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 фундаментального образования

Кафедра «Учебно-Научный Центр информационной безопасности»

Оценка

Руководитель курсового

проектирования Мирвода С. Г.

Члены комиссии

Дата защиты

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по теме: бот по отслеживанию питания и питья

по дисциплине: Языки и методы программирования

Студент: Леханов Л.  
Морозов А.  
Смирнов М.

(Подпись)

Группа: РИ-311055

Екатеринбург

2023

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 фундаментального образования

Кафедра «учебно-научный центр информационной безопасности»

**Задание**

**на курсовую работу**

Студенты:

Специальность/направление подготовки: Информационно-аналитические системы безопасности

1. Тема курсовой работы: бот по отслеживанию питания и питья
2. Содержание работы, в том числе состав графических работ и расчетов: Создание пользовательского интерфейса на основе базы данных филармонии, описание архитектурных диаграмм.
3. План выполнения курсовой работы

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование элементов  проектной работы | Сроки | Примечания | Отметка о выполнении |
| Создание базы данных продуктов и блюд | 20.12.2023 |  | Выполнено |
| Оформление физической диаграммы | 22.12.2023 |  | Выполнено |
| Описание видения проекта, функциональная диаграмма | 25.12.2023 |  | Выполнено |
| Оформление архитектурных диаграмм, написание логики бота | 07.01.2024 |  | Выполнено |
| Написание требований к системному ПО (СУБД) и взаимодействие бота с БД | 07.01.2024 |  | Выполнено |

|  |  |  |  |
| --- | --- | --- | --- |
| Финальное тестирование бота | 10.01.2024 |  | Выполнено |

Руководитель /И.О. Фамилия/

Министерство образования и науки Российской Федерации ФГАОУ ВО «УрФУ имени первого Президента России Б.Н.Ельцина»

**РЕЦЕНЗИЯ**

на курсовую работу

Студента:

(фамилия имя отчество)

Тема курсовой работы:

Модуль/дисциплина:

Соответствие результатов выполнения работы целям и задачам курсового проектирования, результатам обучения по дисциплине/модулю

1. Оригинальность и самостоятельность выполнения работы
2. Полнота и глубина проработки разделов
3. Общая грамотность и качество оформления текстового документа и графических материалов
4. Вопросы и замечания
5. Общая оценка работы

Сведения о рецензенте:

Ф.И.О.

Должность Место работы Уч. звание Уч. степень

Подпись Дата

**Оглавление**

[**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 6**](#_Toc156012448)

[**ВВЕДЕНИЕ 7**](#_Toc156012449)

[**1. АРХИТЕКТУРА ПРОЕКТА 8**](#_Toc156012450)

[**1.1 Проектирование и видение 8**](#_Toc156012451)

[**2. РЕАЛИЗАЦИЯ ПРОЕКТА 12**](#_Toc156012452)

[**2.1 Субд 12**](#_Toc156012453)

[**2.2 Логика бота 13**](#_Toc156012454)

[**3. Алгоритм работы 16**](#_Toc156012455)

[**ЗАКЛЮЧЕНИЕ 20**](#_Toc156012456)

[**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 21**](#_Toc156012457)

[**ПРИЛОЖЕНИЕ А 22**](#_Toc156012458)

# ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

База данных (БД) – некая именованная совокупность данных, отражающая состояние объектов или явлений в некоторой выбранной предметной области и связи между ними. Она создаётся для хранения данных и доступа к ним.

Система управления базами данных (СУБД) – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Пользовательский интерфейс – совокупность средств и методов, при помощи которых пользователь взаимодействует с различными, чаще всего сложными, машинами, устройствами и аппаратурой.

UML – унифицированный графический язык моделирования для описания, визуализации, проектирования и документирования систем.

Язык программирования – это система обозначений, служащая для точного описания программ или алгоритмов для ЭВМ.

# ВВЕДЕНИЕ

Основной и главной целью этого проекта является создание удобного и понятного бота, который будет учитывать и хранить в себе данные о приемах пищи и воды.Также для удобного взаимодействия с информацией полученной от пользователя было решено реализовать возможность редактирования собственного меню и записи его в базу данных

**Название продукта:** " Food & water tracking bot "   
Описание: **Food & water tracking bot** - это телеграм-бот, который помогает пользователям легко управлять своим питанием и поддерживать здоровый образ жизни. Бот предлагает два основных режима работы: "мое меню" и "уведомления".  
**Мое Меню**:   
*Персонализированное меню*: Пользователи могут сохранять свои любимые блюда в "Моем Меню" с указанием калорий и состава.   
*Учет калорий*: Бот автоматически подсчитывает общую калорийность и пищевые компоненты на основе добавленных блюд.   
*Уведомления о приеме пищи*: Бот отправляет персонализированные уведомления, напоминая пользователю о времени приема пищи и предлагая оптимальное блюдо. Преимущества: Легкость в использовании и персонализация. Умное формирование меню с учетом предпочтений и питательных нужд. Помощь в достижении здоровых пищевых привычек. Регулярные напоминания о приеме пищи. **Food & water tracking bot** представляет собой помощника в поддержании здорового образа жизни через подход к питанию и планированию приемов пищи.

# АРХИТЕКТУРА ПРОЕКТА

Проект состоит из структуры взаимодействия СУБД и телеграм-бота в качестве пользовательского интерфейса, что указано в диаграммах.

## 1.1 Проектирование и видение

Для построения архитектуры проекта требуется создать несколько диаграмм, которые отображают видение продукта.

В архитектурной диаграмме (рис. 1) отображается взаимодействие пользователя с системой: для получения необходимой информации человек заходит в телеграм-бот и выдает ему определенный запрос, который далее обрабатывается и передается в базу данных PostgreSQL, откуда найденная информация выводится обратно на экран пользователю.

Изображение выглядит как текст, диаграмма, дизайн, снимок экрана

Автоматически созданное описание

Рисунок 1 – архитектурная диаграмма

Функциональная диаграмма UML изображена на рисунке 2.

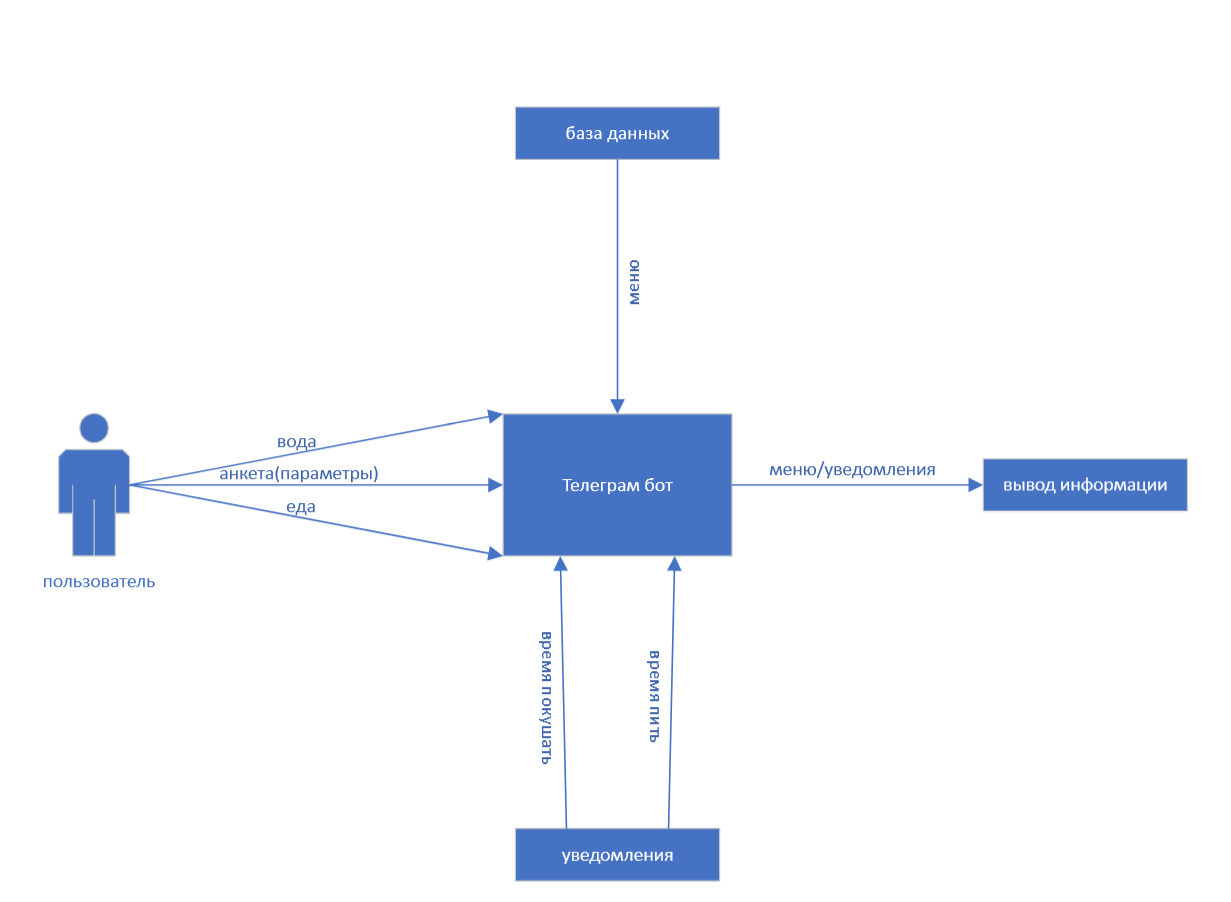


Рисунок 2 – функциональная диаграмма UML

Для обработки запроса вывода запрашиваемой информации представлена диаграмма последовательности на рисунке 3.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Рисунок 3 – диаграмма последовательности UML

В проекте задействованы БД, библиотека C#(API тг бота) и пользовательский интерфейс(бот). Диаграмма пакетов UML небольшая и представлена на рисунке 4.

Изображение выглядит как текст, снимок экрана, Шрифт, Параллельный

Автоматически созданное описание

Рисунок 4 – диаграмма пакетов UML

Диаграмма прецедентов показывает нам какие элементы(акторы) взаимодействуют с объектами(прецедентами), отражено на рисунке 6  
  
Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание

Рисунок 6 – диаграмма прецедентов

# РЕАЛИЗАЦИЯ ПРОЕКТА

## 2.1 Субд

Разработанная база данных состоит из 4 сущностей, атрибуты которых указаны в диаграммах модели. Данная база данных наиболее компактно и логично хранит в себе всю необходимую информацию о блюдах, воде, калориях.

Диаграмма физической модели изображена на рисунке 7.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 7 – диаграмма физической модели

Требования к СУБД:

* Надёжность (целостность БД не должна нарушаться при технических сбоях).
* Целостность данных (непротиворечивость данных).
* Безопасность (защита данных от несанкционированного доступа).
* Скорость доступа (обеспечение быстрого доступа к требуемой информации).
* Возможность актуализации (изменение, корректировка данных).
* Совместимость (база данных должна быть совместима с другими системами и стандартами).
* Управление данными (база данных должна обеспечивать эффективное управление данными).

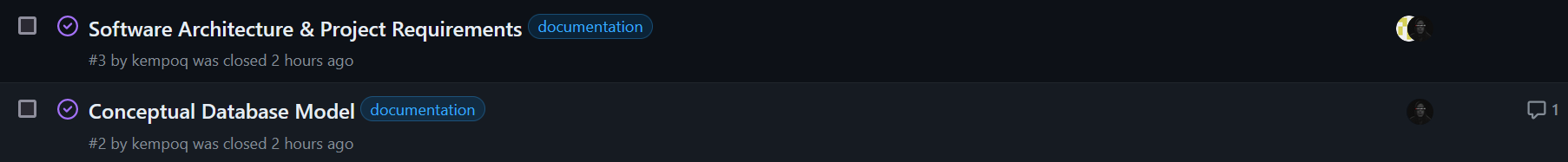
В результате выполнения данного этапа разработки были закрыты следующие задачи.  


Рисунок 8 – закрытые задачи по БД

## 2.2 Логика бота

BACKEND часть бота была наиболее объемной частью работы. Для наиболее результативной работы была составлена диаграмма классов. После визуализации была реализована основная логика бота. Ниже приведены 2 диаграммы которые визуализируют части программы.

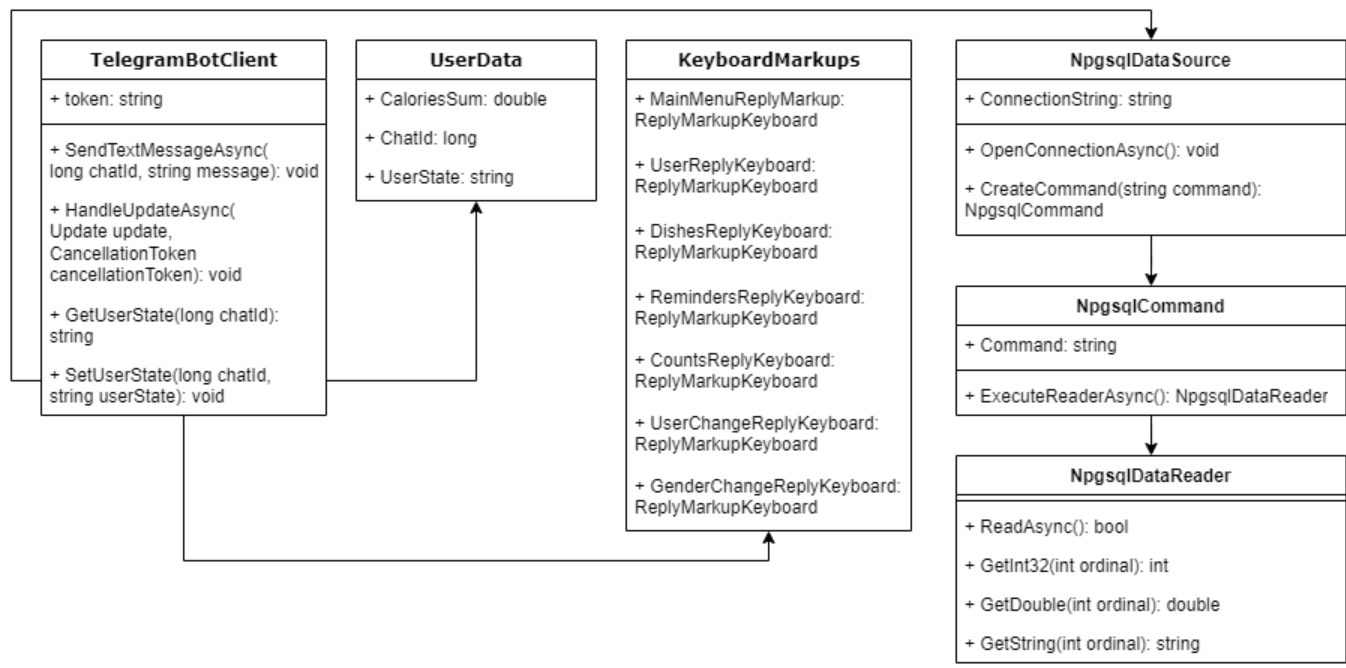


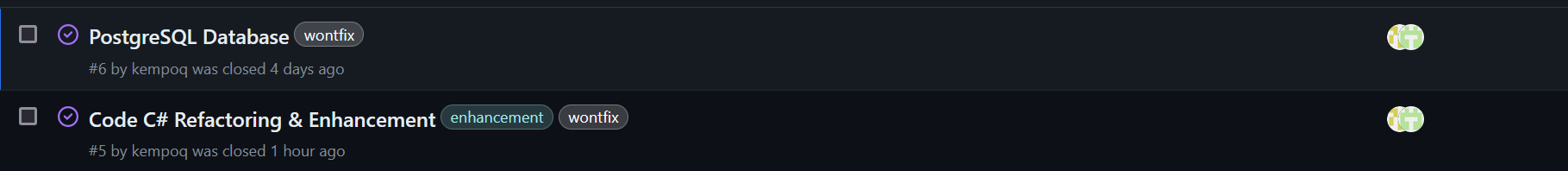
Рисунок 9 – диаграмма классов

Изображение выглядит как текст, снимок экрана, Шрифт, чек

Автоматически созданное описание

Рисунок 10 – диаграмма взаимодействия

В результате реализации и рефакторинга кода были закрыты следующие задачи:

  
Рисунок 11 – закрытые задачи по боту

# Алгоритм работы

Взаимодействие с телеграм ботом интуитивно понятное ввиду явно заметных кнопок интерфейса. Ниже пример последовательного взаимодействия с ботом

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 12 – заполнение параметров

Изображение выглядит как текст, снимок экрана, Шрифт, зеленый

Автоматически созданное описание

Рисунок 13 – заполнение роста

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 14 – анкета

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 15 – меню/еда



Рисунок 16 – запрос данных на основе анкеты о еде и воде

# ЗАКЛЮЧЕНИЕ

В результате выполнения работы был спроектирован и реализован бот для подсчета калорий и питья, а также создания меню. Таким образом, наша программа полностью соответствует первоначальной задумке. Формируемые режим питания и питья поможет избежать проблем и сформирует привычку правильного пищевого поведения.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Что такое Product Vision? [Электронный ресурс]. – URL: <https://www.tryexponent.com/blog/what-is-product-vision> (дата обращения 22.12.2023).
2. Про Product Vision и Product Vision Board [Электронный ресурс]. – URL: <https://upravlenie-proektami.ru/pro-product-vision-i-product-vision-board> (дата обращения 22.12.2023).
3. Что Такое Диаграмма Пакетов В UML? [Электронный ресурс]. – URL: <https://blog.visual-paradigm.com/ru/what-is-a-package-what-is-a-package-diagram-in-uml> (дата обращения 20.12.2023).

**ПРИЛОЖЕНИЕ А**

**Реализация телеграм-бота Program.cs**

﻿using System.Globalization;

using Npgsql;

using Telegram.Bot;

using Telegram.Bot.Exceptions;

using Telegram.Bot.Polling;

using Telegram.Bot.Types;

using Telegram.Bot.Types.Enums;

using Telegram.Bot.Types.ReplyMarkups;

using TG\_Bot;

var addDishString = "";

var eatenDishName = "";

CancellationTokenSource cts = new();

var botClient = new TelegramBotClient(Config.TgApiToken);

var pgCredentials = Config.PgCredentials;

await using var dataSource = NpgsqlDataSource.Create(

$"Host={pgCredentials["host"]};"

+ $"Username={pgCredentials["username"]};"

+ $"Password={pgCredentials["password"]};"

+ $"Database={pgCredentials["database"]};"

+ $"Port={pgCredentials["port"]}");

await using var connection = await dataSource.OpenConnectionAsync();

ReceiverOptions receiverOptions = new()

{

AllowedUpdates = Array.Empty<UpdateType>() // receive all update types except ChatMember related updates

};

botClient.StartReceiving(

HandleUpdateAsync,

HandlePollingErrorAsync,

receiverOptions,

cts.Token

);

var me = await botClient.GetMeAsync();

await Schedule();

async Task<string> GetUserState(long chatId)

{

var state = "";

await using var sqlCommand = dataSource.CreateCommand($"select user\_state from users where chat\_id = {chatId}");

await using var reader = await sqlCommand.ExecuteReaderAsync();

while (await reader.ReadAsync()) state = reader.GetString(0);

return state;

}

async Task SetUserState(long chatId, string state)

{

await using var sqlCommand =

dataSource.CreateCommand($"UPDATE users set user\_state = '{state}' where chat\_id = " + chatId);

await sqlCommand.ExecuteNonQueryAsync();

}

async Task HandleUpdateAsync(ITelegramBotClient telegramBotClient, Update update, CancellationToken cancellationToken)

{

try

{

var user = new UserData

{

ChatId = update.Type == UpdateType.Message

? update.Message.Chat.Id

: update.CallbackQuery.Message.Chat.Id

};

user.UserState = await GetUserState(user.ChatId);

async Task<InlineKeyboardMarkup> DishesMarkupMaker(long chatId)

{

var dishesList = new List<string>();

var command = $"select \"name\" from fooditem where chat\_id = {chatId}";

await using (var sqlCommand1 = dataSource.CreateCommand(command))

await using (var reader = await sqlCommand1.ExecuteReaderAsync())

{

while (await reader.ReadAsync())

for (var i = 0; i < reader.FieldCount; i++)

dishesList.Add(reader.GetString(i));

}

var buttons = dishesList.Select(

dish

=> new[] { InlineKeyboardButton.WithCallbackData(dish, dish) })

.ToList();

return new InlineKeyboardMarkup(buttons);

}

async Task<InlineKeyboardMarkup> MenuMarkupMaker(long chatId)

{

var menu = new List<string>();

var commands = new[]

{

"select \"name\" from fooditem f inner join fooditems\_per\_day fpd on "

+ $"f.chat\_id = fpd.chat\_id and f.fooditem\_id = fpd.breakfast where f.chat\_id = {chatId} limit 1;",

"select \"name\" from fooditem f inner join fooditems\_per\_day fpd on "

+ $"f.chat\_id = fpd.chat\_id and f.fooditem\_id = fpd.lunch where f.chat\_id = {chatId} limit 1;",

"select \"name\" from fooditem f inner join fooditems\_per\_day fpd on "

+ $"f.chat\_id = fpd.chat\_id and f.fooditem\_id = fpd.dinner where f.chat\_id = {chatId} limit 1;"

};

foreach (var command in commands)

{

await using var sqlCommand1 = dataSource.CreateCommand(command);

await using var reader = await sqlCommand1.ExecuteReaderAsync();

while (await reader.ReadAsync()) menu.Add(reader.GetString(0));

}

var buttons = menu.Select(

dish

=> new[] { InlineKeyboardButton.WithCallbackData(dish, dish) })

.ToList();

buttons.Add(new[] { InlineKeyboardButton.WithCallbackData("Всё меню", "whole\_list") });

return new InlineKeyboardMarkup(buttons);

}

switch (update.Type)

{

case UpdateType.Message:

{

var message = update.Message;

switch (message.Type)

{

case MessageType.Text:

{

user.ChatId = message.Chat.Id;

var chatIdExists = false;

if (message.Text != "Вернуться")

{

switch (user.UserState)

{

case "water\_per\_day\_change":

{

await using (var sqlCommand2 = dataSource.CreateCommand(

"UPDATE users set water\_per\_day = " + message.Text +

" where chat\_id = " + user.ChatId))

{

await sqlCommand2.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Планируемый объем воды изменен, поменяем что-нибудь ещё?",

replyMarkup: KeyboardMarkups.UserChangeReplyKeyboard);

user.UserState = "general\_data\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "calories\_per\_day\_change":

{

await using (var sqlCommand2 = dataSource.CreateCommand(

"UPDATE users set calories\_per\_day = " + message.Text +

" where chat\_id = " + user.ChatId))

{

await sqlCommand2.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Планируемый объем калорий изменен, поменяем что-нибудь ещё?",

replyMarkup: KeyboardMarkups.UserChangeReplyKeyboard);

user.UserState = "general\_data\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "height\_change":

{

await using (var sqlCommand2 = dataSource.CreateCommand(

"UPDATE users set height = " +

message.Text + " where chat\_id = " + user.ChatId))

{

await sqlCommand2.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Рост изменен, поменяем что-нибудь ещё?",

replyMarkup: KeyboardMarkups.UserChangeReplyKeyboard);

user.UserState = "general\_data\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "weight\_change":

{

await using (var sqlCommand2 = dataSource.CreateCommand(

"UPDATE users set weight = " +

message.Text + " where chat\_id = " + user.ChatId))

{

await sqlCommand2.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Вес изменен, поменяем что-нибудь ещё?",

replyMarkup: KeyboardMarkups.UserChangeReplyKeyboard);

user.UserState = "general\_data\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "age\_change":

{

await using (var sqlCommand2 = dataSource.CreateCommand(

"UPDATE users set age = " +

message.Text + " where chat\_id = " + user.ChatId))

{

await sqlCommand2.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Возраст изменен, поменяем что-нибудь ещё?",

replyMarkup: KeyboardMarkups.UserChangeReplyKeyboard);

user.UserState = "general\_data\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "gender\_change":

{

await using (var sqlCommand2 = dataSource.CreateCommand(

"UPDATE users set gender = '" +

message.Text + "' where chat\_id = " + user.ChatId))

{

await sqlCommand2.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Пол изменен, поменяем что-нибудь ещё?",

replyMarkup: KeyboardMarkups.UserChangeReplyKeyboard);

user.UserState = "general\_data\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "dish\_add\_weight":

{

addDishString += $"{message.Text}, {user.ChatId})";

await using var sqlCommand2 = dataSource.CreateCommand(addDishString);

await sqlCommand2.ExecuteNonQueryAsync();

await telegramBotClient.SendTextMessageAsync(message.Chat, "Блюдо добавлено",

replyMarkup: KeyboardMarkups.DishesReplyKeyboard);

user.UserState = "";

await SetUserState(user.ChatId, user.UserState);

addDishString = "";

return;

}

case "dish\_add\_calories":

{

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Введите средний вес порции в граммах");

addDishString += $"{message.Text},";

user.UserState = "dish\_add\_weight";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "dish\_add\_type":

{

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Введите калорийность блюда на 100 грамм сухого продукта");

user.UserState = "dish\_add\_calories";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "dish\_add\_name":

{

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Введите тип блюда, например суп");

addDishString =

$"INSERT INTO fooditem (name, calories, weight, chat\_id) VALUES ('{message.Text}',";

user.UserState = "dish\_add\_type";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "water\_reminder\_change\_1":

{

await using (var sqlCommand = dataSource.CreateCommand(

"update reminders set water\_remind = current\_date + interval '$1 day' + " +

$"time '{message.Text}:00' where chat\_id = {user.ChatId};"))

{

await sqlCommand.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"С каким интервалом вы хотите получать напоминания? (Введите время в формате HH:MM)");

user.UserState = "water\_reminder\_change\_2";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "water\_reminder\_change\_2":

{

await using (var sqlCommand = dataSource.CreateCommand(

$"update reminders set water\_remind\_interval = '{message.Text}:00' where chat\_id = {user.ChatId}"))

{

await sqlCommand.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Напоминание установлено на завтра.",

replyMarkup: KeyboardMarkups.MainMenuReplyKeyboard);

user.UserState = "";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "food\_reminder\_change\_1":

{

await using (var sqlCommand = dataSource.CreateCommand(

"update reminders set fooditem\_remind = current\_date + interval '$1 day' + " +

$"time '{message.Text}:00' where chat\_id = {user.ChatId};"))

{

await sqlCommand.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"С каким интервалом вы хотите получать напоминания? (Введите время в формате HH:MM)");

user.UserState = "food\_reminder\_change\_2";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "food\_reminder\_change\_2":

{

await using (var sqlCommand = dataSource.CreateCommand(

$"update reminders set fooditem\_remind\_interval = '{message.Text}' where chat\_id = {user.ChatId}"))

{

await sqlCommand.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Напоминание установлено на завтра.",

replyMarkup: KeyboardMarkups.MainMenuReplyKeyboard);

user.UserState = "";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "i\_have\_drunk":

{

if (message.Text != null)

{

var waterDrunk = int.Parse(message.Text);

var prevWaterDrunk = 0;

await using (var sqlCommand1 =

dataSource.CreateCommand(

$"select water\_drunk from users where chat\_id = {user.ChatId}"))

await using (var reader = await sqlCommand1.ExecuteReaderAsync())

{

while (await reader.ReadAsync()) prevWaterDrunk = reader.GetInt32(0);

}

await using (var sqlCommand2 = dataSource.CreateCommand(

"update users set water\_drunk = " +

$"{prevWaterDrunk + waterDrunk} where chat\_id = {user.ChatId}"))

{

await sqlCommand2.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

$"Отлично! За день вы уже выпили {prevWaterDrunk + waterDrunk} миллилитров",

replyMarkup: KeyboardMarkups.MainMenuReplyKeyboard);

}

user.UserState = "";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "i\_have\_eaten":

{

if (message.Text != null)

{

var eatenDishWeight = double.Parse(message.Text);

double calories = 0;

double caloriesEaten = 0;

double dishStandardWeight = 0;

await using (var sqlCommand1 = dataSource.CreateCommand(

$"select calories, weight from fooditem where \"name\" = '{eatenDishName}' and chat\_id = {user.ChatId}"))

await using (var reader = await sqlCommand1.ExecuteReaderAsync())

{

while (await reader.ReadAsync())

{

calories = reader.GetDouble(0);

dishStandardWeight = reader.GetDouble(1);

}

}

await using (var sqlCommand1 =

dataSource.CreateCommand(

$"select calories\_eaten from users where chat\_id = {user.ChatId}"))

await using (var reader = await sqlCommand1.ExecuteReaderAsync())

{

while (await reader.ReadAsync()) caloriesEaten = reader.GetDouble(0);

}

var sumCalories = Math.Round(caloriesEaten +

calories \* eatenDishWeight / dishStandardWeight, 2);

await telegramBotClient.SendTextMessageAsync(user.ChatId,

$"Отлично! За сегодня вы съели {sumCalories} ККал",

replyMarkup: KeyboardMarkups.MainMenuReplyKeyboard);

await using (var sqlCommand2 = dataSource.CreateCommand(

"update users set calories\_eaten = " +

$"{sumCalories} where chat\_id = {user.ChatId}"))

{

await sqlCommand2.ExecuteNonQueryAsync();

}

}

user.UserState = "";

await SetUserState(user.ChatId, user.UserState);

return;

}

}

switch (message.Text)

{

case "/start":

{

await using (var sqlCommand1 = dataSource.CreateCommand(

"SELECT EXISTS (SELECT 1 FROM users WHERE chat\_id = " +

user.ChatId + ")"))

await using (var reader = await sqlCommand1.ExecuteReaderAsync())

{

while (await reader.ReadAsync())

chatIdExists = reader.GetBoolean(0);

}

if (!chatIdExists)

{

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Привет, " + message.From?.Username + ". " +

"Я - бот который позволяет отслеживать потребляемые калории и выпитую воду\n" +

"Я тебя ещё не знаю, но очень хочу познакомиться\n" +

"Пожалуйста, добавь свои данные используя встроенную клавиатуру\n" +

"P.S. Если что-то сломалось, используй кнопку Вернуться",

replyMarkup: KeyboardMarkups.UserChangeReplyKeyboard);

await using (var sqlCommand2 =

dataSource.CreateCommand(

"INSERT INTO users (chat\_id) VALUES ($1)"))

{

sqlCommand2.Parameters.AddWithValue(user.ChatId);

await sqlCommand2.ExecuteNonQueryAsync();

}

await using (var sqlCommand3 =

dataSource.CreateCommand(

"INSERT INTO reminders (chat\_id) VALUES ($1)"))

{

sqlCommand3.Parameters.AddWithValue(user.ChatId);

await sqlCommand3.ExecuteNonQueryAsync();

}

await using (var sqlCommand4 =

dataSource.CreateCommand(

"INSERT INTO fooditem (chat\_id) VALUES ($1)"))

{

sqlCommand4.Parameters.AddWithValue(user.ChatId);

await sqlCommand4.ExecuteNonQueryAsync();

}

await using (var sqlCommand5 =

dataSource.CreateCommand(

"INSERT INTO fooditems\_per\_day (chat\_id) VALUES ($1)"))

{

sqlCommand5.Parameters.AddWithValue(user.ChatId);

await sqlCommand5.ExecuteNonQueryAsync();

}

user.UserState = "general\_data\_change";

await SetUserState(user.ChatId, user.UserState);

}

else

{

await telegramBotClient.SendTextMessageAsync(message.Chat, "Привет!",

replyMarkup: KeyboardMarkups.MainMenuReplyKeyboard);

}

return;

}

case "Данные о себе":

{

await telegramBotClient.SendTextMessageAsync(message.Chat, "Меню данных о себе",

replyMarkup: KeyboardMarkups.UserReplyKeyboard);

return;

}

case "Еда":

{

var fooditems = "";

var dishesExists = false;

await using (var sqlCommand1 = dataSource.CreateCommand(

$"SELECT EXISTS (SELECT 1 FROM fooditem WHERE chat\_id = {user.ChatId} and \"name\" is not null)"))

await using (var reader = await sqlCommand1.ExecuteReaderAsync())

{

while (await reader.ReadAsync()) dishesExists = reader.GetBoolean(0);

}

if (dishesExists)

{

await using (var sqlCommand =

dataSource.CreateCommand(

$"select f.\"name\" from fooditem f inner join fooditems\_per\_day fpd on f.chat\_id = fpd.chat\_id where f.chat\_id = {user.ChatId}"))

await using (var reader = await sqlCommand.ExecuteReaderAsync())

{

while (await reader.ReadAsync())

for (var i = 0; i < reader.FieldCount; i++)

fooditems = fooditems + reader.GetString(i) + '\n';

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Меню на сегодня\n" + fooditems,

replyMarkup: KeyboardMarkups.DishesReplyKeyboard);

}

else

{

await telegramBotClient.SendTextMessageAsync(message.Chat,

"У вас нет ни одного блюда, добавьте хотя-бы одно блюдо" +

" и создайте рацион чтобы увидеть этот список.",

replyMarkup: KeyboardMarkups.DishesReplyKeyboard);

user.UserState = "";

await SetUserState(user.ChatId, user.UserState);

}

return;

}

case "Напоминания":

{

await telegramBotClient.SendTextMessageAsync(message.Chat, "Меню напоминаний",

replyMarkup: KeyboardMarkups.RemindersReplyKeyboard);

return;

}

case "Расчеты":

{

await telegramBotClient.SendTextMessageAsync(message.Chat, "Меню с расчетами",

replyMarkup: KeyboardMarkups.CountsReplyKeyboard);

return;

}

case "Добавить блюдо":

{

user.UserState = "dish\_add\_name";

await SetUserState(user.ChatId, user.UserState);

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Введите название блюда");

return;

}

case "Данные о себе сейчас":

{

double userWeight = 0;

var userHeight = 0;

var userAge = 0;

double caloriesPerDay = 0;

double caloriesEaten = 0;

var waterPerDay = 0;

var waterDrunk = 0;

await using (var sqlCommand1 = dataSource.CreateCommand(

"SELECT weight, height, age, calories\_per\_day, " +

"calories\_eaten, water\_per\_day, water\_drunk FROM users WHERE chat\_id = " +

user.ChatId))

await using (var reader = await sqlCommand1.ExecuteReaderAsync())

{

while (await reader.ReadAsync())

{

userWeight = reader.GetDouble(0);

userHeight = reader.GetInt32(1);

userAge = reader.GetInt32(2);

caloriesPerDay = reader.GetDouble(3);

caloriesEaten = reader.GetDouble(4);

waterPerDay = reader.GetInt32(5);

waterDrunk = reader.GetInt32(6);

}

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Данные о пользователе\n" +

$"Ваш вес {userWeight}\n" +

$"Ваш рост {userHeight}\n" +

$"Ваш возраст {userAge}\n" +

$"Рекомендуемое дневное количество калорий {caloriesPerDay}\n" +

$"Потребленное количество калорий за сегодня {caloriesEaten}\n" +

$"Рекомендуемое количество выпитой жидкости {waterPerDay}\n" +

$"Выпитое за сегодня количество жидкости {waterDrunk}\n");

return;

}

case "Изменить данные о себе":

{

if (user.UserState != "")

return;

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Что будем менять?",

replyMarkup: KeyboardMarkups.UserChangeReplyKeyboard);

user.UserState = "general\_data\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "Посчитать рекомендуемое количество калорий в день":

{

double userWeight = 0;

var userHeight = 0;

var userAge = 0;

string? userGender = null;

await using (var sqlCommand1 = dataSource.CreateCommand(

"SELECT weight, height, age, gender FROM users WHERE chat\_id = " +

user.ChatId))

await using (var reader = await sqlCommand1.ExecuteReaderAsync())

{

while (await reader.ReadAsync())

{

userWeight = reader.GetDouble(0);

userHeight = reader.GetInt32(1);

userAge = reader.GetInt32(2);

userGender = reader.GetString(3);

}

}

if (userHeight == 0 || userAge == 0 || userWeight == 0 ||

userGender == null)

{

await telegramBotClient.SendTextMessageAsync(user.ChatId,

"Каких-то данных не хватает. Закончите заполнение данных о себе и возвращайтесь!");

return;

}

var userRecommendedCalories =Math.Round((userWeight \* 10 + userHeight \* 6.25 -

userAge \* 5 -

(userGender == "М" ? +5 : -161)) \* 1.2, 2);

await using (var sqlCommand2 = dataSource.CreateCommand(

"UPDATE users set calories\_per\_day = " + userRecommendedCalories +

" where chat\_id = " + user.ChatId))

{

await sqlCommand2.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"В среднем, рекомендуемое количество калорий в день для вас равно " +

userRecommendedCalories +

" ККал/день", replyMarkup: KeyboardMarkups.CountsReplyKeyboard);

return;

}

case "Посчитать водный баланс":

{

double userWeight = 0;

var userHeight = 0;

string? userGender = null;

await using (var sqlCommand1 = dataSource.CreateCommand(

"SELECT weight, height, gender FROM users WHERE chat\_id = " +

user.ChatId))

await using (var reader = await sqlCommand1.ExecuteReaderAsync())

{

while (await reader.ReadAsync())

{

userWeight = reader.GetDouble(0);

userHeight = reader.GetInt32(1);

userGender = reader.GetString(2);

}

}

if (userHeight == 0 || userWeight == 0 || userGender == null)

{

await telegramBotClient.SendTextMessageAsync(user.ChatId,

"Каких-то данных не хватает. Закончите заполнение данных о себе и возвращайтесь!");

return;

}

var userRecommendedWater = Math.Round((userHeight - 100) \*

((userHeight - userWeight <= userHeight - 100

? 30

: 35) - (userHeight - 100) \*

(userGender == "М" ? 0.04 : 0.06)), 2);

await using (var sqlCommand2 = dataSource.CreateCommand(

"UPDATE users set water\_per\_day = " + userRecommendedWater +

" where chat\_id = " + user.ChatId))

{

await sqlCommand2.ExecuteNonQueryAsync();

}

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Мы рекомендуем вам пить примерно " + userRecommendedWater +

" миллилитров в день");

return;

}

case "Количество воды за день":

{

if (user.UserState != "general\_data\_change")

return;

await telegramBotClient.SendTextMessageAsync(user.ChatId,

"Сколько вы бы хотели пить в день (Миллилитров)?");

user.UserState = "water\_per\_day\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "Количество калорий за день":

{

if (user.UserState != "general\_data\_change")

return;

await telegramBotClient.SendTextMessageAsync(user.ChatId,

"Сколько вы бы хотели есть в день (ККал)?");

user.UserState = "calories\_per\_day\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "Рост":

{

if (user.UserState != "general\_data\_change")

return;

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Введите свой рост в сантиметрах");

user.UserState = "height\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "Вес":

{

if (user.UserState != "general\_data\_change")

return;

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Введите свой вес в килограммах");

user.UserState = "weight\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "Возраст":

{

if (user.UserState != "general\_data\_change")

return;

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Введите свой возраст");

user.UserState = "age\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "Пол":

{

if (user.UserState != "general\_data\_change")

return;

await telegramBotClient.SendTextMessageAsync(message.Chat, "Выберите пол",

replyMarkup: KeyboardMarkups.GenderChangeReplyKeyboard);

user.UserState = "gender\_change";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "Настроить график питья":

{

user.UserState = "water\_reminder\_change\_1";

await SetUserState(user.ChatId, user.UserState);

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Во сколько вы хотите получить первое напоминание? (Введите время в формате HH:MM)");

return;

}

case "Настроить график питания":

{

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Во сколько вы хотите получить первое напоминание? (Введите время в формате HH:MM)");

user.UserState = "food\_reminder\_change\_1";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "Создать меню на завтра":

{

user.UserState = "menu\_create\_1";

await SetUserState(user.ChatId, user.UserState);

var dishesExists = false;

await using (var sqlCommand1 = dataSource.CreateCommand(

$"SELECT EXISTS (SELECT 1 FROM fooditem WHERE chat\_id = {user.ChatId} and \"name\" is not null)"))

await using (var reader = await sqlCommand1.ExecuteReaderAsync())

{

while (await reader.ReadAsync()) dishesExists = reader.GetBoolean(0);

}

if (dishesExists)

{

var dishesMarkupMaker =

await DishesMarkupMaker(user.ChatId);

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Какое блюдо будем есть на завтрак?",

replyMarkup: dishesMarkupMaker);

}

else

{

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Чтобы сделать меню, нужно добавить хотя-бы одно блюдо");

user.UserState = "";

await SetUserState(user.ChatId, user.UserState);

}

return;

}

case "Я поел":

{

var menuMarkupMaker = await MenuMarkupMaker(user.ChatId);

await telegramBotClient.SendTextMessageAsync(user.ChatId,

"Хорошо, что вы ели?",

replyMarkup: menuMarkupMaker);

user.UserState = "i\_have\_eaten";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "Я попил":

{

await telegramBotClient.SendTextMessageAsync(user.ChatId,

"Хорошо, введите количество выпитой воды в миллилитрах");

user.UserState = "i\_have\_drunk";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "Удалить блюдо":

{

var dishesMarkupMaker =

await DishesMarkupMaker(user.ChatId);

await telegramBotClient.SendTextMessageAsync(user.ChatId,

"Выберите блюдо которое хотите удалить",

replyMarkup: dishesMarkupMaker);

user.UserState = "delete\_dish";

await SetUserState(user.ChatId, user.UserState);

return;

}

default:

{

await telegramBotClient.SendTextMessageAsync(message.Chat,

"Вы что-то нажали и оно сломалось, попробуйте ещё раз");

return;

}

}

}

else

{

user.UserState = "";

await SetUserState(user.ChatId, user.UserState);

await telegramBotClient.SendTextMessageAsync(message.Chat, "Вернул в начало",

replyMarkup: KeyboardMarkups.MainMenuReplyKeyboard);

return;

}

return;

}

default:

{

return;

}

}

}

case UpdateType.CallbackQuery:

{

var message = update.CallbackQuery;

async Task<double> CaloriesGet(long chatId, string? name)

{

var calories = 0.0;

var weight = 0.0;

await using (var sqlCommand = dataSource.CreateCommand(

$"Select calories, weight from fooditem where \"name\" like '{name}' and chat\_id = " +

chatId))

await using (var reader = await sqlCommand.ExecuteReaderAsync())

{

while (await reader.ReadAsync())

{

calories = reader.GetDouble(0);

weight = reader.GetDouble(1);

}

}

return calories \* weight;

}

switch (user.UserState)

{

case "menu\_create\_1":

{

var dishesMarkupMaker = await DishesMarkupMaker(user.ChatId);

await telegramBotClient.SendTextMessageAsync(user.ChatId,

"Вы добавили " + message.Data + " калорийностью " +

await CaloriesGet(user.ChatId, message.Data) + " на завтрак.\n" +

"Что добавим на обед?", replyMarkup: dishesMarkupMaker);

await using var sqlCommand2 = dataSource.CreateCommand(

$"update fooditems\_per\_day set breakfast = (select fooditem\_id from fooditem where \"name\" like '{message.Data}' and chat\_id = {user.ChatId}) where chat\_id = ({user.ChatId}) ");

await sqlCommand2.ExecuteNonQueryAsync();

user.UserState = "menu\_create\_2";

await SetUserState(user.ChatId, user.UserState);

user.CaloriesSum += await CaloriesGet(user.ChatId, message.Data);

return;

}

case "menu\_create\_2":

{

var menuInlineKeyboard = await DishesMarkupMaker(user.ChatId);

await telegramBotClient.SendTextMessageAsync(user.ChatId,

"Вы добавили " + message.Data + " калорийностью " +

await CaloriesGet(user.ChatId, message.Data) + " на обед.\n" +

"Что добавим на ужин?", replyMarkup: menuInlineKeyboard);

await using var sqlCommand2 = dataSource.CreateCommand(

$"update fooditems\_per\_day set lunch = (select fooditem\_id from fooditem where \"name\" like '{message.Data}' and chat\_id = {user.ChatId}) where chat\_id = ({user.ChatId})");

await sqlCommand2.ExecuteNonQueryAsync();

user.UserState = "menu\_create\_3";

await SetUserState(user.ChatId, user.UserState);

user.CaloriesSum += await CaloriesGet(user.ChatId, message.Data);

return;

}

case "menu\_create\_3":

{

user.CaloriesSum += await CaloriesGet(user.ChatId, message.Data);

await telegramBotClient.SendTextMessageAsync(user.ChatId,

"Вы добавили " + message.Data + " калорийностью " +

await CaloriesGet(user.ChatId, message.Data) + " на ужин.\n" +

"Составление меню закончено, общая калорийность = " + user.CaloriesSum);

await using var sqlCommand2 = dataSource.CreateCommand(

$"update fooditems\_per\_day set dinner = (select fooditem\_id from fooditem where \"name\" like '{message.Data}') where chat\_id = ({user.ChatId}) and chat\_id = {user.ChatId}");

await sqlCommand2.ExecuteNonQueryAsync();

user.CaloriesSum = 0.0;

user.UserState = "";

await SetUserState(user.ChatId, user.UserState);

return;

}

case "i\_have\_eaten":

{

if (message.Data != "whole\_list")

{

eatenDishName = message.Data;

await telegramBotClient.SendTextMessageAsync(user.ChatId,

"Введите вес порции в граммах");

}

else

{

var wholeList = await DishesMarkupMaker(user.ChatId);

await telegramBotClient.SendTextMessageAsync(user.ChatId, "Всё меню",

replyMarkup: wholeList);

}

return;

}

case "delete\_dish":

{

await using var sqlCommand2 = dataSource.CreateCommand(

$"call delete\_dish ({user.ChatId}, '{message.Data}')");

await sqlCommand2.ExecuteNonQueryAsync();

await botClient.SendTextMessageAsync(user.ChatId, "Блюдо удалено", replyMarkup: KeyboardMarkups.MainMenuReplyKeyboard);

user.UserState = "";

await SetUserState(user.ChatId, user.UserState);

return;

}

}

return;

}

default:

throw new ArgumentOutOfRangeException();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

async Task Schedule()

{

Console.WriteLine($"Start listening for @{me.Username}");

//Console.ReadLine();

//cts.Cancel();

try

{

var stop = false;

while (!stop)

{

var chatId = new List<long>();

var fooditemRemind = new List<DateTime>();

var waterRemind = new List<DateTime>();

var fooditemInterval = new List<TimeSpan>();

var waterInterval = new List<TimeSpan>();

await using (var sqlCommand = dataSource.CreateCommand("select \* from reminders"))

await using (var reader = await sqlCommand.ExecuteReaderAsync())

{

while (await reader.ReadAsync())

{

chatId.Add(reader.GetInt64(0));

fooditemRemind.Add(reader.GetDateTime(1));

waterRemind.Add(reader.GetDateTime(2));

fooditemInterval.Add(reader.GetTimeSpan(3));

waterInterval.Add(reader.GetTimeSpan(4));

}

}

for (var i = 0; i < chatId.Count; i++)

{

if (fooditemRemind[i].ToString(CultureInfo.InvariantCulture) != "1/1/2000 12:00:00 AM"

&& fooditemInterval[i].ToString() != "00:00:00")

if (fooditemRemind[i] <= DateTime.Now)

{

await botClient.SendTextMessageAsync(chatId[i], "Пора есть");

var newFooditemReminder =

(fooditemRemind[i].Add(fooditemInterval[i]) != DateTime.Today.AddDays(1) ?

fooditemRemind[i] + fooditemInterval[i]

: DateTime.Today.AddDays(1).AddHours(10));

await using (var sqlCommand = dataSource.CreateCommand(

$"update reminders set fooditem\_remind = '{newFooditemReminder}' where chat\_id = {chatId[i]}"))

{

await sqlCommand.ExecuteNonQueryAsync();

}

}

if (waterRemind[i].ToString(CultureInfo.InvariantCulture) == "1/1/2000 00:00:00 AM"

|| waterInterval[i].ToString() == "00:00:00")

continue;

if (waterRemind[i] > DateTime.Now)

continue;

await botClient.SendTextMessageAsync(chatId[i], "Пора пить");

var newWaterReminder =

(waterRemind[i].Add(waterInterval[i]) != DateTime.Today.AddDays(1) ?

waterRemind[i] + waterInterval[i]

: DateTime.Today.AddDays(1).AddHours(10));

await using (var sqlCommand = dataSource.CreateCommand(

$"update reminders set fooditem\_remind = '{newWaterReminder}' where chat\_id = {chatId[i]}"))

{

await sqlCommand.ExecuteNonQueryAsync();

}

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

Task HandlePollingErrorAsync(ITelegramBotClient telegramBotClient, 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;

}

**ПРИЛОЖЕНИЕ Б**

**Реализация телеграм-бота UserData.cs**

namespace TG\_Bot;

public class UserData

{

public double CaloriesSum;

public long ChatId { get; set; }

public string UserState { get; set; }

}

**ПРИЛОЖЕНИЕ В**

**Реализация телеграм-бота KeyboardMarkups.cs**

using Telegram.Bot.Types.ReplyMarkups;

namespace TG\_Bot;

public static class KeyboardMarkups

{

public static readonly ReplyKeyboardMarkup MainMenuReplyKeyboard = new(

new List<KeyboardButton[]>

{

new[]

{

new("Данные о себе"),

new KeyboardButton("Еда"),

new KeyboardButton("Напоминания"),

new KeyboardButton("Расчеты")

}

}

)

{

ResizeKeyboard = true

};

public static readonly ReplyKeyboardMarkup UserReplyKeyboard = new(

new List<KeyboardButton[]>

{

new[]

{

new("Изменить данные о себе"),

new KeyboardButton("Данные о себе сейчас")

},

new[]

{

new KeyboardButton("Вернуться")

}

})

{

ResizeKeyboard = true

};

public static readonly ReplyKeyboardMarkup DishesReplyKeyboard = new(

new List<KeyboardButton[]>

{

new[]

{

new("Добавить блюдо"),

new KeyboardButton("Удалить блюдо")

},

new[]

{

new KeyboardButton("Создать меню на завтра")

},

new[]

{

new KeyboardButton("Я поел"),

new KeyboardButton("Я попил")

},

new[]

{

new KeyboardButton("Вернуться")

}

})

{

ResizeKeyboard = true

};

public static readonly ReplyKeyboardMarkup RemindersReplyKeyboard = new(

new List<KeyboardButton[]>

{

new[]

{

new("Настроить график питания"),

new KeyboardButton("Настроить график питья")

},

new[]

{

new KeyboardButton("Вернуться")

}

})

{

ResizeKeyboard = true

};

public static readonly ReplyKeyboardMarkup CountsReplyKeyboard = new(

new List<KeyboardButton[]>

{

new[]

{

new("Посчитать водный баланс"),

new KeyboardButton("Посчитать рекомендуемое количество калорий в день")

},

new[]

{

new KeyboardButton("Вернуться")

}

})

{

ResizeKeyboard = true

};

public static readonly ReplyKeyboardMarkup UserChangeReplyKeyboard = new(

new List<KeyboardButton[]>

{

new[]

{

new("Рост"),

new KeyboardButton("Пол")

},

new[]

{

new KeyboardButton("Количество воды за день"),

new KeyboardButton("Количество калорий за день")

},

new[]

{

new KeyboardButton("Возраст"),

new KeyboardButton("Вес")

},

new[]

{

new KeyboardButton("Вернуться")

}

})

{

ResizeKeyboard = true

};

public static readonly ReplyKeyboardMarkup GenderChangeReplyKeyboard = new(

new List<KeyboardButton[]>

{

new[]

{

new("М"),

new KeyboardButton("Ж")

}

})

{

ResizeKeyboard = true

};

}

**ПРИЛОЖЕНИЕ Г**

**Реализация телеграм-бота Config.cs**

using Telegram.Bot;

namespace TG\_Bot;

public static class Config

{

public static readonly string TgApiToken = "ApiToken";

public static readonly Dictionary<string, string> PgCredentials = new()

{

["host"] = "IP",

["username"] = "username",

["password"] = "Password",

["database"] = "database",

["port"] = "5432",

};

}