого поля был выбран исходя из спецификации Telegram-API.
2. gender – поле, которое хранит информации о поле пользователя.
3. language – хранит в себе текущий выбранный язык интерфейса пользователя
4. weight – вес пользователя
5. height – Рост пользователя
6. doctor\_id – внешний ключ, который ссылается на users. Хранит в себе информацию о текущем докторе для пациента.

Листинг 3.1 – скрипт создания таблицы users

CREATE TABLE users (

id BIGSERIAL NOT NULL,

name VARCHAR NOT NULL,

gender VARCHAR NOT NULL,

language VARCHAR NOT NULL,

weight INTEGER NOT NULL,

height INTEGER NOT NULL,

doctor\_id BIGINT,

PRIMARY KEY (id),

FOREIGN KEY(doctor\_id) REFERENCES users (id)

)

## 3.1.2. Таблица doctors

На рисунке 3.2 представлено строение таблицы doctors.

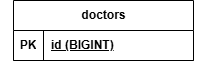


Рисунок 3.2 структура таблицы doctors

Описание таблицы:

1. id – поле, содержащее информацию об идентификаторе пользователя.

Все строки этой таблицы означают то, что пользователи с такими id являются докторами. Эта таблица вынесена отдельно от таблицы users для обеспечения безопасности доступа к личным данным, ведь редактировать ее может только администратор приложения, в отличие от таблицы users.

Листинг 3.2 – скрипт создания таблицы doctors

CREATE TABLE doctors (

id BIGSERIAL NOT NULL,

PRIMARY KEY (id)

)

## 3.1.3. Таблица diseases

На рисунке 3.3 представлено строение таблицы diseases.

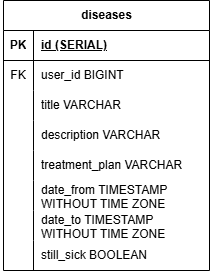


Рисунок 3.3 структура таблицы diseases

Описание таблицы:

1. id – первичный ключ, уникальный идентификатор болезни, тип данных Serial реализует создание поля, если в данных оно не было передано, путем автоинкримента.
2. user\_id – внешний ключ, ссылающийся на таблицу users. Так как каждая болезнь относится к 1 пользователю, то с помощью этого поля реализована связь «один ко многим».
3. title – название болезни, указанное пользователем
4. description – текстовое описание болезни, переданное пользователем
5. treatment\_plan – план лечения, который пользователь добавил для болезни
6. date\_from – дата заболевания.
7. date\_to – дата выздоровления. Если пользователь все еще болеет, то в этом поле устанавливается значение NULL.
8. still\_sick – служебное булево поле, которое означает что болезнь все еще идет.

Листинг 3.3 – скрипт создания таблицы diseases

CREATE TABLE diseases (

id SERIAL NOT NULL,

user\_id BIGINT NOT NULL,

title VARCHAR NOT NULL,

description VARCHAR NOT NULL,

treatment\_plan VARCHAR,

date\_from TIMESTAMP WITHOUT TIME ZONE NOT NULL,

date\_to TIMESTAMP WITHOUT TIME ZONE,

still\_sick BOOLEAN NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY(user\_id) REFERENCES users (id) ON DELETE CASCADE

)

## 3.1.4. Таблица notifications

На рисунке 3.4 представлено строение таблицы notifications.

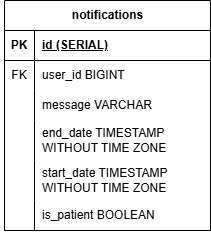


Рисунок 3.4 структура таблицы notifications

Описание таблицы:

1. id – первичный ключ, уникальный идентификатор уведомления, тип данных Serial реализует создание поля, если в данных оно не было передано, путем автоинкримента.
2. user\_id – внешний ключ, ссылающийся на таблицу users. Так как каждая болезнь относится к 1 пользователю, то с помощью этого поля реализована связь «один ко многим».
3. message – сообщение, которое будет отправлено пользователю в уведомлении. Формируется на стороне сервиса Telegram бота во время создания уведомления.
4. end\_date – дата, до которой включительно будет приходить это уведомление
5. start\_date – дата, начиная с которой уведомления будут приходить. Позволяет реализовывать уведомления на будущее, например, уведомление о приеме у врача через неделю.
6. is\_patient – служебное поле, указывает кем было добавлено уведомление. Если истинно, то пациент сам себе добавил уведомление, иначе – уведомление было создано доктором.

Листинг 3.4 – скрипт создания таблицы notifications

CREATE TABLE notifications (

id SERIAL NOT NULL,

user\_id BIGINT NOT NULL,

message VARCHAR,

end\_date TIMESTAMP WITHOUT TIME ZONE NOT NULL,

start\_date TIMESTAMP WITHOUT TIME ZONE,

is\_patient BOOLEAN DEFAULT 't' NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY(user\_id) REFERENCES users (id) ON DELETE CASCADE

)

## 3.1.5. Таблица notification\_times

На рисунке 3.5 представлено строение таблицы notification\_times

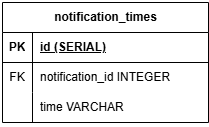


Рисунок 3.5 структура таблицы notification\_times

Описание таблицы:

1. id – первичный ключ, уникальный идентификатор времени уведомления, тип данных Serial реализует создание поля, если в данных оно не было передано, путем автоинкримента.
2. notification\_id – внешний ключ, ссылающийся на таблицу notifications. Так как к одному уведомлению может относиться несколько времен, то с помощью этого поля реализована связь «один ко многим».
3. Time – поле, отвечающее за время уведомления. Хранится в текстовом представлении, в формате ЧЧ:ММ.

Листинг 3.6. – скрипт создания таблицы notification\_times

CREATE TABLE notification\_times (

id SERIAL NOT NULL,

notification\_id INTEGER NOT NULL,

time VARCHAR NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY(notification\_id) REFERENCES notifications (id) ON DELETE CASCADE

)

В рамках проектирования базы данных было создано 5 таблиц, которые связаны между собой различными видами связи. Диаграмма связей таблиц представлена на Рисунке 3.6.

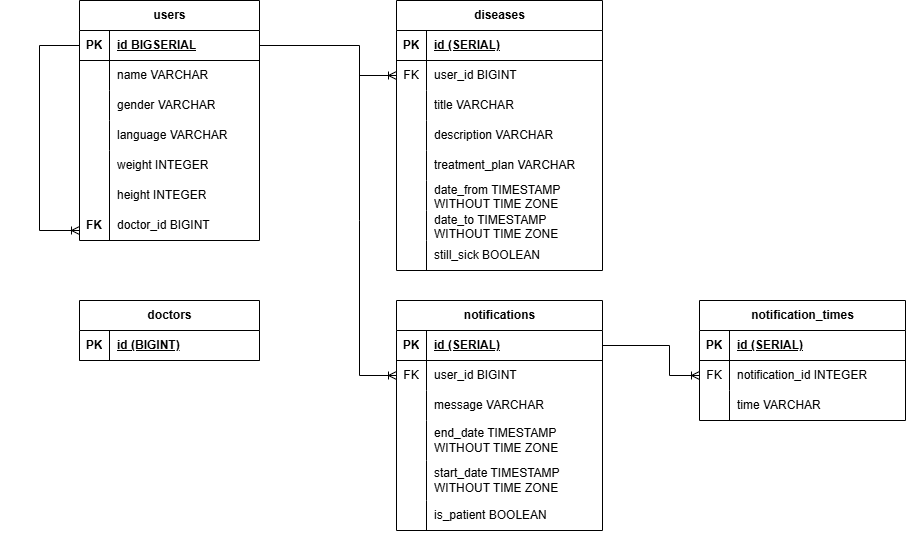


Рисунок 3.6 диаграмма связей таблиц БД HealthDB

## Сервис API

## Архитектура сервиса

Разработка любого приложения начинается с продумывания его архитектуры. В рамках данного проекта более оптимальным выбором будет тот подход, который может обеспечить достаточную скорость разработки, при этом не теряя в производительности и возможности поддержки программного кода.

Одним из таких паттернов является слоистая архитектура сервисов [5]. По концепции этого паттерна компоненты сервиса организуются в горизонтальные слои, каждый из которых выполняет свою обособленную роль. В основном используются 3 слоя: представления, бизнес-логики, доступа к данным.

Рассмотрим реализацию сервиса API послойно, обратившись к Рисунку 3.7.

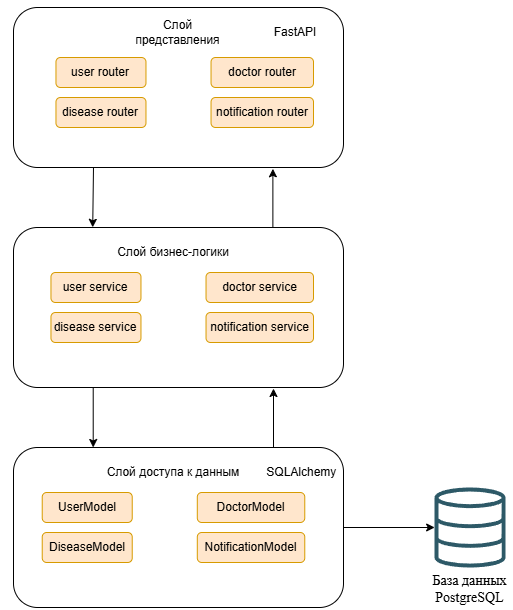


Рисунок 3.7 Архитектура сервиса API

Слой представления реализуется путем использования библиотеки FastApi и механизма переадресации запросов. Для каждой структурной единицы реализован свой роутер, что позволяет разделить ответственность и следовать принципам SOLID [6], а именно принципу единой ответственности.

Вся бизнес-логика, а именно проверка данных, обработка ошибок и фильтрация вынесены в одноименный слой, реализованный с помощью сервисов. Любой запрос из слоя представления вызывает метод у соответствующего сервиса, который обрабатывает его и подготавливает к обращению к слою данных. В том числе при ошибках работы с базой данных этот слой обработает их и предоставит на вышестоящий уже подготовленными для ответа пользователю.

И наконец, слой доступа к данным разработан с помощью библиотеки SQLAlchemy, которая позволяет декларативно описать все необходимые модели, а затем выполнять над ними различные операции по поиску, преобразованию и удалению данных.

## Router’ы

После получения сервисом запроса от клиента он попадает в роутеры, распределяясь в них исходя из URL запроса. Рассмотрим все обработчики и их конечные точки.

## UserRouter

Таблица 3.1. – конечные точки UserRouter

|  |  |  |
| --- | --- | --- |
| Метод | URL | Результат |
| GET | /users/ | List[dict] |
| POST | /users/ | dict |
| GET | /users/{user\_id} | dict |
| PATCH | /users/{user\_id} | dict |
| GET | /users/by\_doctor/{doctor\_id} | List[dict] |

1. GET /users/ - возвращает всех зарегистрированных пользователей из базы данных.
2. POST /users/ - добавляет пользователя с переданными параметрами в БД.
3. GET /users/{user\_id} – возвращает одного пользователя с переданным user\_id.
4. PATCH /users/{user\_id} – обновляет данные пользователя с переданным user\_id данными, которые были отправлены в теле запроса.
5. GET /users/by\_doctor/{doctor\_id} – возвращает всех пациентов доктора с идентификатором равным doctor\_id.

## DoctorRouter

Таблица 3.2. – конечные точки DoctorRouter

|  |  |  |
| --- | --- | --- |
| Метод | URL | Результат |
| GET | /doctor/ | List[int] |
| POST | /doctor/ | dict |

1. GET /doctor/ - получает список идентификаторов всех докторов из БД.
2. POST /doctor/ - добавляет в список докторов переданный в теле запроса идентификатор. Этот запрос доступен только администратору приложения и не может быть отправлен из пользовательского интерфейса.

## DiseasesRouter

Таблица 3.3 – конечные точки DiseasesRouter

|  |  |  |
| --- | --- | --- |
| Метод | URL | Результат |
| POST | /diseases/ | List[dict] |
| GET | /diseases/{disease\_id} | dict |
| GET | /diseases/for\_user/{user\_id} | List[dict] |
| PATCH | /diseases/mark\_as\_finished/{disease\_id} | dict |

1. POST /diseases/ - добавляет болезнь с переданными в теле запроса параметрами.
2. GET /diseases/{disease\_id} – возвращает конкретную болезнь из БД с указанным идентификатором.
3. GET /diseases/for\_user/{user\_id} – возвращает список болезней для пользователя с идентификатором равным user\_id.
4. PATCH /diseases/mark\_as\_finished/{disease\_id} – изменяет болезнь с переданным идентификатором, устанавливая дату завершения. Иными словами – завершает болезнь.

## NotificationsRouter

Таблица 3.4 – конечные точки NotificationsRouter

|  |  |  |
| --- | --- | --- |
| Метод | URL | Результат |
| GET | /notifications/ | List[dict] |
| POST | /notifications/ | dict |
| DELETE | /notifications/{notification\_id} | dict |
| GET | /notifications/for\_user/{user\_id} | List[dict] |
| GET | /notifications/schedule/{doctor\_id} | List[dict] |

1. GET /notifications/ - возвращает список всех уведомлений из БД.
2. POST /notifications/ - добавляет уведомление с переданными параметрами в теле запроса в базу данных.
3. DELETE /notifications/{notification\_id} – удаляет выбранное по идентификатору notification\_id уведомление.
4. GET /notifications/for\_user/{user\_id} – возвращает список всех уведомлений для конкретного пользователя с переданным user\_id.
5. GET /notifications/schedule/{doctor\_id} – возвращает расписание приемов врача с переданным идентификатором doctor\_id.

## Сервисы

Для каждого контроллера из слоя представления реализован одноименной сервис, который реализует слой бизнес-логики. Они позволяют отделить логику обработки данных от пользователя при запросе, и данных из БД при формировании ответа. А также предоставляют возможность переиспользования методов для уменьшения повторений в коде.

Рассмотрим методы каждого сервиса подробнее.

## Сервис пользователей

Описание методов сервиса приведено в таблице 3.5:

Таблица 3.5 – методы UserService

|  |  |  |
| --- | --- | --- |
| Название метода | Описание | Возвращаемый параметр |
| create\_user | Создание нового пользователя | dict |
| get\_user | Получение пользователя по ID | dict или None |
| get\_all\_users | Получение всех пользователей с возможной фильтрацией | list[dict] |
| update\_user | Частичное обновление данных пользователя | dict |
| get\_users\_by\_doctor | Получение всех пользователей, прикреплённых к врачу | list[dict] |

## Сервис докторов

Описание методов сервиса приведено в таблице 3.6:

Таблица 3.6 – методы DoctorService

|  |  |  |
| --- | --- | --- |
| Название метода | Описание | Возвращаемый параметр |
| get\_all\_doctors | Получение идентификаторов всех докторов из БД | list[int] |
| add\_doctor | Добавление нового доктора в базу данных | dict |

## Сервис болезней

Описание методов сервиса приведено в таблице 3.7:

Таблица 3.7 – методы DiseasesService

|  |  |  |
| --- | --- | --- |
| Название метода | Описание | Возвращаемый параметр |
| add\_disease | Добавление нового заболевания в базу данных | dict |
| get\_disease | Получение информации о заболевании по ID | dict |
| get\_all\_user\_diseases | Получение заболеваний пользователя с фильтрацией и форматированием (json, docx, html) | list | docx | html |
| mark\_disease\_as\_finished | Обновление заболевания: установка still\_sick=False и даты окончания | dict |

## Сервис уведомлений

Описание методов сервиса приведено в таблице 3.8

Таблица 3.8 – методы NotificationsService

|  |  |  |
| --- | --- | --- |
| Название метода | Описание | Возвращаемый параметр |
| add\_notification | Создание и сохранение нового уведомления | dict |
| get\_all\_notifications | Получение всех уведомлений с подгрузкой времён | List[dict] |
| delete\_notification | Удаление уведомления по ID | dict |
| get\_notifications\_for\_user | Получение уведомлений конкретного пользователя | List[dict] |
| get\_doctor\_schedule | Получение расписания врача по дате и статусу is\_patient=False | List[dict] |

## Слой доступа к данным

Как говорилось ранее, слой доступа к данным реализован с помощью декларативного подхода описания моделей БД с использованием библиотеки SQLAlchemy 2. Таким образом, описанные во второй главе сущности базы данных являются сгенерированными из их декларативных моделей.

Рассмотрим каждую из них подробнее, обратившись к листингам 3.1-3.4.

Листинг 3.1 – модель Disease

class Disease(Base):  
 \_\_tablename\_\_ = 'diseases'  
  
 id: Mapped[int] = mapped\_column(primary\_key=True, autoincrement=True)  
 user\_id: Mapped[int] = mapped\_column(ForeignKey("users.id", ondelete="CASCADE"))  
 user: Mapped["User"] = relationship(back\_populates="diseases")  
 title: Mapped[str]  
 description: Mapped[str]  
 treatment\_plan: Mapped[Optional[str]]  
 date\_from: Mapped[datetime.datetime]  
 date\_to: Mapped[Optional[datetime.datetime]]  
 still\_sick: Mapped[bool]

Листинг 3.2 – модель User

class User(Base):  
 \_\_tablename\_\_ = 'users'  
 id: Mapped[int] = mapped\_column(BigInteger, primary\_key=True)  
 diseases: Mapped[list["Disease"]] = relationship(back\_populates="user")  
 notifications: Mapped[list["Notification"]] = relationship(back\_populates="user")  
 name: Mapped[str]  
 gender: Mapped[str]  
 language: Mapped[str]  
 weight: Mapped[int]  
 height: Mapped[int]  
 doctor\_id: Mapped[int] = mapped\_column(BigInteger, ForeignKey("users.id"), nullable=True)

Листинг 3.3 – модели Notification

class Notification(Base):  
 \_\_tablename\_\_ = 'notifications'  
  
 id: Mapped[int] = mapped\_column(primary\_key=True, autoincrement=True)  
 user\_id: Mapped[int] = mapped\_column(ForeignKey("users.id", ondelete="CASCADE"))  
 user: Mapped["User"] = relationship(back\_populates="notifications")  
 message: Mapped[str] = mapped\_column(nullable=True)  
 end\_date: Mapped[datetime.datetime]  
 start\_date: Mapped[datetime.datetime] = mapped\_column(nullable=True)  
 time\_notifications: Mapped[list["NotificationTime"]] = relationship(back\_populates="notification", uselist=True)  
 is\_patient: Mapped[bool] = mapped\_column(server\_default='t')

class NotificationTime(Base):  
 \_\_tablename\_\_ = 'notification\_times'  
  
 id: Mapped[int] = mapped\_column(primary\_key=True, autoincrement=True)  
 notification\_id: Mapped[int] = mapped\_column(ForeignKey("notifications.id", ondelete="CASCADE"))  
 notification: Mapped[Notification] = relationship(back\_populates="time\_notifications")  
 time: Mapped[str]

Листинг 3.4 – модель Doctor

class Doctor(Base):  
 \_\_tablename\_\_ = "doctors"  
 id: Mapped[int] = mapped\_column(BigInteger, primary\_key=True)

## Тестирование

Большинство современных информационных систем разрабатываются по подходу TDD – test driven development, что в переводе означает «разработка через тестирование». Такой подход позволяет обеспечить высокое качество разрабатываемого продукта, но требует достаточно больших вложений на разработку и автоматизацию процесса тестирования.

Именно с целью обеспечения высокого качества и надежности приложения были написаны автоматические тесты бекенд-сервиса.

В рамках библиотеки **Pytest** поддерживается несколько видов тестов, каждый из которых решает свою задачу:

* **юнит-тесты**, которые проверяют работу отдельных функций или методов в изоляции;
* **интеграционные тесты**, предназначенные для проверки взаимодействия между компонентами системы;
* **функциональные тесты**, которые оценивают корректность выполнения пользовательских сценариев;
* **end-to-end (сквозные) тесты**, имитирующие поведение пользователя и проверяющие всю систему целиком.

Также в Pytest можно реализовывать **параметризованные тесты** для проверки функции на различных входных данных и **фикстуры** для настройки окружения перед выполнением тестов. Такой подход позволяет гибко покрыть все слои приложения и своевременно выявлять ошибки.

Рассмотрим один из реализованных тестов подробнее, обратившись к Листингу 3.5

Листинг 3.5 – автоматический тест создания пользователя

@pytest.mark.asyncio()  
async def test\_create\_user(db\_session, async\_client):  
 data = {  
 "id": 111,  
 "name": "Test",  
 "gender": "test",  
 "language": "ru",  
 "weight": 100,  
 "height": 120  
 }  
 response = await async\_client.post("/users/", json=data)  
 assert response.status\_code == status.HTTP\_201\_CREATED  
  
 result\_data = response.json()  
  
 assert result\_data == data

Этот тест проверяет, что запрос создания пользователя работает корректно, возвращает статус успешного ответа, а также проверяет что поля созданного пользователя совпадают с теми, которые были отправлены в запросе.

Всего в приложении написано 26 тестов, которые покрывают все четыре обработчика запросов. Они позволяют проверить работоспособность уже существующих функций приложения после внесения изменений. Таким образом при дальнейшей разработке программист может гарантировать, что его изменения не ломают логику, которой пользуется пользователь.

Тесты запускаются одной командой «pytest» в настроенном окружении для разработки. Результат проверки предоставляется в консоли, опционально он может быть сведен в HTML таблицу для наглядности результатов. Пример результата работы тестов приведен на Рисунке 3.8.

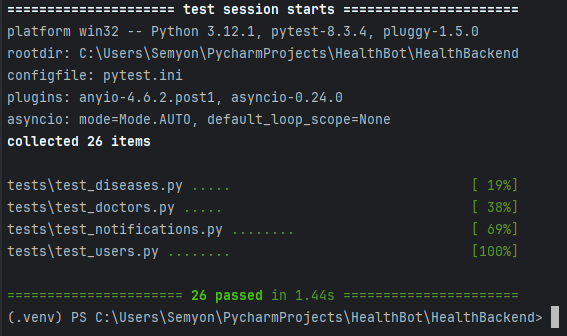


Рисунок 3.8 – результат работы тестов в приложении

# Глава 4 Разработка пользовательского интерфейса приложения

Для взаимодействия пользователя с приложением было выбрано использовать популярный в наше время мессенджер Telegram. Как сообщил его создатель, 19 марта 2024 года мессенджер достиг отметки в 1 миллиард пользователей, что несомненно означает всемирное признание удобства мессенджера. Именно исходя из этих фактов было принято решение использовать Telegram бота как средство связи с пользователями, ведь для использования приложения им не нужно будет дополнительно проходить регистрацию или устанавливать новые приложения.

Любой Telegram бот начинается с регистрации в корневом боте – официальном аккаунте мессенджера – Bot Father, создание бота приведено на рисунке 4.1. Он позволяет без участия третьих лиц создать собственного бота с выбранным названием и получить необходимые данные для работы с ним – API токен. Сохранность этого секретного ключа остается на совести разработчика, для этого в приложении используются переменные окружения, которые позволяют безопасно хранить секреты как в среде разработки, так и на продуктовом сервере.

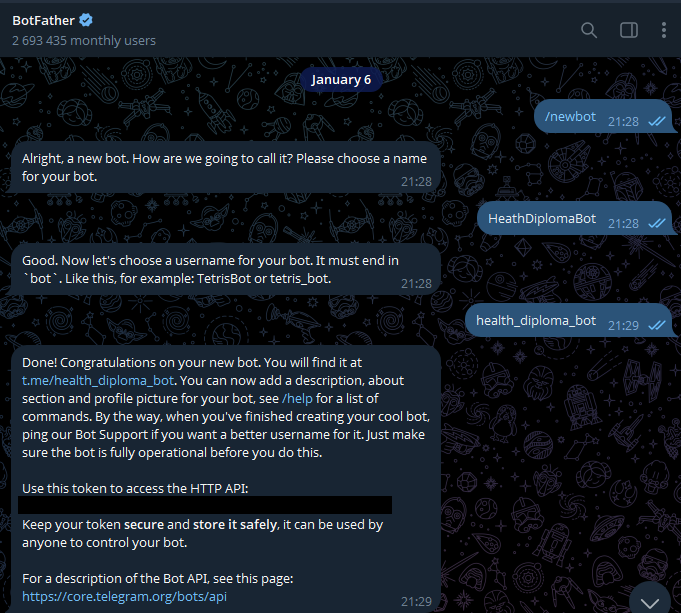


Рисунок 4.1 – создание бота с помощью BotFather

## Регистрация пользователя

Для получения возможности использования бота пользователю необходимо пройти быструю регистрацию, которая позволит в будущем формировать карту его болезней и его профиль.

Регистрация состоит из 5 этапов:

1. Выбор языка интерфейса. Все следующие вопросы бота будут на выбранном языке, изменить его можно будет после регистрации в профиле пользователя.

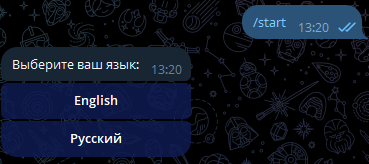


Рисунок 4.1 – выбор языка при регистрации

1. Ввод имени и фамилии. Это поле не проходит проверки, поэтому пользователь вправе указать информацию, которую считает достаточной для своей идентификации.

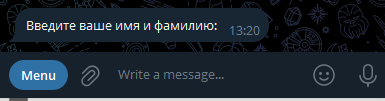


Рисунок 4.2 – ввод имени и фамилии при регистрации

1. Выбор пола – мужской или женский.

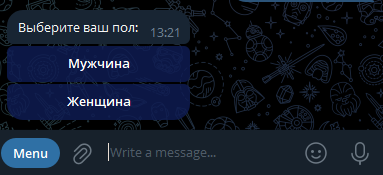


Рисунок 4.3 – выбор пола при регистрации

1. Ввод веса. Принимаются численные значения, с ограничением – от 0 до 1000.

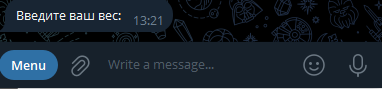


Рисунок 4.4 – ввод веса при регистрации

1. Ввод роста. Бот подсказывает, что нужно вводить рост в сантиметрах. Ограничение значений так же от 0 до 1000.

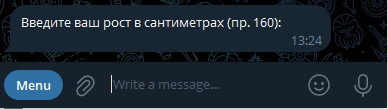


Рисунок 4.5 – ввод роста при регистрации

После заполнения всех полей регистрация завершается и происходит переход в главное меню, которое представлено на Рисунке 4.6:

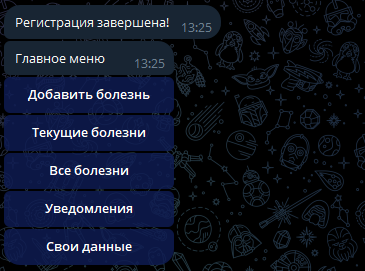


Рисунок 4.6 – главное меню пользователя

В следующих пунктах будут рассмотрены каждый из вариантов использования главного меню.

## Сценарий добавления болезни

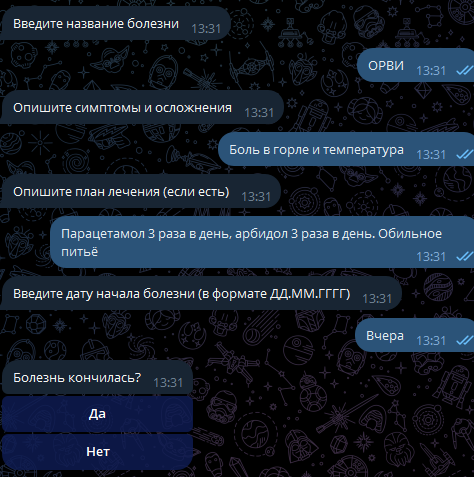


Рисунок 4.7 – сценарий добавления болезни.

Последним вопросом при добавлении болезни бот уточняет завершилась ли болезнь или нет. При выборе «нет» - болезнь сразу же добавится как «активная», в ином случае – бот предложит ввести дату выздоровления для добавления «завершенной» болезни. При успешном добавлении болезни будет отправлено соответствующее сообщение, приведенное на Рисунке 4.8.

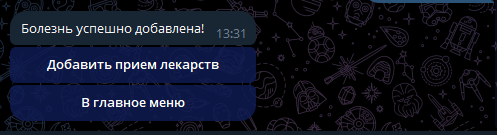


Рисунок 4.8 – Успешное добавление болезни

## Просмотр активных болезней

После добавления болезни как «активной» можно просмотреть ее в списке активных болезней, из главного меню. Список активных болезней представлен на Рисунке 4.9.

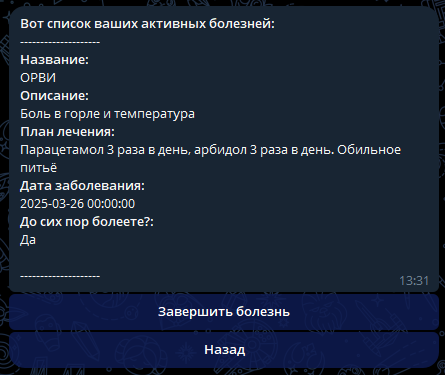


Рисунок 4.9 – список активных болезней

В списке активных болезней есть кнопка завершения болезни, которая предоставит выбор (если есть несколько активных болезней) какую именно болезнь пользователь хочет завершить.

## Сценарий просмотра истории болезней

Просмотр истории болезней начинается с выбора периода, за который пользователь, он представлен на Рисунке 4.10.

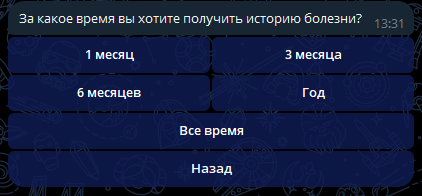


Рисунок 4.10 – выбор периода для просмотра истории болезней

Следующий выбор – способ представления истории болезней. Бот предоставляет возможность получения в трех разных форматах, они представлены на Рисунке 4.11.

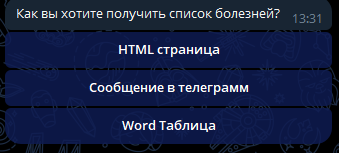


Рисунок 4.11 – выбор формата получения списка болезней

После нажатия на кнопку пользователю будет отправлен выбранный тип сообщения – либо текстовое сообщение, либо ссылка на HTML страницу, либо документ в формате .docx. Пример текстового сообщения приведен на Рисунке 4.12, пример веб-страницы приведен на Рисунке 4.13.

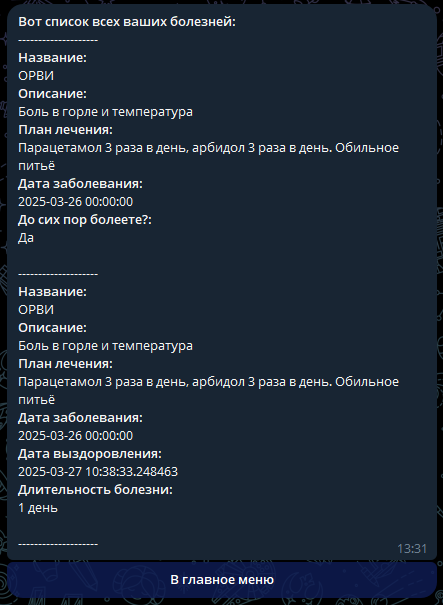


Рисунок 4.12 – список болезней в формате сообщения

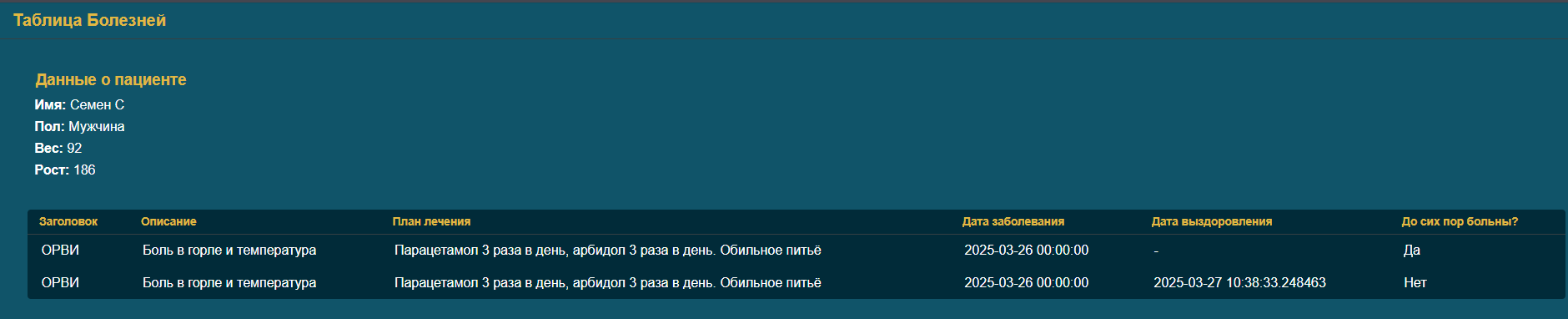


Рисунок 4.13 – пример истории болезней в формате веб-страницы

## Меню уведомлений

Управление уведомлениями пользователь можно производить из одноименного меню, представленного на Рисунке 4.14.

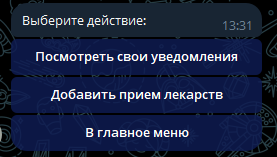


Рисунок 4.14 – меню уведомлений

## Сценарий добавления уведомления о приеме лекарств

Для добавления приема лекарств пользователю нужно ответить на ряд вопросов, среди которых – название лекарства, длительность и частота приема, а также время приема. После заполнения всех полей пользователю будет отправлено сообщение об успешном добавлении приема лекарств. Пример диалога добавления уведомления приведен на Рисунке 4.15.

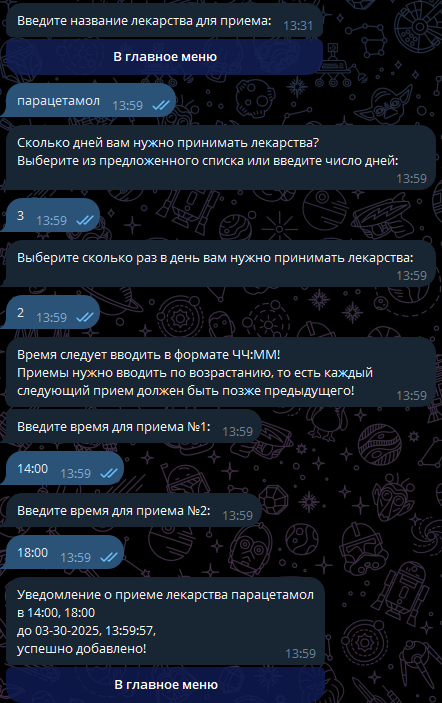


Рисунок 4.15 – сценарий добавления приема лекарств

## Просмотр списка уведомлений

После добавления уведомления пользователь может просмотреть их список в том же меню. Во время просмотра уведомлений предоставляется возможность удалить любое из них, выбрав его номер. Пример списка приведен на Рисунке 4.16

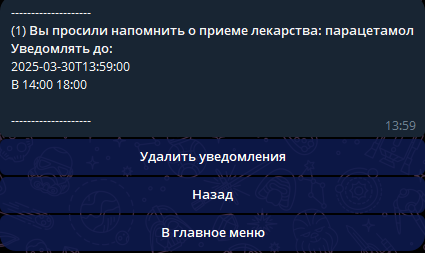


Рисунок 4.16 – просмотр списка уведомлений

## Просмотр данных пользователя

Из главного меню пользователь может просмотреть свои данные, а также редактировать их. В нем так же прописывается текущая болезнь, а если текущих болезней нет – статус «здоров».

Меню просмотра данных человека представлено на Рисунке 4.17.

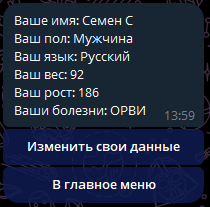


Рисунок 4.17 – просмотр данных пользователя

## Меню доктора

Использовав команду /doctor пользователь может попасть в меню доктора, но обязательным условием является наличие доступа к этому меню – идентификатор пользователя должен быть добавлен администратором в список докторов. Меню доктора приведено на Рисунке 4.18.

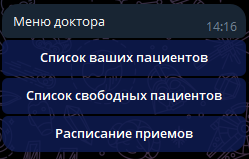


Рисунок 4.18 – меню доктора

## Список свободных пациентов

В списке свободных пациентов отображаются все пользователи приложения, у которых в текущий момент нет действующего врача. В нем же можно выбрать пациента. Сразу же после сделанного выбора, пациент перейдет из списка свободных в список пациентов врача.

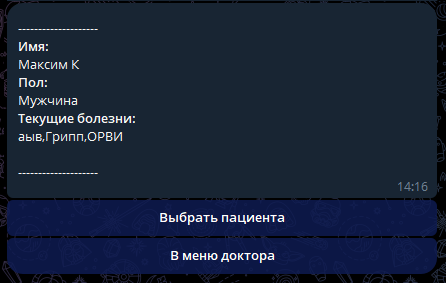


Рисунок 4.19 – список свободных пациентов

## Список пациентов доктора

После того, как доктор выберет пациента, он может просмотреть всех своих пациентов из меню. В нем можно выбрать конкретного по имени и начать работу с ним.

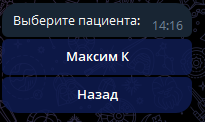


Рисунок 4.20 - меню выбора пациентов из списка "своих"

## Меню работы с пациентом

У врача есть несколько вариантов действий с пациентом, которые приведены на Рисунке 4.21. Рассмотрим каждый из них подробнее.

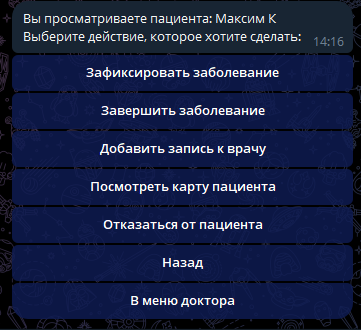


Рисунок 4.21 – меню действий с пациентом

1. Кнопка «зафиксировать заболевание» выполняет идентичный сценарий со сценарием добавления пользователем болезни для себя.
2. Кнопка «завершить заболевание» позволяет врачу завершить болезнь пользователя, аналогично логике завершения болезни из меню пользователя.
3. Кнопка «добавить запись к врачу» добавляет уведомление для пользователя за день до визита к врачу и в день визита к врачу. Опционально доктор может добавить уведомление в день визита для себя.
4. Кнопка «просмотреть карту пациента» работает аналогично просмотру карты из меню пациента, лишь с тем нюансом что доктору демонстрируется не своя карта, а карта просматриваемого человека.
5. Кнопка «отказаться от пациента» переводит пациента из «взятых» в «свободные», и он становится доступен для других врачей.

## Просмотр расписания доктора

Для удобства доктора реализован список его приемов, где он может узнать все свои записи. Важным нюансом этого списка является то, что, если при добавлении приема врач не добавит уведомление – в этом списке приема так же не будет, ведь доктор сам решил не делать этого. Пример списка представлен на Рисунке 4.22.

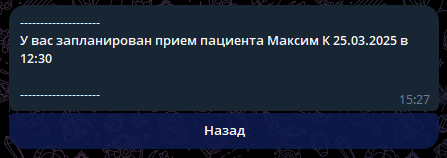


Рисунок 4.22 – список приемов доктора

# Глава 5 Экономическое обоснование ВКР

# Заключение

В рамках данной работы было спроектировано и реализовано приложение, предназначенное для удобной записи истории болезни пользователя в формате Telegram-бота. Приложение позволяет фиксировать сведения о болезнях, получать напоминания о приемах у врача и приеме лекарств, а также реализует возможность взаимодействия между врачом и пациентом.

На всех этапах разработки основной упор делался на удобство использования, безопасность хранения данных и расширяемость архитектуры. Заранее продуманная структура полей и логики взаимодействия позволила избежать значительных переделок в процессе реализации и обеспечила стабильную и предсказуемую работу системы.

Была тщательно продумана архитектура приложения, позволяющая обеспечить горизонтальное масштабирование и высокую отказоустойчивость. Использование современных технологий, таких как FastAPI, PostgreSQL, Docker, AIOgram и SQLAlchemy, позволило достичь высокой скорости разработки без потери качества и надёжности.

Особое внимание было уделено взаимодействию с пользователем. Благодаря использованию Telegram в качестве пользовательского интерфейса пользователю не требуется устанавливать дополнительное программное обеспечение, а регистрация сведена к минимуму.

Система позволяет врачам эффективно работать с пациентами, назначать приёмы и отслеживать динамику болезней, а пациентам — контролировать собственное состояние здоровья и своевременно получать напоминания.

Разработанное решение легко масштабируется и может служить основой для создания более крупной медицинской платформы с возможностью подключения мобильного приложения, веб-интерфейса и интеграции с медицинскими учреждениями.

Таким образом, поставленные задачи были успешно реализованы, а конечный результат соответствует всем заявленным требованиям.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. TIOBE Index for March 2025. Рейтинг языков программирования [Электронный ресурс]. Режим доступа: <https://www.tiobe.com/tiobe-index/> (Дата обращения: 25.03.2025).
2. Филисов Денис Александрович АРХИТЕКТУРА ВЫСОКОНАГРУЖЕННЫХ ПРИЛОЖЕНИЙ // Universum: технические науки. 2023. №10-1 (115). URL: https://cyberleninka.ru/article/n/arhitektura-vysokonagruzhennyh-prilozheniy (дата обращения: 25.03.2025).
3. Документация программной платформы Docker [Электронный ресурс]. Режим доступа: https://docs.docker.com/ (Дата обращения: 25.03.2025).
4. Telegram bot API [Электронный ресурс]. Режим доступа: https://core.telegram.org/bots/api (Дата обращения: 26.03.2025).
5. Mark Richards. Software Architecture Patterns: O’Reilly Media, Inc., 2015. 55c. (Дата обращения: 26.03.2025)
6. Ramachandrappa, Naveen Chikkanayakanahalli. "SOLID Design Principles in Software Engineering." *International Journal of Computer Trends and Technology* 72.9 (2024): 18-23. (Дата обращения: 26.03.2025)
7. Официальная документация по PostgreSQL [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs (Дата обращения: 27.03.2025)
8. Официальная документация фреймворка SQLAlchemy 2.0 [Электронный ресурс]. Режим доступа: <https://docs.sqlalchemy.org/en/20/index.html> (Дата обращения: 27.03.2025)
9. Официальная документация библиотеки Pydantic [Электронный ресурс]. Режим доступа: <https://docs.pydantic.dev/latest/> (Дата обращения 27.03.2025)
10. Официальная документация библиотеки FastAPI [Электронный ресурс]. Режим доступа: https://fastapi.tiangolo.com/ (Дата обращения 27.03.2025)
11. Официальная документация NginX [Электронный ресурс]. Режим доступа: https://nginx.org/ru/docs/ (Дата обращения 27.03.2025)
12. Официальная документация библиотеки PyTest [Электронный ресурс]. Режим доступа: https://docs.pytest.org/en/stable/index.html (Дата обращения 27.03.2025)
13. Test Driven Development [Электронный ресурс]. Режим доступа: <https://en.wikipedia.org/wiki/Test-driven_development> (Дата обращения: 27.03.2025).