### **Введение**

**Тема проекта**: Разработка Telegram-бота для изучения основ работы с базами данных.

**Цель проекта**: Создание Telegram-бота, который автоматизирует процесс обучения основам работы с базами данных. Бот предоставляет пользователям теоретические материалы, примеры SQL-запросов, интерактивные задания и тесты для проверки знаний. Программа рассчитана на использование в образовательных целях: для студентов, начинающих программистов и всех, кто хочет освоить базы данных.

Первый раздел носит название «Анализ задачи». В нем вы сможете ознакомиться с постановкой задачи, включающей исследование предметной области изучения баз данных и методов автоматизации обучения через Telegram. Также в разделе будут описаны входные данные (например, запросы пользователя, выбор тем) и выходные данные (справочная информация, примеры SQL-запросов, результаты тестов). В подразделе «Инструменты разработки» будет рассмотрена выбранная среда программирования Python, база данных PostgreSQL и Telegram API, а также установлены минимальные требования к техническим средствам для работы программы.

В разделе «Проектирование задачи» будут рассмотрены основные аспекты разработки Telegram-бота. Описывается структура хранения данных (например, таблицы для хранения тем, вопросов, примеров кода, результатов тестов) и способы их обработки. В этом разделе будет подробно описан функционал бота: обработка пользовательских команд, выбор темы или задания, предоставление теоретических материалов и проверка решений. Также будут составлены алгоритмы взаимодействия с базой данных, включая добавление, обновление и выборку данных.

«Реализация задачи» – это третий раздел пояснительной записки, в котором приводятся описание всех элементов программы, используемых для реализации функционала. Описываются основные функции Telegram-бота, например: предоставление справочной информации, выполнение SQL-запросов, проверка заданий, формирование отчетов по результатам тестов. Также в разделе будут представлены аннотации к файлам, модулям Python и структуре базы данных.

Четвертый раздел – «Тестирование». В этом разделе будет описано тестирование функционала бота. Проверка охватит обработку пользовательских запросов, корректность выполнения SQL-запросов, а также корректность проверки пользовательских решений. Будут рассмотрены возможные сценарии работы, включая ввод некорректных данных и моделирование ошибок.

В разделе «Применение» будет описано назначение программы, её область применения (например, использование в учебных заведениях, на курсах программирования или для самостоятельного обучения), а также условия эксплуатации.

«Заключение» подведет итоги разработки, кратко сформулирует задачу, представит выводы о проделанной работе, а также оценит эффективность Telegram-бота как инструмента для изучения баз данных.

В разделе «Литература» будет приведен список источников, использованных при разработке программы.

В приложениях к пояснительной записке будет представлен листинг кода Telegram-бота с комментариями, SQL-скрипты для создания базы данных и графическая часть, содержащая схемы структуры данных.

В приложении А будет представлена диаграмма вариантов использования.

В приложении Б будет представлена диаграмма Ганта.

В приложении В будет представлена структура приложения.

В приложении Г будет представлена диаграмма последовательности.

В приложении Д будет представлена диаграмма деятельности.

В приложении Е будет представлена модель данных.

В приложении Ж будут представлены тест-кейсы.

В приложении К будут представлены UX- и UI-прототипы.

### **Анализ задачи**

### **Постановка задачи**

#### Организационно-экономическая сущность задачи

**Наименование задачи**: Telegram-бот для изучения основ работы с базами данных.

**Цель разработки**: Создание бота, который упрощает процесс изучения баз данных, предоставляя теоретические материалы, примеры SQL-запросов, интера

**Назначение**: Данный программный продукт предназначен для студентов, начинающих программистов и всех, кто хочет освоить базы данных через практические задания.

**Периодичность использования**: По мере необходимости.

**Источники и способы получения данных**: Пользовательские запросы через Telegram-бот.

**Обзор существующих аналогичных решений**:

1. **SQLzoo** — интерактивная платформа для изучения SQL, предоставляющая задания по различным темам с проверкой решений;
2. **Mode Analytics SQL Tutorial** — платформа для изучения SQL с теорией, примерами и упражнениями;
3. **Kaggle Learn SQL** — онлайн-курсы для обучения SQL на реальных данных;

#### Функциональные требования

Описание перечня функций и задач, которые должен выполнять Telegram-**бот:**

**Пользователь:**

1. Получение справочной информации по темам баз данных;
2. Выполнение интерактивных заданий (например, написание SQL-запросов);
3. Прохождение тестов на знание теории;
4. Получение примеров запросов и объяснений;
5. Получение подсказок при решении заданий.

#### Описание входной, выходной и условно-постоянной информации

**Перечень исходной информации:**

1. Название темы (например, SELECT, JOIN, GROUP BY);
2. Описание задания;
3. Вопросы тестов;
4. Примеры запросов;
5. Подсказки и объяснения.

**Форма представления информации:**

Данные хранятся в базе данных PostgreSQL в формате таблиц. Доступ к ним осуществляется через запросы, обрабатываемые ботом.

Пример заполнения данных представлен в таблице 1:

Таблица 1 – Пример заполнения данных.

|  |  |  |  |
| --- | --- | --- | --- |
| **Тема** | **Вопрос** | **Пример** | **Подсказка** |
| **1** | **2** | **3** | **4** |
| ВЫБИРАТЬ | Напишите запрос для выборки всех строк из таблицы Students. | … \* FROM Students; | Используйте ключевое слово SELECT. |

**Перечень пользователей исходной информации:**

Пользователи: отправляют команды боту и получают доступ к справочной информации, заданиям и тестам.

**Перечень результатной информации**:

1. Справочная информация по выбранной теме;
2. Результаты выполнения заданий;
3. Итоговые баллы за прохождение теста;
4. Рекомендации по исправлению ошибок.

**Формы представления результатной информации**:

1. **На экране бота**: сообщения с текстом, примерами кода, результатами выполнения команд.
2. **В виде файлов**: отчеты о результатах обучения могут быть выгружены в формате docx или pdf.

**Периодичность и сроки представления**:

По запросу пользователя: предоставление данных происходит немедленно после ввода команды.

**Перечень пользователей результатной информации**:

Пользователи: получают результаты выполнения заданий и тестов, справочную информацию.

**Перечень условно-постоянной информации:**

1. **Справочник тем**: список всех доступных тем (например, SELECT, JOIN, INSERT);
2. **Справочник заданий**: информация о заданиях и их сложности;
3. **Справочник тестов**: вопросы, правильные ответы и пояснения.

**Формы представления**:

1. В виде таблиц базы данных;
2. Доступ через команды Telegram-бота.

#### Нефункциональные (эксплуатационные) требования

**Требования к применению**:

1. Интерфейс должен быть удобным и понятным даже для пользователей без технического опыта;
2. Использование стандартных элементов Telegram (команды, кнопки);
3. Возможность получения быстрого результата (справки, задания);
4. Документация для пользователей (список команд, описание возможностей).

**Требования к производительности**:

1. Бот должен отвечать на запросы пользователя в течение 1 секунды;
2. Корректная работа с базой данных, содержащей до 10 000 записей;
3. Поддержка многопользовательской работы без снижения производительности.

**Требования к реализации**:

1. Разработка на языке Python с использованием библиотеки psycopg2 для работы с PostgreSQL;
2. Поддержка Telegram API;
3. Среда разработки: Visual Studio Code.

**Требования к надежности**:

1. Бот должен работать стабильно, без сбоев при стандартной нагрузке;
2. Реализация функций сохранения состояния для восстановления работы после сбоя.

**Требования к интерфейсу**:

1. Удобный текстовый интерфейс Telegram;
2. Команды пользователя вводятся через текст или кнопки;
3. Вывод результатов в виде сообщений или файлов (например, результаты тестов в pdf).
4. **Диаграмма вариантов использования**

В качестве актеров представлены: гость и пользователь, которые взаимодействуют с системой. Им представлены следующие варианты использования:

1. Регистрация – гость регистрируется в системе. Включает ввод личных данных;
2. Просмотр списка тем – пользователь просматривает доступные темы для изучения;
3. Выбор темы – пользователь выбирает тему для выполнения заданий;
4. Выполнение заданий – пользователь решает задания по выбранной теме. Включает сохранение результата в базе данных и переход к следующему вопросу;
5. Просмотр личного счета – пользователь просматривает личный прогресс. Имеет расширение: сброс прогресса;
6. Просмотр таблицы лидеров – пользователь просматривает список лидеров на основе их достижений.

Разработанная диаграмма вариантов использования представлена в приложении А.

1. **Выбор стратегии разработки и модели жизненного цикла**

Для разработки электронного средства обучения по SQL следует выбрать стратегию разработки и модель жизненного цикла. Осуществляем выбор посредством составления таблиц:

Таблица 2 **–** Выбор модели жизненного цикла на основе характеристик требований

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **№ критерия** | **Критерии категории требований** | **Каскадная** | **V-образная** | **RAD** | **Инкрементная** | **Быстрого прототипирования** | **Эволюционная** |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| 1. | Являются ли требования к проекту легко определимыми и реализуемыми? | Да | Да | Да | Нет | Нет | Нет |
| 2. | Могут ли требования быть сформулированы в начале ЖЦ? | Да | Да | Да | Да | Нет | Нет |
| 3. | Часто ли будут изменяться требования на протяжении ЖЦ? | Нет | Нет | Нет | Нет | Да | Да |
| 4. | Нужно ли демонстрировать требования с целью их определения? | Нет | Нет | Да | Нет | Да | Да |
| 5. | Требуется ли проверка концепции программного средства или системы? | Нет | Нет | Да | Нет | Да | Да |
| 6. | Будут ли требования изменяться или уточняться с ростом сложности системы (программного средства) в ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 7. | Нужно ли реализовать основные требования на ранних этапах разработки? | Нет | Нет | Да | Да | Да | Да |

Вычисления: 5 за каскадную, 5 за V-образную, 4 за RAD, 6 за инкрементную, 2 за быстрого прототипирования и 2 за эволюционную.

Итог: На основе результатов заполнения табл. 2 подходящей является инкрементная модель.

Таблица 3 **–** Выбор модели жизненного цикла на основе характеристик команды разработчиков

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **№ критерия** | **Критерии категории команды разработчиков**  **проекта** | **Каскадная** | **V-образная** | **RAD** | **Инкрементная** | **Быстрого прототипирования** | **Эволюционная** |
| **1** | **2** | **3** | **4** | **5** | **7** | **8** | **9** |
| 1. | Являются ли проблемы предметной области проекта новыми для большинства разработчиков? | Нет | Нет | Нет | Нет | Да | Да |
| 2. | Являются ли инструментальные средства, используемые в проекте, новыми для большинства разработчиков? | Да | Да | Нет | Нет | Нет | Да |
| 3. | Изменяются ли роли участников проекта на протяжении ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 4. | Является ли структура процесса разработки более значимой для разработчиков, чем гибкость? | Да | Да | Нет | Да | Нет | Нет |
| 5. | Важна ли легкость распределения человеческих ресурсов проекта? | Да | Да | Да | Да | Нет | Нет |
| 6. | Приемлет ли команда разработчиков оценки, проверки, стадии разработки? | Да | Да | Нет | Да | Да | Да |

Вычисления: 3 за каскадную, 3 за V-образную, 4 за RAD, 3 за инкрементную, 4 за быстрого прототипирования и 3 за эволюционную.

Итог: На основе результатов заполнения табл. 3 подходящими являются RAD и инкрементная модели.

Таблица 4 **–** Выбор модели жизненного цикла на основе характеристик коллектива пользователей

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **№ критерия** | **Критерии категории коллектива пользователей** | **Каскадная** | **V-образная** | **RAD** | **Инкрементная** | **Быстрого прототипирования** | **Эволюционная** |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| 1. | Будет ли присутствие пользователей ограничено в ЖЦ разработки? | Да | Да | Нет | Да | Нет | Да |
| 2. | Будут ли пользователи оценивать текущее состояние программного продукта (системы) в процессе разработки? | Нет | Нет | Нет | Да | Да | Да |
| 3. | Будут ли пользователи вовлечены во все фазы ЖЦ разработки? | Нет | Нет | Да | Нет | Да | Нет |
| 4. | Будет ли заказчик отслеживать ход выполнения проекта? | Нет | Нет | Нет | Нет | Да | Да |

Вычисления: 2 за каскадную, 2 за V-образную, 2 за RAD, 3 за инкрементную, 2 за быстрого прототипирования и 3 за эволюционную.

Итог: На основе результатов заполнения табл. 4 подходящей является модель инкрементная и эволюционная.

Таблица 5 **–** Выбор модели жизненного цикла на основе характеристик типа проектов и рисков

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **№ критерия** | **Критерии категории типов проекта и рисков** | **Каскадная** | **V-образная** | **RAD** | **Инкрементная** | **Быстрого прототипирования** | **Эволюционная** |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| 1. | Разрабатывается ли в проекте продукт нового для организации направления? | Нет | Нет | Нет | Да | Да | Да |
| 2. | Будет ли проект являться расширением существующей системы? | Да | Да | Да | Да | Нет | Нет |
| 3. | Будет ли проект крупно- или среднемасштабным? | Нет | Нет | Нет | Да | Да | Да |
| 4. | Ожидается ли длительная эксплуатация продукта? | Да | Да | Нет | Да | Нет | Да |
| 5. | Необходим ли высокий уровень надежности продукта проекта? | Нет | Да | Нет | Да | Нет | Да |
| 6. | Предполагается ли эволюция продукта проекта в течение ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 7. | Велика ли вероятность изменения системы (продукта) на этапе сопровождения? | Нет | Нет | Нет | Да | Да | Да |
| 8. | Является ли график сжатым? | Нет | Нет | Да | Да | Да | Да |
| 9. | Предполагается ли повторное использование компонентов? | Нет | Нет | Да | Да | Да | Да |
| 10. | Являются ли достаточными ресурсы (время, деньги, инструменты, персонал)? | Нет | Нет | Нет | Нет | Да | Да |

Вычисления: 4 за каскадную, 3 за V-образную, 7 за RAD, 5 за инкрементную, 7 за быстрого прототипирования и 5 за эволюционную.

Итог: На основе результатов заполнения табл. 5 подходящей является RAD и быстрого прототипирования модели.

Общие вычисления: 13 за каскадную, 14 за V-образную, 10 за RAD, 18 за инкрементную, 14 за быстрого прототипирования и 15 за эволюционную.

Общий итог: в итоге заполнения табл. 2 – 5 наиболее подходящей является инкрементная модель

### **Инструменты разработки**

Для разработки **Telegram-бота с использованием Python и PostgreSQL** будет выбрана среда разработки **PyCharm** , которая является одной из самых популярных и актуальных сред для разработки на Python. Она поддерживает все необходимые инструменты для создания бота, включая интеграцию с PostgreSQL и возможность отладки.

Разработка будет производиться на таких языках программирования, как:

1. **Python** — основной язык программирования для создания Telegram-бота. Python широко используется благодаря своей простоте и мощным библиотекам, таким как python-telegram-bot, которые облегчают создание ботов для Telegram;
2. **SQL** — для взаимодействия с базой данных PostgreSQL. SQL будет использоваться для написания запросов к базе данных, обработки данных и выполнения необходимых операций с таблицами.

Иные инструменты, используемые при разработке и написании сопутствующей документации:

1. **PostgreSQL** — система управления базами данных (СУБД), которая будет использоваться для хранения информации о пользователях бота, их запросах и результатах. PostgreSQL был выбран за свою надежность, масштабируемость и поддержку сложных запросов;
2. **DBeaver** — инструмент для работы с PostgreSQL и другими СУБД, который будет использоваться для администрирования базы данных, выполнения запросов и анализа данных;
3. **Telegram API** — для интеграции с платформой Telegram. Это API предоставляет все необходимые средства для создания и управления ботами, а также для взаимодействия с пользователями;
4. **Postman** — инструмент для тестирования API и отправки запросов к серверу. Используется для проверки правильности работы API Telegram и взаимодействия с сервером;
5. **Git** — система контроля версий для отслеживания изменений в коде и совместной работы с командой разработчиков;
6. **Microsoft Office Word** — для написания документации к проекту, включая описание архитектуры, функционала и пользовательского интерфейса;
7. **Draw.io** — для создания схем и диаграмм, таких как UML-диаграммы, которые помогут визуализировать структуру бота и его взаимодействие с пользователями;
8. Figma — для создания прототипов интерфейсов, планирования пользовательских взаимодействий и разработки дизайна. Figma удобна для совместной работы в реальном времени;
9. Visio — для создания сложных диаграмм и схем, таких как блок-схемы или диаграммы взаимодействий, которые помогают визуализировать архитектуру и логику работы системы;
10. Visual Studio Code (VSC) — легкий и мощный редактор кода, который будет использоваться для написания скриптов на Python, работы с различными файлами конфигурации и интеграции с Git. VSC поддерживает большое количество расширений и плагинов для улучшения производительности разработки.

Разработка проекта будет происходить на компьютере с следующими параметрами:

1. **Процессор:** AMD Ryzen 5 4600H
2. **Объем оперативной памяти:** 16.00 GB
3. **Объем места на жестком диске:** 512 GB
4. **Видеокарта:** NVIDIA® GEFORCE RTX 3050
5. **Операционная система:** Windows 11 Pro

Однако программа не очень требовательна к системным ресурсам и способна работать на любых современных компьютерах.

1. **Разработка плана работы над проектом**

**Диаграмма Ганта** (Gantt chart) — это визуальный инструмент для планирования и управления проектами, представляющий собой горизонтальную гистограмму, где задачи отображаются в виде полос, соответствующих временным интервалам их выполнения.

#### Основные элементы:

1. **Ось времени:** горизонтальная шкала для отображения временных рамок;
2. **Задачи:** полосы, представляющие виды работ, их начало, завершение и длительность;
3. **Вехи:** значимые точки проекта, синхронизирующие задачи;
4. **Проценты завершения:** индикатор выполнения задач;
5. **Зависимости:** связи между задачами, указывающие на их последовательность.

#### Преимущества:

1. Позволяет наглядно увидеть весь проект и его задачи;
2. Помогает отслеживать сроки и этапы выполнения;
3. Обеспечивает контроль зависимостей между задачами;
4. Поддерживает гибкость в управлении проектами.

#### Недостатки:

1. Не отображает значимость или ресурсоемкость задач;
2. Становится сложной для крупных проектов.

#### Применение:

Диаграммы Ганта широко используются в различных сферах: **IT**, маркетинг, строительство, здравоохранение, образование, финансовый сектор и др.

#### Как создать диаграмму Ганта:

1. Составьте список задач с указанием сроков;
2. Определите их длительность и последовательность;
3. Нарисуйте горизонтальные полосы на временной шкале;
4. Укажите зависимости между задачами.

#### Заключение:

Диаграмма Ганта остается одной из самых востребованных методик в управлении проектами, благодаря своей наглядности и простоте. Она помогает командам эффективно планировать рабочие процессы, распределять задачи, соблюдать сроки и минимизировать риски. Современные инструменты позволяют автоматизировать создание таких диаграмм, делая их доступными для проектов любого масштаба — от небольших командных задач до масштабных многоуровневых проектов.

Используя диаграмму Ганта, менеджеры проектов могут значительно улучшить коммуникацию внутри команды, более точно отслеживать прогресс выполнения задач, выявлять критические пути и своевременно реагировать на изменения в графике. Несмотря на существующие ограничения, такие диаграммы по праву считаются стандартом де-факто в управлении проектами и остаются важным элементом профессионального арсенала как в традиционных, так и в гибких методологиях.

Разработаную диаграмму Ганта можно рассмотреть в приложении Б.

1. **Проектирование задачи**
2. **Разработка структуры приложения, системы меню, навигации**

Что это такое: Схема приложения — это визуальное представление структуры вашего проекта, включая основные его компоненты, взаимодействие между ними и процессы, происходящие в приложении. Это помогает понять, как данные передаются между различными частями системы, а также на каких этапах взаимодействуют пользователи и какие операции выполняются.

Мое приложение — это Telegram-бот, предназначенный для обучения SQL с элементами тестирования.

1. Регистрация пользователей:
   * Пользователи регистрируются в боте, вводя своё полное имя и возраст.
   * После регистрации появляется главное меню с тремя кнопками: Список тем, Лидеры, Мой счёт
2. Список тем:
   * Пользователь может выбрать одну из пяти тем, по каждой из которых есть теория.
   * После изучения теории, пользователю предлагается пройти тест.
3. Тестирование:
   * Пользователю отправляется два вопроса с примерами.
   * Бот сверяет ответы с базой данных и сообщает, правильный ли ответ.
   * После второго вопроса показывается итоговый результат с количеством правильных ответов.
4. Лидеры:
   * Показан список пользователей с их местом и количеством правильных ответов.
5. Мой счёт:
   * Пользователь может просматривать свою статистику и, если нужно, сбросить количество правильных ответов.
6. База данных:
   * Хранит информацию о пользователях, их прогрессе, вопросах тестов, и лидерах.

Бот работает с PostgreSQL и использует таблицы для хранения информации о пользователях, прогрессе, задачах и результатах тестов.

Разработаную схему приложения можно рассмотреть в приложении В.

1. **Разработка UML-диаграмм**
2. **Диаграмма последовательности**

Диаграмма последовательности – это UML-диаграмма, которая показывает взаимодействие объектов в системе в определённой временной последовательности.  
На диаграмме представлены:

1. Объекты, обозначенные прямоугольниками с подчеркнутыми именами;
2. Сообщения (вызовы методов), обозначенные линиями со стрелками;
3. Возвраты результатов – пунктирными линиями со стрелками.  
   Временная ось на диаграмме указывает порядок вызова сообщений.

На диаграмме отображён процесс отображения личного счета.

Диаграмму последовательности можно рассмотреть в приложении Г.

1. **Диаграмма деятельности**

Диаграмма деятельности – это UML-диаграмма, иллюстрирующая последовательность действий или поток управления в системе.

Основные элементы диаграммы:

1. Действия или шаги процесса;
2. Переходы между действиями;
3. Условные ветвления и циклы.

Диаграмма используется для описания логики выполнения процессов.

Диаграмму деятельности можно рассмотреть в приложении Д.

1. **Модель данных**

Модель данных описывает структуру данных, их атрибуты, а также взаимосвязи между сущностями в системе.

Включает:

1. Сущности – объекты или концепции, представленные таблицами в базе данных;
2. Атрибуты – свойства или характеристики сущностей (например, поля таблиц);
3. Связи – отношения между сущностями (например, связь "один ко многим").

Модели данных помогают проектировать базы данных и обеспечивать структурированное хранение информации.

Модель данных можно рассмотреть в приложении Е.

1. **Разработка пользовательского интерфейса**

Важным этапом проектирования данного программного продукта является создание удобного и интуитивно понятного интерфейса для обучения SQL.

Для разработки дизайна использовались современные, минималистичные подходы с акцентом на ясность и читабельность, чтобы пользователю было комфортно работать с материалом. Визуальные элементы интерфейса выполнены в нейтральных и мягких цветах, что способствует продолжительной работе без утомления глаз.

Целостность интерфейса обеспечивается группировкой всех функций по тематическим разделам. Это позволяет пользователю легко находить нужные инструменты для обучения, уменьшает время на поиск и повышает эффективность освоения материала.

Прототипы интерфейса были разработаны с использованием платформы Figma:

1. UX-прототипы отображают логику взаимодействия пользователя с системой и пути выполнения задач.
2. UI-прототипы демонстрируют внешний вид интерфейса.

С прототипами UX можно ознакомиться по ссылке: https://www.figma.com/design/k894Ie8oK6OlXX4fBf9zio/TG\_bot\_UX?node-id=0-1&t=di0ZvMXg7jN6z3i9-1.  
С прототипами UI можно ознакомиться по ссылке: https://www.figma.com/design/w2cextsUi4hgRLECWa2ZXT/TG\_Bot\_UI?node-id=0-1&t=YqubuHFXA4nP1Hha-1.

Электронное средство обучения по SQL предоставляет пользователю удобный доступ к теоретическим материалам и практическим заданиям, снижая затраты времени и усилий на освоение баз данных.

1. **Реализация**
2. **Руководство программиста**

### Программный продукт разработан на языке программирования Python с использованием библиотеки TeleBot.

Эта библиотека позволяет создавать телеграм-боты на языке Python и предоставляет функционал для обработки команд, взаимодействия с пользователем, работы с клавиатурами и кнопками. Для работы с базой данных использовалась система управления базами данных PostgreSQL, а для управления и визуализации базы данных – DBeaver. Код проекта писался в текстовом редакторе Visual Studio Code.

### Основные части кода и их назначение

#### **Подключение к базе данных**

Отвечает за установление связи между ботом и PostgreSQL.

conn = psycopg2.connect(

dbname="postgres",

user="alex",

password="1234",

host="localhost",

port="5432"

Назначение:

* Настраивает подключение к базе данных.
* Используется для выполнения SQL-запросов.

#### **Регистрация пользователя**

Функция добавляет нового пользователя в таблицу users.

def register\_user(chat\_id, username, full\_name, age):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

INSERT INTO users (chat\_id, username, full\_name, age, registration\_date)

VALUES (%s, %s, %s, %s, NOW())

""", (chat\_id, username, full\_name, age))

connection.commit()

connection.close()

Назначение:

* Заполняет данные нового пользователя.
* Используется, когда пользователь впервые взаимодействует с ботом.

#### **Обработка команд Telegram**

Функции отвечают за действия при нажатии кнопок или вводе команд.

@bot.message\_handler(commands=['start'])

def send\_welcome(message):

bot.send\_message(message.chat.id, "Добро пожаловать в обучающий бот по SQL!")

show\_main\_menu(message)

@bot.message\_handler(commands=['help'])

def show\_help(message):

bot.send\_message(message.chat.id, "Список доступных команд: /start, /help, /topics")

Назначение:

* Реализует стартовое взаимодействие с пользователем.
* Показывает помощь и основные команды.

#### **Работа с заданиями**

Функция получает данные о заданиях из базы данных и отображает их пользователю.

def show\_task(user\_id, topic\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

SELECT question, example FROM tasks WHERE topic\_id = %s LIMIT 1

""", (topic\_id,))

task = cursor.fetchone()

connection.close()

if task:

bot.send\_message(user\_id, f"Вопрос: {task[0]}\nПример: {task[1]}")

else:

bot.send\_message(user\_id, "Задания для этой темы отсутствуют.")

Назначение:

* Показывает пользователю вопрос и пример по выбранной теме.

#### **Проверка ответа пользователя**

Проверяет корректность введенного ответа и обновляет прогресс.

def check\_answer(user\_id, sub\_id, user\_answer):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

SELECT answer FROM tasks WHERE sub\_id = %s

""", (sub\_id,))

correct\_answer = cursor.fetchone()[0]

if user\_answer.strip().lower() == correct\_answer.strip().lower():

bot.send\_message(user\_id, "Верно!")

increment\_user\_score(user\_id)

else:

bot.send\_message(user\_id, f"Неверно. Правильный ответ: {correct\_answer}")

connection.close()

Назначение:

* Сравнивает ответ пользователя с правильным.
* Обновляет счет в таблице leaderboard.

#### **Отображение таблицы лидеров**

Функция выводит текущий рейтинг пользователей.

def show\_leaderboard(chat\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

SELECT username, correct\_answers FROM leaderboard

ORDER BY correct\_answers DESC LIMIT 10

""")

leaders = cursor.fetchall()

connection.close()

leaderboard = "Таблица лидеров:\n"

for rank, (username, score) in enumerate(leaders, 1):

leaderboard += f"{rank}. {username} - {score} баллов\n"

bot.send\_message(chat\_id, leaderboard)

Назначение:

* Показывает рейтинг лучших участников.

#### **Основное меню**

Отвечает за выбор тем, отображение помощи и переход к тестам.

def show\_main\_menu(message):

markup = types.ReplyKeyboardMarkup(resize\_keyboard=True)

markup.add("Темы", "Лидеры", "Помощь")

bot.send\_message(message.chat.id, "Выберите действие:", reply\_markup=markup)

Назначение:

* Организует навигацию пользователя по боту.

#### **Логика переходов между заданиями**

Обеспечивает переход к следующему вопросу или теме.

def go\_to\_next\_task(user\_id, current\_topic\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

SELECT id FROM tasks WHERE topic\_id = %s AND id > (

SELECT MAX(task\_id) FROM user\_progress WHERE user\_id = %s

) LIMIT 1

""", (current\_topic\_id, user\_id))

next\_task = cursor.fetchone()

if next\_task:

show\_task(user\_id, next\_task[0])

else:

bot.send\_message(user\_id, "Вы завершили все задания по этой теме.")

connection.close()

Назначение:

* Автоматически предлагает следующее задание.

#### **Добавление прогресса пользователя**

Функция записывает прогресс пользователя после выполнения задания.

def add\_user\_progress(user\_id, topic\_id, sub\_id, is\_correct):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

INSERT INTO user\_progress (user\_id, topic\_id, sub\_id, is\_correct, completion\_date)

VALUES (%s, %s, %s, %s, NOW())

""", (user\_id, topic\_id, sub\_id, is\_correct))

connection.commit()

connection.close()

Назначение:

* Обновляет таблицу user\_progress.
* Используется для отслеживания успешности выполнения заданий.

#### **Сброс прогресса пользователя**

Функция удаляет текущий прогресс и обнуляет счет.

def reset\_user\_score(user\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

DELETE FROM user\_progress WHERE user\_id = %s

""", (user\_id,))

cursor.execute("""

UPDATE leaderboard SET correct\_answers = 0 WHERE user\_id = %s

""", (user\_id,))

connection.commit()

connection.close()

Назначение:

* + Сбрасывает прогресс выполнения заданий.
  + Очищает счетчик правильных ответов в таблице leaderboard.

#### **Отображение тем**

Функция показывает список доступных тем для изучения.

def show\_topics(chat\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

SELECT id, name, description FROM topics

""")

topics = cursor.fetchall()

connection.close()

response = "Доступные темы:\n"

for topic\_id, name, description in topics:

response += f"{topic\_id}. {name}: {description}\n"

bot.send\_message(chat\_id, response)

Назначение:

* + Выводит перечень тем из таблицы topics.
  + Позволяет пользователю выбрать тему для прохождения.

#### **Получение имени пользователя**

Извлекает полное имя пользователя из таблицы users.

def get\_full\_name(user\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

SELECT full\_name FROM users WHERE user\_id = %s

""", (user\_id,))

full\_name = cursor.fetchone()[0]

connection.close()

return full\_name

Назначение:

* + Предоставляет полное имя для персонализации сообщений.

#### **Получение возраста пользователя**

Получает возраст из таблицы users.

def get\_age(user\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

SELECT age FROM users WHERE user\_id = %s

""", (user\_id,))

age = cursor.fetchone()[0]

connection.close()

return age

Назначение:

* + Используется для анализа целевой аудитории и персонализации.

#### **Обработка кнопок тем**

Реализует действия при выборе темы.

@bot.callback\_query\_handler(func=lambda call: call.data.startswith("topic\_"))

def handle\_topic\_button(call):

topic\_id = int(call.data.split("\_")[1])

bot.send\_message(call.message.chat.id, f"Вы выбрали тему с ID: {topic\_id}")

show\_task(call.message.chat.id, topic\_id)

Назначение:

* + Сопоставляет выбранную тему с её ID.
  + Запускает первый вопрос по теме.

#### **Отображение справки**

Функция показывает список доступных команд и подсказок.

def show\_help(chat\_id):

help\_text = """

Доступные команды:

/start - начать работу с ботом.

/help - показать справку.

/topics - посмотреть доступные темы.

"""

bot.send\_message(chat\_id, help\_text)

Назначение:

* + Информирует пользователя о функционале бота.

#### **Обработка кнопки "Лидеры"**

Реализует действие при нажатии кнопки отображения лидеров.

@bot.message\_handler(func=lambda message: message.text == "Лидеры")

def handle\_leaderboard\_button(message):

show\_leaderboard(message.chat.id)

@bot.message\_handler(func=lambda message: message.text == "Лидеры") def handle\_leaderboard\_button(message): show\_leaderboard(message.chat.id)

Назначение:

* + Вызывает функцию show\_leaderboard.
  + Отображает таблицу лидеров.

#### **Переключение между заданиями**

Переходит к следующему вопросу в выбранной теме.

def go\_to\_next\_task(user\_id, current\_topic\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

SELECT id FROM tasks WHERE topic\_id = %s AND id NOT IN (

SELECT sub\_id FROM user\_progress WHERE user\_id = %s

) LIMIT 1

""", (current\_topic\_id, user\_id))

next\_task = cursor.fetchone()

if next\_task:

show\_task(user\_id, next\_task[0])

else:

bot.send\_message(user\_id, "Вы завершили все задания по этой теме.")

connection.close()

Назначение:

* Переходит к следующему доступному заданию

#### **Проверка состояния прогресса**

Проверяет, завершил ли пользователь текущую тему.

def handle\_next\_topic(user\_id, current\_topic\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

SELECT COUNT(\*) FROM tasks WHERE topic\_id = %s

""", (current\_topic\_id,))

total\_tasks = cursor.fetchone()[0]

cursor.execute("""

SELECT COUNT(\*) FROM user\_progress WHERE user\_id = %s AND topic\_id = %s

""", (user\_id, current\_topic\_id))

completed\_tasks = cursor.fetchone()[0]

connection.close()

if completed\_tasks == total\_tasks:

bot.send\_message(user\_id, "Вы завершили эту тему! Выберите другую.")

else:

go\_to\_next\_task(user\_id, current\_topic\_id)

Назначение:

* Оценивает прогресс выполнения темы.

#### **Увеличение счёта**

Функция добавляет очки за правильный ответ.

def increment\_user\_score(user\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

UPDATE leaderboard SET correct\_answers = correct\_answers + 1 WHERE user\_id = %s

""", (user\_id,))

connection.commit()

connection.close()def increment\_user\_score(user\_id):

connection = connect\_to\_db()

cursor = connection.cursor()

cursor.execute("""

UPDATE leaderboard SET correct\_answers = correct\_answers + 1 WHERE user\_id = %s

""", (user\_id,))

connection.commit()

connection.close()

Назначение:

* + Обновляет счет пользователя в таблице leaderboard.

#### 3.1.1 Организация данных

Проект состоит из следующего файла:

main.py – основной файл для запуска телеграм-бота, содержащий весь функционал и логику работы программы.

#### Организация базы данных

В качестве СУБД используется PostgreSQL. Основными таблицами базы данных являются:

1. **users** – содержит информацию о зарегистрированных пользователях:

* user\_id: целое число, ключевое поле (идентификатор пользователя);
* chat\_id: целое число (идентификатор чата в Telegram);
* username: текст (имя пользователя в Telegram);
* full\_name: текст (полное имя пользователя);
* age: целое число (возраст пользователя);
* registration\_date: дата (дата регистрации пользователя).

1. **user\_progress** – хранит прогресс пользователей:

* id: целое число, ключевое поле (идентификатор записи);
* user\_id: целое число (идентификатор пользователя);
* topic\_id: целое число (идентификатор темы);
* sub\_id: целое число (идентификатор подтемы);
* is\_correct: bool тип (правильность ответа);
* completion\_date: дата (дата завершения задания).

1. **leaderboard** – таблица лидеров:

* user\_id: целое число, ключевое поле (идентификатор пользователя);
* correct\_answers: целое число (количество правильных ответов).

1. **tasks** – содержит задания для пользователей:

* id: целое число, ключевое поле (идентификатор задания);
* sub\_id: целое число (идентификатор подтемы);
* question: текст (текст задания);
* example: текст (пример выполнения задания);
* answer: текст (правильный ответ).

1. **topics** – список доступных тем:

* id: целое число, ключевое поле (идентификатор темы);
* name: текст (название темы);
* description: текст (описание темы).

1. **Тестирование**
2. **Тесты на использование**

При разработке данной программы многие возникающие ошибки и недоработки были исправлены на этапе реализации проекта. После завершения испытания реализации программы было проведено тщательное функциональное тестирование. Функциональное тестирование должно гарантировать работу всех элементов программы в автономном режиме.

Тестирование представлено в таблице 6.