Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ) Институт радиоэлектроники и информационных технологий – РтФ

Школа профессионального и академического образования

Учебно-научный центр «Информационная безопасность»

Оценка \_\_\_\_\_\_\_\_\_\_

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА ПО КУРСОВОЙ РАБОТЕ**

**«TELEGRAM БОТ ОБУЧАЮЩИЙ ПРОГРАММИРОВАНИЮ»**

**ПО ДИСЦИПЛИНЕ: «ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ»**

|  |  |
| --- | --- |
| Преподаватель | Мирвода С.Г. |
| Студенты | Данилов А. И., Сайфутдинов Д.Р., Пырин И.А. |
| Специальность | Информационная безопасность телекоммуникационных систем |
| Группа | РИ-311050 |

Екатеринбург 2023

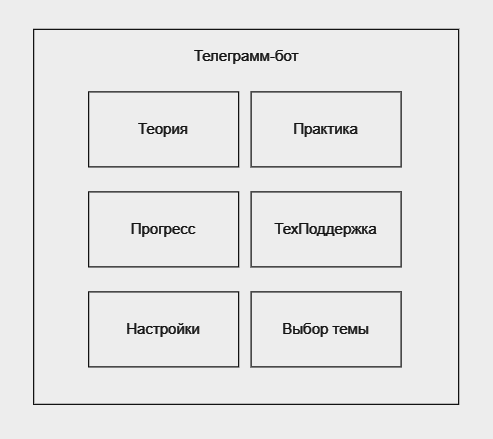
**Введение:**

Цель данной пояснительной записки – представить общее видение продукта. Проект ориентирован на создание инструмента для обучения программированию в удобном формате.

**Видение продукта:**

Цель проекта: Данный проект направлен на создание альтернативного подхода к обучению программированию через Telegram-платформу. Практичность данного метода позволит сделать обучение более интересным и доступным. Широкая распространенность Telegram позволит охватить широкий круг пользователей.

**Функциональная диаграмма продукта:**



# БЛОК ПРОЕКТИРОВАНИЯ

**Требования к ПО:**

Программа разработана на языке С# .NET 8.0.

Программа работает на ОС: Windows 10 Version 1607+ и выше, Alpine Linux 3.17+, Debian 11+, Fedora 37+, openSUSE 15+, Oracle Linux 8+, Red Hat Enterprise Linux 8+, SUSE Enterprise Linux 12 SP5+, Ubuntu 20.04+, MacOS 10.15+.

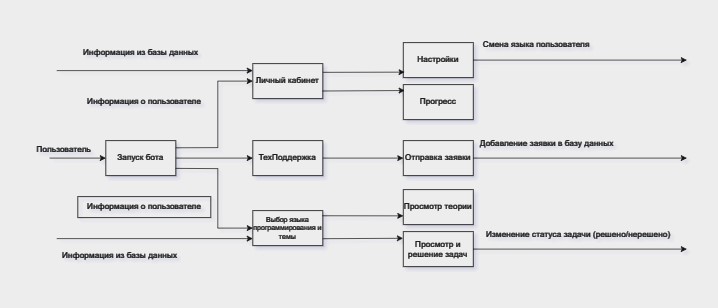
Интерфейс: Telegram 4.11.6+

СУБД: SQLite Version 3

**Архитектурная диаграмма:**

Изображение выглядит как зарисовка, диаграмма, рисунок, Шрифт

Автоматически созданное описание

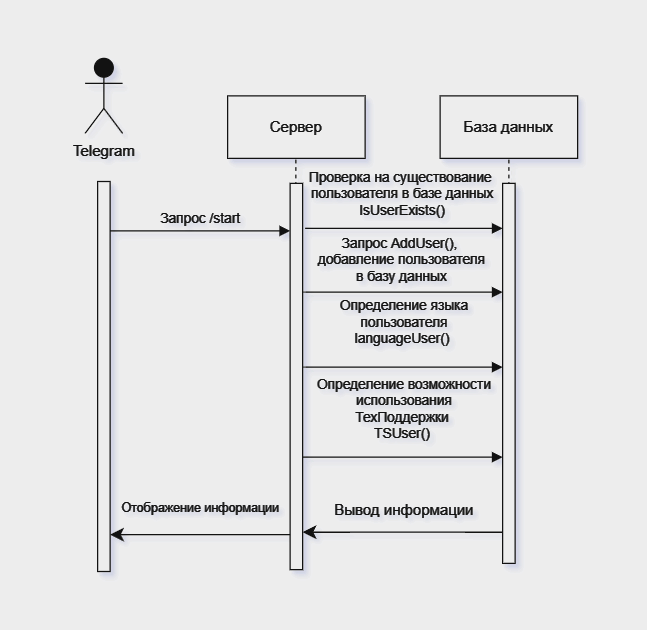
**Функциональная диаграмма UML:** 

**Концептуальная модель БД:**

**Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание**

**Диаграмма последовательности:**



**Реализация пользовательского интерфейса:**

Пользовательский интерфейс осуществляется посредством кнопок в Telegram чате:

* Кнопка «Начать»: Выбор языка программирования и темы для начала обучения.
* Кнопка «Личный кабинет»: Просмотр прогресса, управление настройками.
* Кнопка «ТехПоддержка»: Возможность оставить сообщение для разработчиков с указанием ошибки или некорректной информации.

**БЛОК РАЗРАБОТКИ**

В качестве языка программирования был выбран C#, так как он имеет следующие преимущества:

* **Производительность:** C# является компилируемым языком, что обеспечивает хорошую производительность. Это важно, особенно если ваш бот обрабатывает большой объем данных или выполняет сложные вычисления.
* **Многозадачность и асинхронность:** C# поддерживает асинхронное программирование, что полезно при обработке нескольких запросов одновременно. Это важно для ботов, работающих с множеством пользователей.
* **Кроссплатформенность:** С использованием .NET Core или .NET 5 и выше, телеграмм-бот на C# может быть кроссплатформенным, что позволяет запускать его на различных операционных системах.

В ходе работы была разработана программа (см. Приложение A), а также оформлены следующие диаграммы:

1. Диаграмма физической модели СУБД.

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

1. Диаграмма пакетов.

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Приложение А

Код программы

﻿Program.cs

using Telegram.Bot;

using Telegram.Bot.Types.Enums;

using Telegram.Bot.Exceptions;

using Telegram.Bot.Polling;

using Telegram.Bot.Types;

using Telegram.Bot.Types.ReplyMarkups;

using tgbot;

using static System.Net.Mime.MediaTypeNames;

using static System.Data.Entity.Infrastructure.Design.Executor;

using System.CodeDom.Compiler;

using static System.Runtime.InteropServices.JavaScript.JSType;

string TELEGRAM\_TOKEN = "…";

var botClient = new TelegramBotClient(TELEGRAM\_TOKEN);

using var cts = new CancellationTokenSource();

var receiverOptions = new ReceiverOptions

{

AllowedUpdates = { }

};

botClient.StartReceiving(HandleUpdateAsync, HandleErrorAsync, receiverOptions, cancellationToken: cts.Token);

Console.ReadLine();

async Task HandleUpdateAsync(ITelegramBotClient botClient, Telegram.Bot.Types.Update update, CancellationToken token)

{

if (update.Message != null && update.Message.Text != null)

{

var chatID = update.Message.Chat.Id;

var messageText = update.Message.Text;

var id\_user = update.Message.From.Id;

var name = update.Message.From.FirstName;

var messageId = update.Message.MessageId;

var userts = SQL.TSUser(id\_user);

string nickname;

if (update.Message.From.Username == null)

{

nickname = update.Message.From.FirstName;

}

else

{

nickname = update.Message.From.Username;

}

var language = SQL.languageUser(id\_user);

if (language == 1)

{

if (messageText.StartsWith("@ProgEd\_bot") && userts == 1)

{

string text = SQL.TechSup(messageText,id\_user,language);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("Главное меню", "ButtonBackMenu")

}

});

SQL.ChangeTS(id\_user, 0);

await botClient.SendTextMessageAsync(chatID, text, replyMarkup: inlineKeyboard);

}

if (messageText == "/start")

{

SQL.AddUser(id\_user, nickname);

SQL.ChangeTS(id\_user, 0);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("⭐ Начать", "Start")

},

new[]

{

InlineKeyboardButton.WithCallbackData("🗃️ Личный кабинет", "LK")

},

new[]

{

InlineKeyboardButton.WithCallbackData("💻 ТехПоддержка", "TS")

}

});

await botClient.SendTextMessageAsync(chatID,"Главное меню", replyMarkup: inlineKeyboard);

}

}

if (language == 2)

{

if (messageText.StartsWith("@ProgEd\_bot") && userts == 1)

{

string text = SQL.TechSup(messageText, id\_user, language);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("Главное меню", "ButtonBackMenu")

}

});

SQL.ChangeTS(id\_user, 0);

await botClient.SendTextMessageAsync(chatID, text, replyMarkup: inlineKeyboard);

}

if (messageText == "/start")

{

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("⭐ Start", "Start")

},

new[]

{

InlineKeyboardButton.WithCallbackData("🗃️ Account", "LK")

},

new[]

{

InlineKeyboardButton.WithCallbackData("💻 Technical support", "TS")

}

});

await botClient.SendTextMessageAsync(chatID, "Main menu", replyMarkup: inlineKeyboard);

}

}

}

if (update.Type == UpdateType.CallbackQuery)

{

await HandleCallbackQuery(botClient, update.CallbackQuery);

return;

}

}

async Task HandleCallbackQuery(ITelegramBotClient botClient, CallbackQuery callbackQuery)

{

var chatID = callbackQuery.Message.Chat.Id;

var name = callbackQuery.Message.Chat.FirstName;

string nickname;

if (callbackQuery.Message.Chat.Username == null)

{

nickname = callbackQuery.Message.Chat.FirstName;

}

else

{

nickname = callbackQuery.Message.Chat.Username;

}

var messageID = callbackQuery.Message.MessageId;

var data = callbackQuery.Data;

var language = SQL.languageUser(chatID);

if (data == "ButtonLangEng")

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

SQL.ChangeLanguage(chatID.ToString(), 2);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("Хорошо", "Settings")

}

});

await botClient.SendTextMessageAsync(chatID,"Язык изменён", replyMarkup: inlineKeyboard);

}

if (data == "ButtonLangRus")

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

SQL.ChangeLanguage(chatID.ToString(), 1);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("Ok", "Settings")

}

});

await botClient.SendTextMessageAsync(chatID, "The language has been changed", replyMarkup: inlineKeyboard);

}

if (language == 1)

{

if (data == "LK")

{

DeleteMessage(botClient, chatID, messageID);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("📈 Прогрес", "Progress")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⚙️ Настройки", "Settings")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Назад", "ButtonBackMenu")

}

});

await botClient.SendTextMessageAsync(chatID, $"Личный кабинет\nПользователь: {name}", replyMarkup: inlineKeyboard);

}

if (data == "Progress")

{

DeleteMessage(botClient, chatID, messageID);

List<(string,int, int)> lst = SQL.StatusProcent(SQL.GetProgressStatus(chatID, language));

int count = lst.Count;

string str = $"Прогрес пользователя {name}:\n\n";

for (int i = 0; i < count; i++)

{

if (lst[i].Item2 == 1)

{

string addStr = "(Python) " + lst[i].Item1 + ": " + lst[i].Item3.ToString() + "%\n";

str += addStr;

}

if (lst[i].Item2 == 2)

{

string addStr = "(C++) " + lst[i].Item1 + ": " + lst[i].Item3.ToString() + "%\n";

str += addStr;

}

}

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Назад", "LK")

}

});

await botClient.SendTextMessageAsync(chatID,str , replyMarkup: inlineKeyboard);

}

if (data == "TS")

{

DeleteMessage(botClient, chatID, messageID);

SQL.ChangeTS(chatID, 0);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("✉️ Отправить сообщение", "SendMes")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Назад", "ButtonBackMenu")

}

});

await botClient.SendTextMessageAsync(chatID, "ТехПоддержка", replyMarkup: inlineKeyboard);

}

if (data == "SendMes")

{

DeleteMessage(botClient, chatID, messageID);

SQL.ChangeTS(chatID, 1);

await botClient.SendTextMessageAsync(chatID, "Чтобы отправить сообщение в ТехПоддержку, отправьте сообщение, которое начинается с @ProgEd\_bot.\nЧтобы вернуться назад, напишите /start.");

}

if (data == "ButtonBackMenu")

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("⭐ Начать", "Start")

},

new[]

{

InlineKeyboardButton.WithCallbackData("🗃️ Личный кабинет", "LK")

},

new[]

{

InlineKeyboardButton.WithCallbackData("💻 ТехПоддержка", "TS")

}

});

await botClient.SendTextMessageAsync(chatID, "Главное меню", replyMarkup: inlineKeyboard);

}

if (data == "Start")

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("Python", "Menu1"),

InlineKeyboardButton.WithCallbackData("C++", "Menu2")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Назад", "ButtonBackMenu")

}

});

await botClient.SendTextMessageAsync(chatID, "Выбор языка программирования", replyMarkup: inlineKeyboard);

}

if (data == "Settings")

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("🔄 Сменить язык", "ButtonLangEng")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Назад", "LK")

}

});

await botClient.SendTextMessageAsync(chatID, "Выбран язык: Русский", replyMarkup: inlineKeyboard);

}

if (data == "Menu1")

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("📖 Теория", "Theory1"),

InlineKeyboardButton.WithCallbackData("📝 Практика", "Practic1")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Назад", "Start")

}

});

await botClient.SendTextMessageAsync(chatID, "Выбран язык программирования: Python", replyMarkup: inlineKeyboard);

}

if (data == "Menu2")

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("📖 Теория", "Theory2"),

InlineKeyboardButton.WithCallbackData("📝 Практика", "Practic2")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Назад", "Start")

}

});

await botClient.SendTextMessageAsync(chatID, "Выбран язык программирования: С++", replyMarkup: inlineKeyboard);

}

if (data.StartsWith("Practic"))

{

DeleteMessage(botClient, chatID, messageID);

int id\_proglanguage = int.Parse(data.Replace("Practic", ""));

// Получение данных из базы данных

List<(string, string, int)> courses = SQL.GetCourses(language, id\_proglanguage);

// Создание списка списков кнопок на основе полученных данных из базы

var keyboardButtons = courses.Select(course =>

new List<InlineKeyboardButton> { InlineKeyboardButton.WithCallbackData(course.Item1, $"prac{course.Item3}/{id\_proglanguage}") }

).ToList();

// Создание дополнительной кнопки в новой строке

var additionalButton = new List<InlineKeyboardButton> { InlineKeyboardButton.WithCallbackData("⬅️ Назад", $"Menu{id\_proglanguage}") };

keyboardButtons.Add(additionalButton);

// Создание клавиатуры с кнопками

var inlineKeyboard = new InlineKeyboardMarkup(keyboardButtons);

// Отправка сообщения с клавиатурой кнопок

await botClient.SendTextMessageAsync(

chatId: chatID,

text: "Выберите тему задач",

replyMarkup: inlineKeyboard

);

}

if (data.StartsWith("prac"))

{

DeleteMessage(botClient, chatID, messageID); // Перед этим определите эту функцию для удаления сообщений

string[] parts = data.Split('/');

int id\_proglanguage;

int courseId;

if (int.TryParse(parts[1], out id\_proglanguage) && int.TryParse(parts[0].Replace("prac", ""), out courseId))

{

List<(string, string, int)> exercises = SQL.GetExercises(courseId);

int count = exercises.Count;

var keyboardButtons = new List<List<InlineKeyboardButton>>();

// Создаем кнопки для каждого элемента в списке

for (int i = 0; i < count; i++)

{

string status = SQL.CheckStatus(chatID, exercises[i].Item3);

var button = InlineKeyboardButton.WithCallbackData($"Задание №{i + 1}" + $" ({status})", $"Task{exercises[i].Item3}/{courseId}/{id\_proglanguage}");

keyboardButtons.Add(new List<InlineKeyboardButton> { button });

}

var backButton = new List<InlineKeyboardButton>

{

InlineKeyboardButton.WithCallbackData("⬅️ Назад", $"Practic{id\_proglanguage}")

};

keyboardButtons.Add(backButton);

// Создаем клавиатуру из сгенерированных кнопок

var inlineKeyboard = new InlineKeyboardMarkup(keyboardButtons);

await botClient.SendTextMessageAsync(

chatId: chatID,

text: "Список заданий",

replyMarkup: inlineKeyboard

);

}

}

if (data.StartsWith("Task"))

{

DeleteMessage(botClient, chatID, messageID);

string[] parts = data.Split('/');

int id\_proglanguage;

int courseId;

int id\_task;

if (int.TryParse(parts[1], out courseId) && int.TryParse(parts[2], out id\_proglanguage) && int.TryParse(parts[0].Replace("Task", ""), out id\_task))

{

List<(string, string, int)> task = SQL.ShowTask(id\_task);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("✅ Решено", $"Ready{id\_task}"),

InlineKeyboardButton.WithCallbackData("❓ Ответ", $"Ans{id\_task}/{id\_proglanguage}/{courseId}")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Назад", $"prac{courseId}/{id\_proglanguage}")

}

});

await botClient.SendTextMessageAsync(chatID, task[0].Item1, replyMarkup: inlineKeyboard);

}

}

if (data.StartsWith("Ready"))

{

int id\_exercise = int.Parse(data.Replace("Ready", ""));

SQL.ChangeStatus(chatID, id\_exercise);

}

if (data.StartsWith("Ans"))

{

DeleteMessage(botClient, chatID, messageID);

string[] parts = data.Split('/');

int id\_proglanguage;

int courseId;

int id\_task;

if (int.TryParse(parts[2], out courseId) && int.TryParse(parts[1], out id\_proglanguage) && int.TryParse(parts[0].Replace("Ans", ""), out id\_task))

{

List<(string, string, int)> task = SQL.ShowTask(id\_task);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Назад", $"Task{id\_task}/{courseId}/{id\_proglanguage}")

}

});

await botClient.SendTextMessageAsync(chatID, task[0].Item2, replyMarkup: inlineKeyboard);

}

}

if (data.StartsWith("Theory"))

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

int id\_proglanguage = int.Parse(data.Replace("Theory", ""));

// Получение данных из базы данных

List<(string, string, int)> courses = SQL.GetCourses(language, id\_proglanguage);

// Создание списка списков кнопок на основе полученных данных из базы

var keyboardButtons = courses.Select(course =>

new List<InlineKeyboardButton> { InlineKeyboardButton.WithCallbackData(course.Item1 + $" ({course.Item2})", $"Command{course.Item3}/{id\_proglanguage}") }

).ToList();

// Создание дополнительной кнопки в новой строке

var additionalButton = new List<InlineKeyboardButton> { InlineKeyboardButton.WithCallbackData("⬅️ Назад", $"Menu{id\_proglanguage}") };

keyboardButtons.Add(additionalButton);

// Создание клавиатуры с кнопками

var inlineKeyboard = new InlineKeyboardMarkup(keyboardButtons);

// Отправка сообщения с клавиатурой кнопок

await botClient.SendTextMessageAsync(

chatId: chatID,

text: "Выберите тему для изучения",

replyMarkup: inlineKeyboard

);

}

if (data.StartsWith("Command"))

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

string[] parts = data.Split('/');

int id\_proglanguage;

int courseId;

if (int.TryParse(parts[1], out id\_proglanguage) && int.TryParse(parts[0].Replace("Command", ""), out courseId))

{

string theory = SQL.ShowTheory(courseId);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Назад", $"Theory{id\_proglanguage}")

}

});

await botClient.SendTextMessageAsync(chatID, theory, replyMarkup: inlineKeyboard);

}

}

}

if (language == 2)

{

if (data == "LK")

{

DeleteMessage(botClient, chatID, messageID);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("📈 Progress", "Progress")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⚙️ Settings", "Settings")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Back", "ButtonBackMenu")

}

});

await botClient.SendTextMessageAsync(chatID, $"Account\nUser: {name}", replyMarkup: inlineKeyboard);

}

if (data == "TS")

{

DeleteMessage(botClient, chatID, messageID);

SQL.ChangeTS(chatID, 0);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("✉️ Send message", "SendMes")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Back", "ButtonBackMenu")

}

});

await botClient.SendTextMessageAsync(chatID, "Technical support", replyMarkup: inlineKeyboard);

}

if (data == "SendMes")

{

DeleteMessage(botClient, chatID, messageID);

SQL.ChangeTS(chatID, 1);

await botClient.SendTextMessageAsync(chatID, "To send a message to Tech Support, send a message that starts with @ProgEd\_bot.\nTo go back, write /start.");

}

if (data == "Progress")

{

DeleteMessage(botClient, chatID, messageID);

List<(string, int, int)> lst = SQL.StatusProcent(SQL.GetProgressStatus(chatID, language));

int count = lst.Count;

string str = $"{name}'s progress:\n\n";

for (int i = 0; i < count; i++)

{

if (lst[i].Item2 == 1)

{

string addStr = "(Python) " + lst[i].Item1 + ": " + lst[i].Item3.ToString() + "%\n";

str += addStr;

}

if (lst[i].Item2 == 2)

{

string addStr = "(C++) " + lst[i].Item1 + ": " + lst[i].Item3.ToString() + "%\n";

str += addStr;

}

}

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Back", "LK")

}

});

await botClient.SendTextMessageAsync(chatID, str, replyMarkup: inlineKeyboard);

}

if (data == "ButtonBackMenu")

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("⭐ Start", "Start")

},

new[]

{

InlineKeyboardButton.WithCallbackData("🗃️ Account", "LK")

},

new[]

{

InlineKeyboardButton.WithCallbackData("💻 Technical support", "TS")

}

}) ;

await botClient.SendTextMessageAsync(chatID, "Main menu", replyMarkup: inlineKeyboard);

}

if (data == "Start")

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("Python", "Menu1"),

InlineKeyboardButton.WithCallbackData("C++", "Menu2")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Back", "ButtonBackMenu")

}

});

await botClient.SendTextMessageAsync(chatID, "Choosing a programming language", replyMarkup: inlineKeyboard);

}

if (data == "Settings")

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("🔄 Change the language", "ButtonLangRus")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Back", "LK")

}

});

await botClient.SendTextMessageAsync(chatID, "Selected language: English", replyMarkup: inlineKeyboard);

}

if (data == "Menu1")

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("📖 Theory", "Theory1"),

InlineKeyboardButton.WithCallbackData("📝 Exercises", "Practic1")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Back", "Start")

}

});

await botClient.SendTextMessageAsync(chatID, "Programming language selected: Python", replyMarkup: inlineKeyboard);

}

if (data == "Menu2")

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("📖 Theory", "Theory2"),

InlineKeyboardButton.WithCallbackData("📝 Exercises", "Practic2")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Back", "Start")

}

});

await botClient.SendTextMessageAsync(chatID, "Programming language selected: C++", replyMarkup: inlineKeyboard);

}

if (data.StartsWith("Practic"))

{

DeleteMessage(botClient, chatID, messageID);

int id\_proglanguage = int.Parse(data.Replace("Practic", ""));

// Получение данных из базы данных

List<(string, string, int)> courses = SQL.GetCourses(language, id\_proglanguage);

// Создание списка списков кнопок на основе полученных данных из базы

var keyboardButtons = courses.Select(course =>

new List<InlineKeyboardButton> { InlineKeyboardButton.WithCallbackData(course.Item1, $"prac{course.Item3}/{id\_proglanguage}") }

).ToList();

// Создание дополнительной кнопки в новой строке

var additionalButton = new List<InlineKeyboardButton> { InlineKeyboardButton.WithCallbackData("⬅️ Back", $"Menu{id\_proglanguage}") };

keyboardButtons.Add(additionalButton);

// Создание клавиатуры с кнопками

var inlineKeyboard = new InlineKeyboardMarkup(keyboardButtons);

// Отправка сообщения с клавиатурой кнопок

await botClient.SendTextMessageAsync(

chatId: chatID,

text: "Select a exercise topic:",

replyMarkup: inlineKeyboard

);

}

if (data.StartsWith("prac"))

{

DeleteMessage(botClient, chatID, messageID); // Перед этим определите эту функцию для удаления сообщений

string[] parts = data.Split('/');

int id\_proglanguage;

int courseId;

if (int.TryParse(parts[1], out id\_proglanguage) && int.TryParse(parts[0].Replace("prac", ""), out courseId))

{

List<(string, string, int)> exercises = SQL.GetExercises(courseId);

int count = exercises.Count;

var keyboardButtons = new List<List<InlineKeyboardButton>>();

// Создаем кнопки для каждого элемента в списке

for (int i = 0; i < count; i++)

{

string status = SQL.CheckStatus(chatID, exercises[i].Item3);

var button = InlineKeyboardButton.WithCallbackData($"Exercise №{i + 1}" + $" ({status})", $"Task{exercises[i].Item3}/{courseId}/{id\_proglanguage}");

keyboardButtons.Add(new List<InlineKeyboardButton> { button });

}

var backButton = new List<InlineKeyboardButton>

{

InlineKeyboardButton.WithCallbackData("⬅️ Back", $"Practic{id\_proglanguage}")

};

keyboardButtons.Add(backButton);

// Создаем клавиатуру из сгенерированных кнопок

var inlineKeyboard = new InlineKeyboardMarkup(keyboardButtons);

await botClient.SendTextMessageAsync(

chatId: chatID,

text: "List of exercises",

replyMarkup: inlineKeyboard

);

}

}

if (data.StartsWith("Task"))

{

DeleteMessage(botClient, chatID, messageID);

string[] parts = data.Split('/');

int id\_proglanguage;

int courseId;

int id\_task;

if (int.TryParse(parts[1], out courseId) && int.TryParse(parts[2], out id\_proglanguage) && int.TryParse(parts[0].Replace("Task", ""), out id\_task))

{

List<(string, string, int)> task = SQL.ShowTask(id\_task);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("✅ Ready", $"Ready{id\_task}"),

InlineKeyboardButton.WithCallbackData("❓ Answer", $"Ans{id\_task}/{id\_proglanguage}/{courseId}")

},

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Back", $"prac{courseId}/{id\_proglanguage}")

}

});

await botClient.SendTextMessageAsync(chatID, task[0].Item1, replyMarkup: inlineKeyboard);

}

}

if (data.StartsWith("Ready"))

{

int id\_exercise = int.Parse(data.Replace("Ready", ""));

SQL.ChangeStatus(chatID, id\_exercise);

}

if (data.StartsWith("Ans"))

{

DeleteMessage(botClient, chatID, messageID);

string[] parts = data.Split('/');

int id\_proglanguage;

int courseId;

int id\_task;

if (int.TryParse(parts[2], out courseId) && int.TryParse(parts[1], out id\_proglanguage) && int.TryParse(parts[0].Replace("Ans", ""), out id\_task))

{

List<(string, string, int)> task = SQL.ShowTask(id\_task);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Back", $"Task{id\_task}/{courseId}/{id\_proglanguage}")

}

});

await botClient.SendTextMessageAsync(chatID, task[0].Item2, replyMarkup: inlineKeyboard);

}

}

if (data.StartsWith("Theory"))

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

int id\_proglanguage = int.Parse(data.Replace("Theory", ""));

// Получение данных из базы данных

List<(string, string, int)> courses = SQL.GetCourses(language, id\_proglanguage);

// Создание списка списков кнопок на основе полученных данных из базы

var keyboardButtons = courses.Select(course =>

new List<InlineKeyboardButton> { InlineKeyboardButton.WithCallbackData(course.Item1 + $" ({course.Item2})", $"Command{course.Item3}/{id\_proglanguage}") }

).ToList();

// Создание дополнительной кнопки в новой строке

var additionalButton = new List<InlineKeyboardButton> { InlineKeyboardButton.WithCallbackData("⬅️ Back", $"Menu{id\_proglanguage}") };

keyboardButtons.Add(additionalButton);

// Создание клавиатуры с кнопками

var inlineKeyboard = new InlineKeyboardMarkup(keyboardButtons);

// Отправка сообщения с клавиатурой кнопок

await botClient.SendTextMessageAsync(

chatId: chatID,

text: "Choose a topic to study:",

replyMarkup: inlineKeyboard

);

}

if (data.StartsWith("Command"))

{

//await botClient.DeleteMessageAsync(chatID, messageID);

DeleteMessage(botClient, chatID, messageID);

string[] parts = data.Split('/');

int id\_proglanguage;

int courseId;

if (int.TryParse(parts[1], out id\_proglanguage) && int.TryParse(parts[0].Replace("Command", ""), out courseId))

{

string theory = SQL.ShowTheory(courseId);

var inlineKeyboard = new InlineKeyboardMarkup(new[]

{

new[]

{

InlineKeyboardButton.WithCallbackData("⬅️ Back", $"Theory{id\_proglanguage}")

}

});

await botClient.SendTextMessageAsync(chatID, theory, replyMarkup: inlineKeyboard);

}

}

}

}

async Task DeleteMessage(ITelegramBotClient botClient, long chatId, int messageId)

{

try

{

await botClient.DeleteMessageAsync(chatId, messageId);

}

catch (ApiRequestException ex)

{

// Обработка ошибки удаления сообщения

Console.WriteLine($"Ошибка удаления сообщения: {ex.Message}");

}

}

Task HandleErrorAsync(ITelegramBotClient botClient, Exception exception, CancellationToken cancellationToken)

{

var errorMessage = exception switch

{

ApiRequestException apiRequestException => $"Telegram API Error:\n{apiRequestException.ErrorCode}\n{apiRequestException.Message}",

\_ => exception.ToString()

};

Console.WriteLine(errorMessage);

return Task.CompletedTask;

}

SQL.cs

﻿using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Data.SQLite;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Telegram.Bot.Types;

using Telegram.Bot;

using System.IO.Pipes;

namespace tgbot

{

public class SQL

{

public static string CONNECTION\_STRING = @"Data Source =" + Environment.CurrentDirectory + @"\proged.sqlite; Version=3;";

public static void ChangeLanguage(string id\_user, int id\_language)

{

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

var command = connection.CreateCommand();

command.CommandText = $"UPDATE Users SET id\_language = {id\_language} WHERE id\_user like {id\_user}";

command.ExecuteNonQuery();

connection.Close();

}

}

public static int languageUser(long id\_user)

{

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

var command = new SQLiteCommand();

command.Connection = connection;

command.CommandText = $"SELECT id\_language FROM Users WHERE id\_user = {id\_user}";

using (SQLiteDataReader reader = command.ExecuteReader())

{

if (reader.Read()) // Проверяем наличие строк для чтения

{

int id = Convert.ToInt32(reader["id\_language"]);

return id;

}

}

return 1;

}

}

public static int TSUser(long id\_user)

{

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

var command = new SQLiteCommand();

command.Connection = connection;

command.CommandText = $"SELECT use\_techsup FROM Users WHERE id\_user = {id\_user}";

using (SQLiteDataReader reader = command.ExecuteReader())

{

if (reader.Read()) // Проверяем наличие строк для чтения

{

int id = Convert.ToInt32(reader["use\_techsup"]);

return id;

}

}

return 1;

}

}

public static void ChangeTS(long id\_user, int use\_techsup)

{

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

var command = connection.CreateCommand();

command.CommandText = $"UPDATE Users SET use\_techsup = {use\_techsup} WHERE id\_user like {id\_user}";

command.ExecuteNonQuery();

connection.Close();

}

}

public static string TechSup(string str, long user\_id, int language)

{

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

var command = new SQLiteCommand();

command.Connection = connection;

command.CommandText = $"insert into TechnicalSupport(id\_user,appel) values ({user\_id}, @str)";

command.Parameters.AddWithValue("@str", str);

command.ExecuteNonQuery();

connection.Close();

if (language == 1)

{

return "Сообщение успешно отправлено";

}

if (language == 2)

{

return "The message has been sent successfully";

}

return "1";

}

}

public static bool IsUserExists(long id\_user)

{

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

var command = new SQLiteCommand();

command.Connection = connection;

command.CommandText = $"Select \* from Users where id\_user = {id\_user}";

return command.ExecuteScalar() == null;

}

}

public static int GetCountExercise()

{

List<int> count = new List<int>();

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

var command = new SQLiteCommand();

command.Connection = connection;

command.CommandText = $"SELECT id\_exercise from Exercise";

using (SQLiteDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

int id = Convert.ToInt32(reader["id\_exercise"]);

count.Add(id);

}

}

connection.Close();

}

return count.Count;

}

public static void AddUser(long id\_user, string nickname)

{

if (IsUserExists(id\_user))

{

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

var command = new SQLiteCommand();

command.Connection = connection;

command.CommandText = $"insert into Users(id\_user, nickname, id\_language, use\_techsup) values ({id\_user},'{nickname}',1,0)";

command.ExecuteNonQuery();

}

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

int count = GetCountExercise();

for (int i = 0; i < count; i++)

{

using (var command = new SQLiteCommand())

{

command.Connection = connection;

command.CommandText = $"insert into Progress\_status (id\_user, id\_exercise, status) values ({id\_user},{i + 1},'NotCompleted')";

command.ExecuteNonQuery();

}

}

connection.Close();

}

}

}

public static string ShowTheory(int id\_theory)

{

string result = ""; // Инициализируем переменную заранее

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

var command = new SQLiteCommand();

command.Connection = connection;

command.CommandText = $"SELECT theory FROM Theory WHERE id\_Theory = {id\_theory}";

using (SQLiteDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

result = reader["theory"].ToString();

}

}

connection.Close();

}

return result; // Возвращаем значение result после завершения работы с базой данных

}

public static List<(string, string, int)> ShowTask(int id\_exercise)

{

List<(string, string, int)> result = new List<(string, string, int)>(); // Инициализируем переменную заранее

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

var command = new SQLiteCommand();

command.Connection = connection;

command.CommandText = $"SELECT exercise, answer, id\_exercise FROM Exercise WHERE id\_exercise = {id\_exercise}";

using (SQLiteDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

string ex = reader["exercise"].ToString();

string ans = reader["answer"].ToString();

result.Add((ex, ans, id\_exercise));

}

}

connection.Close();

}

return result; // Возвращаем значение result после завершения работы с базой данных

}

public static List<(string, string, int)> GetCourses(int id\_language, int id\_proglanguage)

{

List<(string, string, int)> courses = new List<(string, string, int)>();

using (SQLiteConnection connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

string query = $"SELECT name, difficulty, id\_course FROM Courses where id\_language = {id\_language} and id\_proglanguage = {id\_proglanguage}"; // Выбираем оба столбца

using (SQLiteCommand command = new SQLiteCommand(query, connection))

{

using (SQLiteDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

string name = reader["name"].ToString();

string difficulty = reader["difficulty"].ToString();

int id = Convert.ToInt32(reader["id\_course"]);

courses.Add((name, difficulty, id));

}

}

}

connection.Close();

}

return courses;

}

public static List<(string, string, int)> GetExercises(int id\_course)

{

List<(string, string, int)> exercises = new List<(string, string, int)>();

using (SQLiteConnection connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

string query = $"SELECT exercise, answer, id\_exercise FROM Exercise where id\_course = {id\_course}"; // Выбираем оба столбца

using (SQLiteCommand command = new SQLiteCommand(query, connection))

{

using (SQLiteDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

string exercise = reader["exercise"].ToString();

string answer = reader["answer"].ToString();

int id = Convert.ToInt32(reader["id\_exercise"]);

exercises.Add((exercise, answer, id));

}

}

}

connection.Close();

}

return exercises;

}

public static void ChangeStatus(long id\_user, int id\_exercise)

{

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

List<(string, string, int)> task = ShowTask(id\_exercise);

List<int> lst = SearchExercise(task[0].Item2);

int count = lst.Count;

for (int i = 0; i < count; i++)

{

using (var command = new SQLiteCommand())

{

command.Connection = connection;

command.CommandText = $"Update Progress\_status set status = 'Completed' where id\_user = {id\_user} and id\_exercise = {lst[i]}";

command.ExecuteNonQuery();

}

}

}

}

public static List<int> SearchExercise(string Answer)

{

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

List<int> ids = new List<int>();

var command = new SQLiteCommand();

command.Connection = connection;

command.CommandText = $"select id\_exercise from Exercise where answer = @Answer";

command.Parameters.AddWithValue("@Answer", Answer);

using (SQLiteDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

int id = Convert.ToInt32(reader["id\_exercise"]);

ids.Add(id);

}

}

connection.Close();

return ids;

}

}

public static string CheckStatus(long id\_user, int id\_exercise)

{

string status = "";

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

var command = new SQLiteCommand();

command.Connection = connection;

command.CommandText = $"select status from Progress\_status where id\_user = {id\_user} and id\_exercise = {id\_exercise}";

using (SQLiteDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

status = reader["status"].ToString();

}

}

connection.Close();

}

if (status == "NotCompleted")

{

return "✗";

}

if (status == "Completed")

{

return "✓";

}

return "1";

}

public static List<(string, int, int)> StatusProcent(List<(int, int, List<int>, List<int>)> lst)

{

List<(string, int ,int)> finalyLst = new List<(string, int ,int)>();

int count = lst.Count;

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

for (int i = 0; i < count; i++)

{

List<int> ids\_exercise = new List<int>();

using (var command = new SQLiteCommand())

{

command.Connection = connection;

command.CommandText = $"select name from Courses where id\_course = {lst[i].Item1}";

using (SQLiteDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

string name = reader["name"].ToString();

float test = ((float)lst[i].Item4.Count / (float)lst[i].Item3.Count) \* 100;

int id = Convert.ToInt32(test);

finalyLst.Add((name, lst[i].Item2,id));

}

}

}

}

connection.Close();

return finalyLst;

}

}

public static List<(int,int, List<int>, List<int>)> GetProgressStatus(long id\_user, int id\_language)

{

List<(int, int, List<int>, List<int>)> lst = new List<(int,int, List<int>, List<int>)>();

List<(int,int)> ids\_courses = new List<(int,int)>();

List<(int, List<int>)> lst\_exercise = new List<(int, List<int>)>();

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

var command = new SQLiteCommand();

command.Connection = connection;

command.CommandText = $"select id\_course, id\_proglanguage from Courses where id\_language = {id\_language}";

using (SQLiteDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

int id = Convert.ToInt32(reader["id\_course"]);

int id\_pr = Convert.ToInt32(reader["id\_proglanguage"]);

ids\_courses.Add((id, id\_pr));

}

}

connection.Close();

}

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

int count = ids\_courses.Count();

for (int i = 0; i < count; i++)

{

List<int> ids\_exercise = new List<int>();

using (var command = new SQLiteCommand())

{

command.Connection = connection;

command.CommandText = $"select id\_exercise from Exercise where id\_course = {ids\_courses[i].Item1}";

using (SQLiteDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

int id = Convert.ToInt32(reader["id\_exercise"]);

ids\_exercise.Add(id);

}

}

}

List<int> Completed = GetCompleted(id\_user, ids\_exercise);

lst.Add((ids\_courses[i].Item1, ids\_courses[i].Item2,ids\_exercise,Completed));

}

connection.Close();

}

return lst;

}

public static List<int> GetCompleted(long id\_user, List<int> ids\_exercise)

{

using (var connection = new SQLiteConnection(CONNECTION\_STRING))

{

connection.Open();

List<int> lst = new List<int>();

int count = ids\_exercise.Count();

for (int i = 0; i < count; i++)

{

List<int> ids\_complexercise = new List<int>();

using (var command = new SQLiteCommand())

{

command.Connection = connection;

command.CommandText = $"select status from Progress\_status where id\_exercise = {ids\_exercise[i]} and id\_user = {id\_user}";

using (SQLiteDataReader reader = command.ExecuteReader())

{

while (reader.Read())

{

string status = reader["status"].ToString();

if (status == "Completed")

{

lst.Add(ids\_exercise[i]);

}

}

}

}

}

connection.Close();

return lst;

}

}

}

}