1 Введение

Данный отчет описывает процесс и результат разработки бота для мессенджера Telegram в рамках задания "Персональный менеджер задач с веб-интерфейсом".

Целью работы являлась разработка бота, который помогает пользователям управлять личными списками дел. Бот должен поддерживать основные CRUD-операции (Create, Read, Update, Delete) для управления задачами, обеспечивать хранение данных в базе данных и предоставлять интуитивно понятный интерфейс взаимодействия.

2 Описание функциональности

Разработанный бот предоставляет пользователям следующий набор команд для управления задачами:

- /**start** — выводит приветственное сообщение со списком доступных команд. Это начальная точка взаимодействия с ботом.
- /**help** — выводит сообщение со всеми доступными командами с коротким описанием их.
- /**add [текст задачи]** — добавляет новую задачу в список дел пользователя. В свою очередь, бот подтверждает успешное добавление и присваивает задаче уникальный ID.
- /**list** — отображает список всех невыполненных задач. Для удобства пользователя список реализован с пагинацией (по 5 задач на страницу) и интерактивными кнопками управления.
- /**done [ID]** — помечает задачу с указанным ID как выполненную. Задача перестает отображаться в общем списке.
- /**delete [ID]** — полностью удаляет задачу с указанным ID из базы данных, следовательно, и из списка задач.

Кроме того, бот корректно обрабатывает синтаксические ошибки (например, ввод команды без аргументов) и сообщает о неизвестных командах, игнорируя обычные сообщения, которые не начинаются с символа команды "/".

2.1 Демонстрация работы

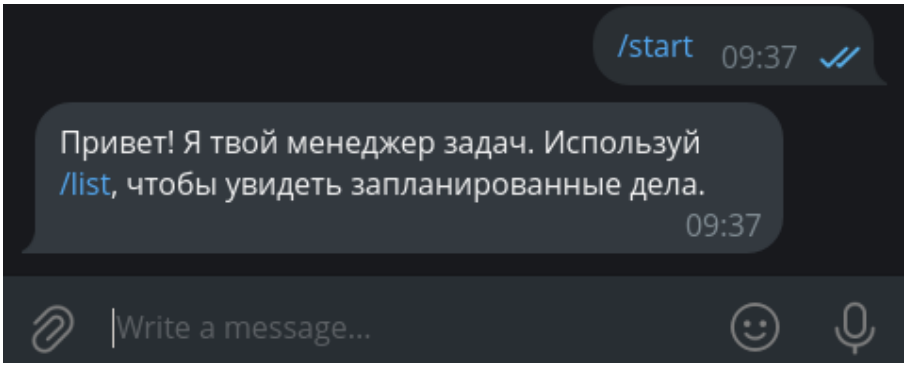

Рис. 1: Начало работы с ботом: команда /start

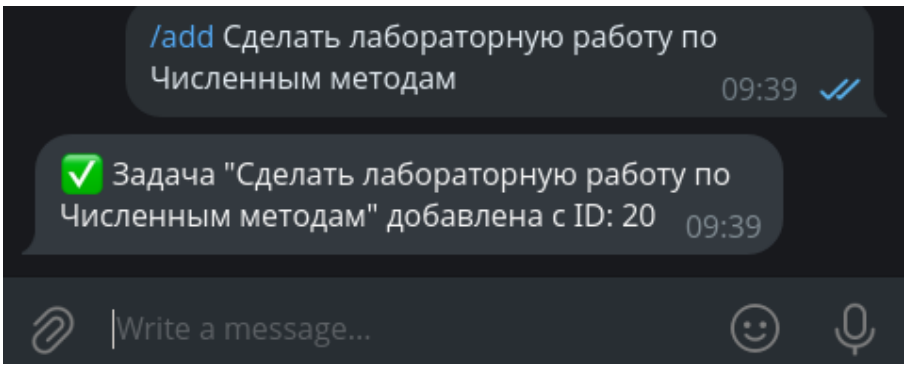

Рис. 2: Добавление новых задач

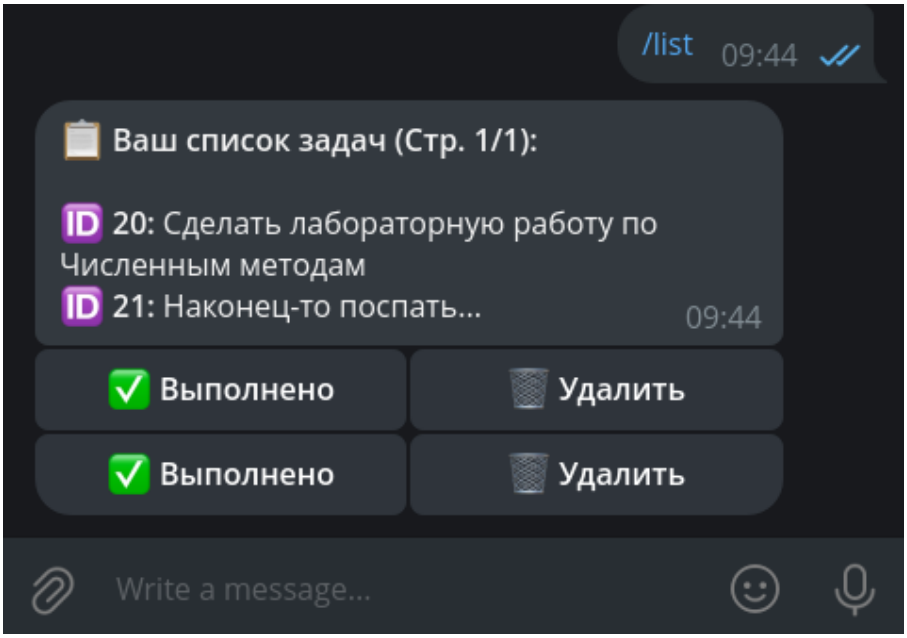

Рис. 3: Просмотр списка задач с пагинацией и кнопками управления

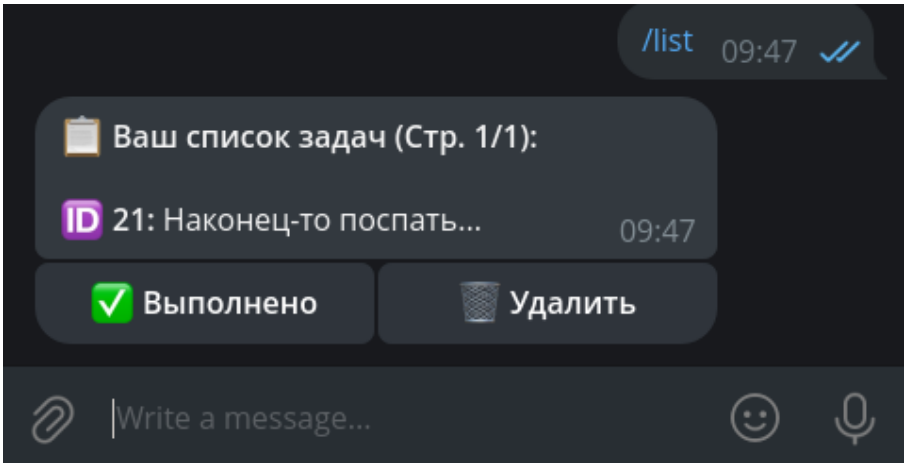

Рис. 4: Реакция на нажатие кнопки: сообщение обновляется

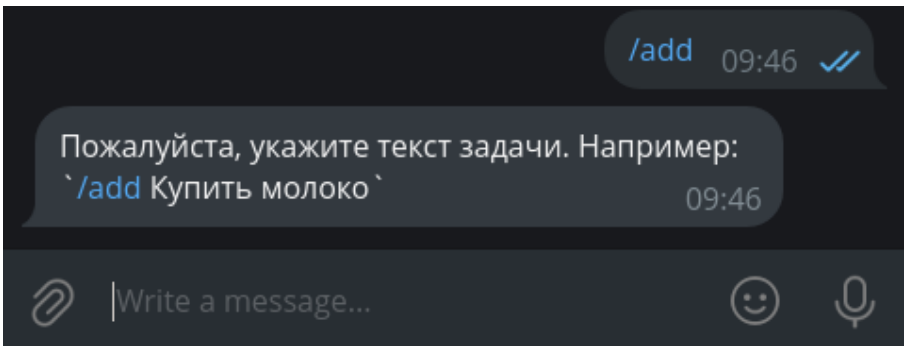

Рис. 5: Обработка некорректного ввода

3 Технический стек и архитектура

Проект реализован на платформе Node.js. Были использованы следующие технологии:

- **Node.js** — среда выполнения JavaScript.
- **node-telegram-bot-api** — библиотека для взаимодействия с Telegram Bot API.
- **SQLite (через библиотеку `sqlite3`)** — легковесная реляционная база данных для хранения задач. Выбор обусловлен простотой настройки и отсутствием необходимости в отдельном сервере БД.
- **.env** — модуль для управления переменными окружения, что позволяет безопасно хранить токен бота отдельно от исходного кода (токен задаётся в переменную $TELEGRAM\_BOT\_TOKEN$).

Архитектура проекта следует принципу разделения ответственности:

- **bot.js** — основной файл, отвечающий за логику бота: обработку команд, отправку сообщений и взаимодействие с пользователем.
- **database.js** — модуль, инкапсулирующий всю работу с базой данных (подключение, создание таблиц, выполнение CRUD-операций). Этот модуль ничего не знает о Telegram и предоставляет только интерфейс для работы с данными.

4 Обеспечение чистоты и безопасности кода

Одним из ключевых требований задания было обеспечение чистоты и безопасности кода. Эта задача была решена следующим образом:

4.1 Защита от SQL-инъекций

Для предотвращения атак типа «SQL-инъекция» все запросы к базе данных выполняются с использованием параметризованных запросов (prepared statements). Данные, полученные от пользователя, передаются в SQL-запрос не напрямую (через конкатенацию строк), а в виде отдельного массива параметров. Драйвер базы данных сам заботится об их безопасном экранировании.

```
const markTaskDone = (taskId, userId) => {
  const sql = `UPDATE tasks SET is_done = 1 WHERE id = ? AND user_id = ?`;
  return new Promise((resolve, reject) => {
    db.run(sql, [taskId, userId], function (err) {
      if (err) {
        reject(new Error('...'));
      } else {
        resolve(this.changes);
      }
    });
  });
};
```

Листинг 1: Пример безопасного параметризованного запроса в `database.js`

Такой подход полностью исключает возможность выполнения вредоносного SQL-кода, внедренного в данные от пользователя.

4.2 Чистота кода

Код структурирован и разделен на логические модули. Использование асинхронных функций с синтаксисом `async/await` делает код более читаемым и простым для поддержки по сравнению с традиционными callback-функциями.

5 Заключение

В ходе выполнения работы был успешно разработан Telegram-бот «Персональный менеджер задач». Бот полностью реализует требуемую функциональность CRUD, обеспечивает надежное хранение данных и предоставляет удобный интерактивный интерфейс.

Ключевые цели задания, включая стабильную работу, корректную интеграцию с базой данных, а также обеспечение безопасности и чистоты кода, были полностью достигнуты.