

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ТЕЛЕГРАМ-БОТ «НАПОМИНАНИЯ»

Студент 30.05.25 Чабуев Чабуев Ростислав Павлович
(подпись, дата) (ФИО полностью)

Руководитель Юрова 30.05.25 Юрова О.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр Бычков 30.05.2025 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой Митрохин Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от
2 июня 2025 г. № 8)

Секретарь ГЭК Гурин Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 06 » « 05 » 20 25 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Чабуев Ростислав Павлович гр. 21ВВП2 факультета ВТ

2. Тема работы «Telegram-бот "Напоминания"»

Тема утверждена приказом ПГУ № 601/о от "6" мая 2025 г.

3. Руководитель бакалаврской работы к.т.н., доцент кафедры ВТ ПГУ Юрова О. В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Необходимо спроектировать и разработать Telegram-бота, который будет управлять
напоминаниями с расширенным функционалом, включая: создание обычных, с
предупреждением и повторяющихся напоминаний, гибкие настройки времени и
периодичности, удобный интерфейс для просмотра и удаления напоминаний, а также
надежное хранение данных и точную отправку уведомлений.

5. Перечень подлежащих разработке вопросов _____

Требуется произвести анализ предметных областей: необходимо изучить возможные альтернативы данному инструменту, другие площадки, на которых возможно базирование телеграм-бота, а также способы разработки бота как создание на базе Visual Studio 2022 и языка программирования C#.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	24	06.05.2025-08.05.2025	<i>фиа</i>
2.	Изучение аналогов	16	09.05.2025-10.05.2025	<i>фиа</i>
3.	Анализ требований к телеграм-боту	16	11.05.2025-12.05.2025	<i>фиа</i>
4.	Выбор технологий разработки	24	13.05.2025-15.05.2025	<i>фиа</i>
5.	Настройка среды разработки	8	16.05.2025-17.05.2025	<i>фиа</i>
6.	Разработка телеграм-бота	64	18.05.2025-25.05.2025	<i>фиа</i>
7.	Тестирование бота	16	26.05.2025-27.05.2025	<i>фиа</i>
8.	Оформление пояснительной записки	24	28.05.2025-30.05.2025	<i>фиа</i>

Дата выдачи задания " 6 " мая 2025 г.

Руководитель бакалаврской работы *фиа* Юрова О.В.
(подпись, дата)

Задание к исполнению принял студент *Чабуев* Чабуев Р.П.
(подпись, дата)

Работу к защите допустить
Декан факультета *Л.Р. Фионова* Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 92 листов, 30 рисунков, 2 таблицы, 12 источников, 3 приложения.

ТЕЛЕГРАМ-БОТ, ДОБАВЛЕНИЕ НАПОМИНАНИЙ, НАПОМИНАНИЯ, БАЗА ДАННЫХ, C#, VISUAL STUDIO, SQLITE, QUARTZ

Цель работы – разработать Telegram-бота, который будет управлять напоминаниями с расширенным функционалом, которые упростят повседневную жизнь.

Одним из путей достижения поставленной задачи является телеграм-бот, обеспечивающий выполнение следующих функций: создание обычных, с предупреждением и повторяющихся напоминаний, гибкие настройки времени и периодичности, удобный интерфейс для просмотра и удаления напоминаний, надежное хранение данных и точную отправку уведомлений.

В конечном итоге разработано приложение, полностью соответствующий цели данной выпускной квалификационной работы. В ходе достижения результата, были использованы следующие технологии: Visual Studio 2022, C#, SQLite, Quartz. Приложение протестировано и готово к эксплуатации.

					ПГУ1.090301.48.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Чабуев Р.П.	<i>Чабуев Р.П.</i>	20.05.25	Лит.	Лист	Листов
Провер.		Юрова О.В.	<i>Юрова О.В.</i>	20.05.25		2	92
Н. Контр.		Бычков А. С.	<i>Бычков А. С.</i>	20.05.25	Телеграм-бот «Напоминания» Пояснительная записка ФВТ, каф. ВТ гр. 21ВВП2		
Утверд.		Митрохин М.А.	<i>Митрохин М.А.</i>	20.05.25			

Содержание

Перечень сокращений и обозначений	5
Введение.....	6
1 Telegram-бот «Напоминания»	7
1.1 Анализ предметной области	7
1.2 Обзор аналогов	8
1.3 Постановка задачи	14
1.3.1 Назначение бота	14
1.3.2 Функциональные требования	14
1.3.3 Команды бота	15
1.3.4 Технические требования	15
1.3.5 Дополнительные возможности	15
1.3.6 Критерии успешной реализации	15
1.3.7 Минимальные системные требования	16
1.4 Архитектура приложения	16
1.4.1 Общая структура	16
1.4.2 Основные компоненты	17
1.4.3 Взаимодействие компонентов	17
1.5 Выбор инструментов разработки	19
1.5.1 Язык программирования	19
1.5.2 Библиотеки и фреймворки	19
1.5.3 Среда разработки	20
1.5.4 Настройка среды разработки Visual Studio 2022	21
2 Проектирование и разработка	28
2.1 Основные компоненты	30
2.1.1 BotBackgroundService	30
2.1.2 ReminderService	32
2.1.3 DatabaseService	33
2.1.4 ReminderJob	35

2.2 Тестирование	36
3 Безопасность жизнедеятельности	46
3.1 Правила безопасности при использовании ПК	46
3.2 Правила безопасности при использовании смартфона	46
Заключение	50
Список использованных источников	51
Приложение А – UML – диаграммы	52
Приложение Б – Листинг программы	57
Приложение В – Презентация	82

Перечень сокращений и обозначений

БД – база данных

ОС – операционная система

ПК – персональный компьютер

API (Application Programming Interface) – программный интерфейс приложения

DI (Dependency Injection) – паттерн проектирования, который позволяет объектам программы внедрять зависимости

Git (Global Information Tracker) – распределенная система управления версиями

GSM (Groupe Special Mobile) – стандарт сотовой связи

ID (identifier) – идентификатор

IDE (Integrated Development Environment) – программное обеспечение, которое помогает разработчикам писать и тестировать код

IOS (Iphone Operating System) – мобильная операционная система

LTE (Long Term Evolution) – долговременное развитие относящееся к технологии 4G

MacOS (Macintosh Operating System) – операционная система персонального компьютера

ORM (Object-Relational Mapping) – объектно-реляционное отображение

SDK (Software Development Kit) – пакет инструментов, предназначенный для разработки программного обеспечения

SQL (Structured Query Language) – язык структурированных запросов

UML (Unified Modeling Language) – графический язык, используемый для визуализации, описания и документирования программных систем и бизнес-процессов

URL (Uniform Resource Locator) – адрес ресурса в интернете

VS (Visual Studio) – линейка систем разработки программного обеспечения от компании Microsoft

					ПГУ1.090301.48.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В современном мире, где количество задач и дел постоянно растет, эффективное управление временем становится критически важным. Многие люди сталкиваются с проблемой забывчивости, что приводит к пропуску важных событий, дедлайнов и встреч. Традиционные методы напоминаний, такие как бумажные списки дел или встроенные календари, не всегда удобны и доступны.

Telegram-боты предоставляют удобное решение этой проблемы, позволяя пользователям создавать, управлять и получать напоминания прямо в мессенджере. Благодаря широкой распространенности Telegram и его кроссплатформенности, такие боты становятся популярными инструментами для личного и корпоративного использования.

Целью работы разработать Telegram-бота, который будет управлять напоминаниями с расширенным функционалом, включая:

- создание обычных, с предупреждением и повторяющихся напоминаний;
- гибкие настройки времени и периодичности;
- удобный интерфейс для просмотра и удаления напоминаний;
- надежное хранение данных и точную отправку уведомлений.

1 Telegram-бот «Напоминания»

1.1 Анализ предметной области

Общение через Telegram-бота потеснило традиционные списки дел и ручные напоминания. В настоящее время многие пользователи предпочитают доверять планирование задач автоматизированным помощникам, а не записывать их в блокнот или держать в памяти. Современные боты умеют не только напоминать о событиях в заданное время, но и анализировать повторяющиеся дела, синхронизироваться с календарями и даже предлагать оптимальное расписание.

Как правило, бот хранит данные о всех запланированных пользователем событиях — от разовых покупок до регулярных встреч. По этим данным можно определить загруженность человека, его привычки (например, время пробуждения или обеденного перерыва), а также важные даты (дни рождения, дедлайны).

Разработчики Telegram-ботов [1], как крупные компании, так и независимые команды используют эти данные для двух основных целей:

- улучшения функционала – например, предлагая шаблоны напоминаний или интеллектуальное расписание;
- персонализации – подстраивая интерфейс и уведомления под ритм жизни конкретного пользователя.

Но не только создатели ботов заинтересованы в такой информации.

Например:

- корпоративные клиенты могут использовать бота для координации задач сотрудников;
- образовательные платформы для напоминаний студентам о лекциях и дедлайнах;
- медицинские учреждения для уведомлений пациентов о приёме лекарств.

					ПГУ1.090301.48.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

Кроме того, аналитика использования такого бота позволяет выявлять популярные временные промежутки для планирования дел, определять среднюю загруженность пользователей и даже прогнозировать их потребности. Например, предлагать вечерние напоминания тем, кто чаще всего планирует дела на утро.

1.2 Обзор аналогов

Изучив множество источников сети интернет, выяснилось, что приложения с напоминаниями имеют большую популярность. К сожалению, про телеграм-ботов, такое не скажешь. Так как пользователи привыкли к удобству в разных приложениях. Это говорит о том, что в ботах нет такого функционала, как в популярных приложениях с напоминаниями. Разберем несколько примеров ботов и сравним с приложениями.

Первым на очереди «бот-напоминалка» (@уапаромпу_bot) — бот для простого и удобного управления задачами.

В нем создаются напоминания на нужные даты и время, планируются дела и автоматически приходят уведомления о важных делах и задачах прямо в Telegram. Данный бот представлен на рисунках 1, 2.

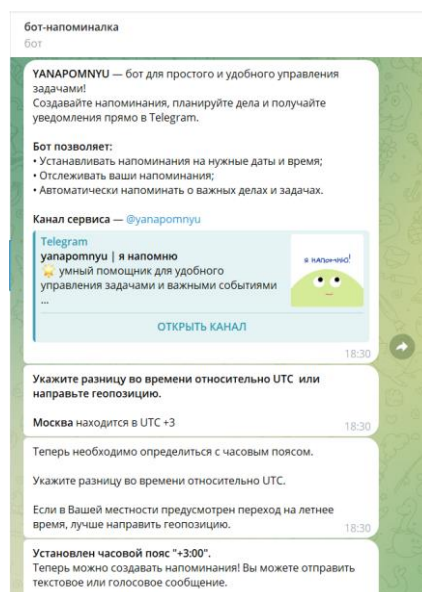


Рисунок 1 — Начальное окно приветствия

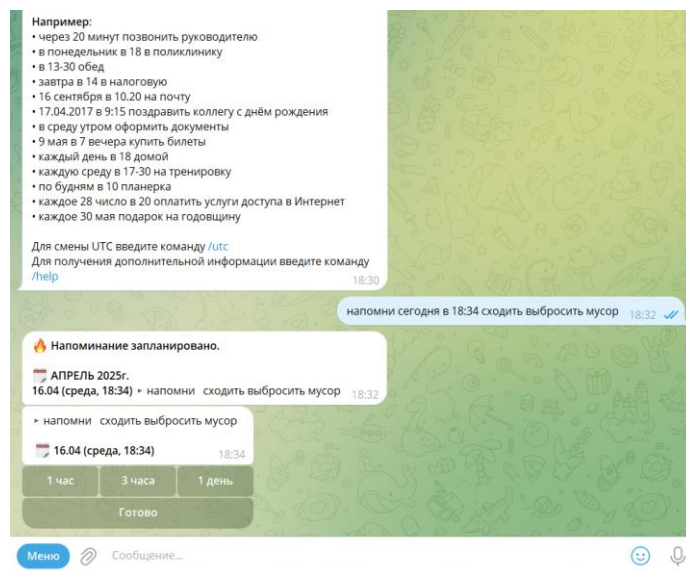


Рисунок 2 – Создание и получение напоминания

Вторым по счету бот «Skeddy» (@skeddybot) — это удобный бот внутри Telegram, который помогает не забывать о важных делах, встречах и привычках. Он работает как персональный секретарь, позволяя быстро создавать напоминания без установки дополнительных приложений. Функционал схож с телеграм-ботом, что представлен ранее, но есть некоторые преимущества и недостатки, ознакомиться можно на рисунках 3-6.

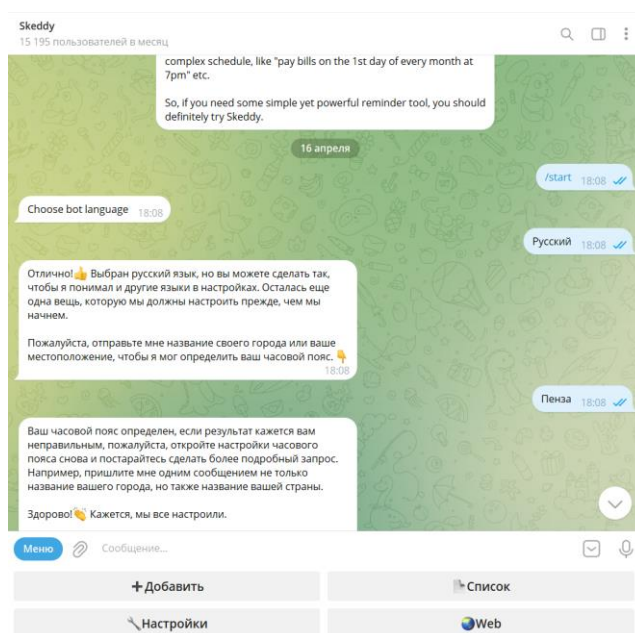


Рисунок 3 – Начальное окно приветствия и настройки бота

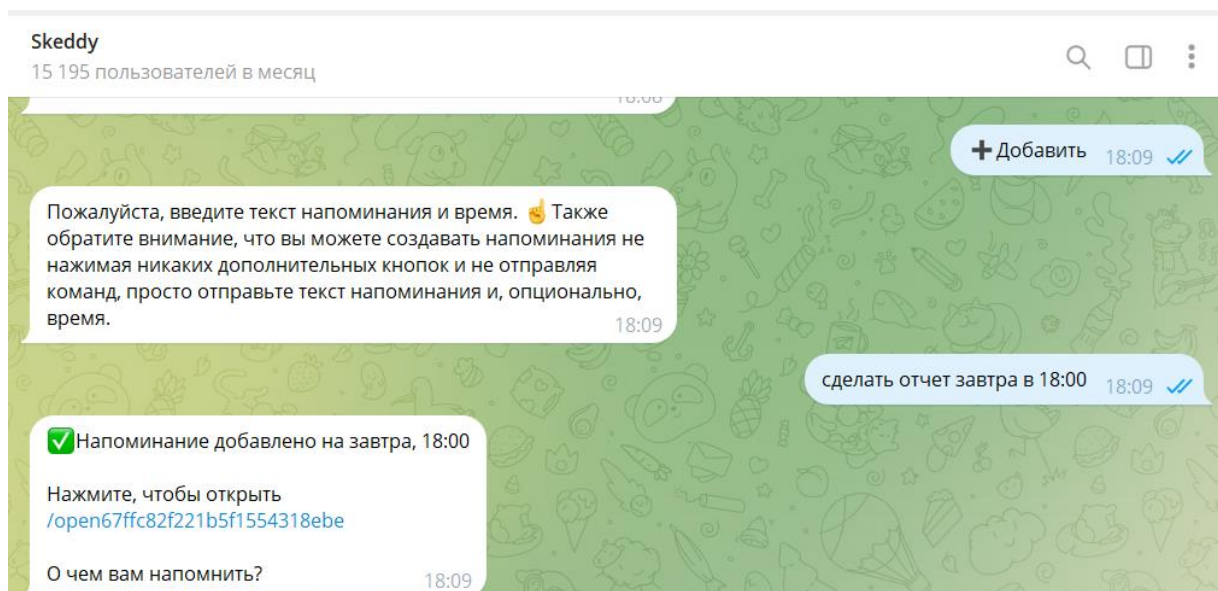


Рисунок 4 – Создание напоминания

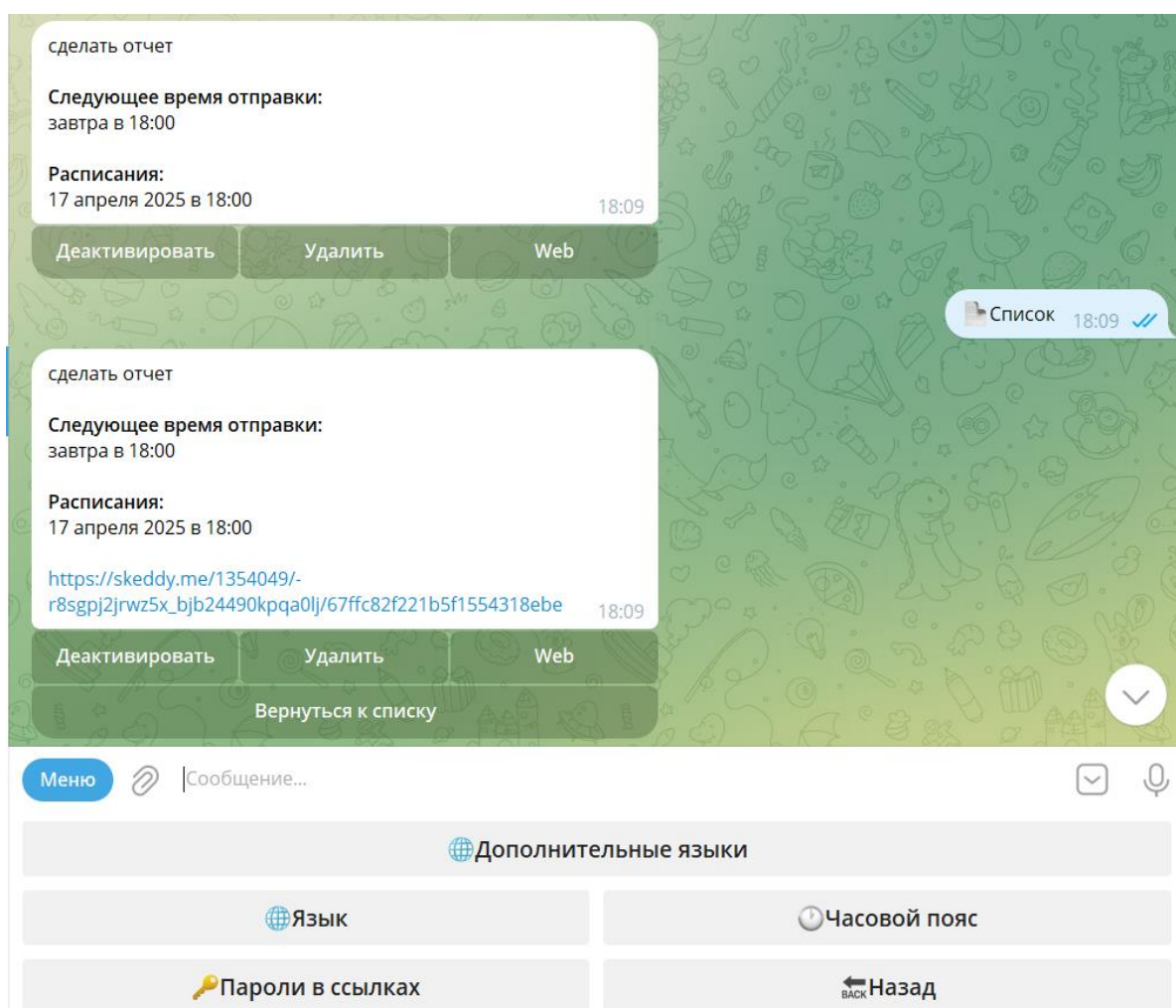


Рисунок 5 – Список созданных напоминаний

Рисунок 6 – Веб-версия напоминания

Третьим ботом был выбран «To_do_list_bot» (@to_list_do_bot) – это самый простой телеграм-бот для создания напоминаний из представленных ранее. Его базовый функционал – это создать, просмотреть и удалить напоминание, увидеть это можно на рисунке 7.

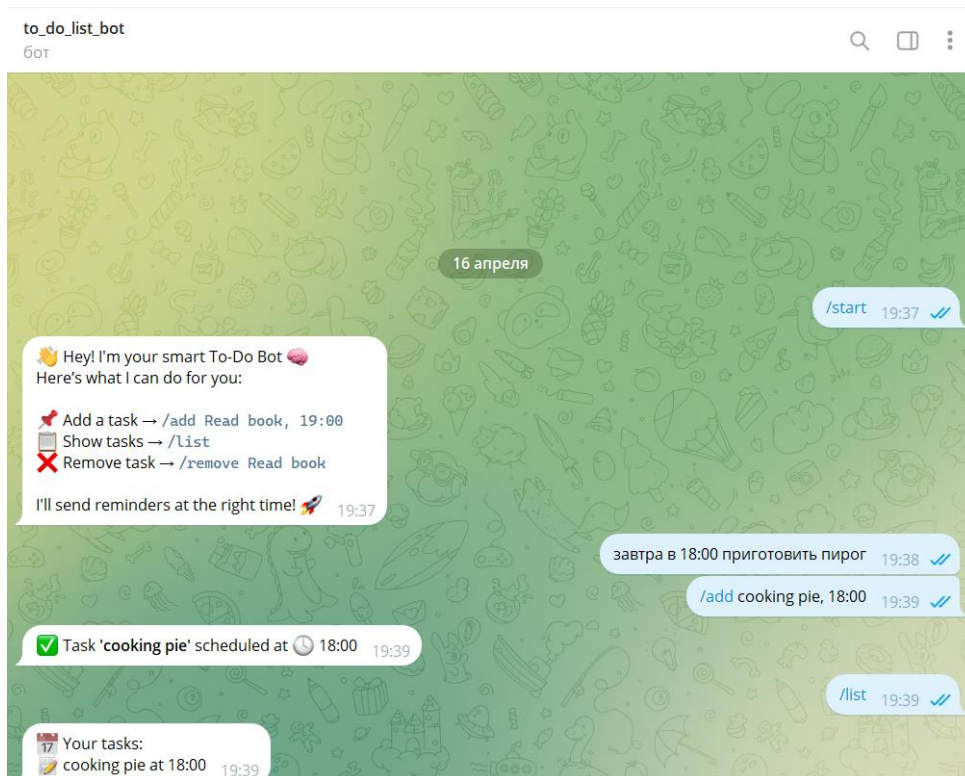


Рисунок 7 – Интерфейс и создание напоминаний в телеграм-боте

Ознакомившись с ботами, следующий выбор пал на популярное приложение, такое как, Google Keep [2] – это простой сервис для создания заметок и напоминаний от Google. В который можно добавлять напоминания к любой заметке, и они будут полностью синхронизированы по воздуху на всех устройствах. Скриншоты сервиса представлены ниже на рисунке 8.

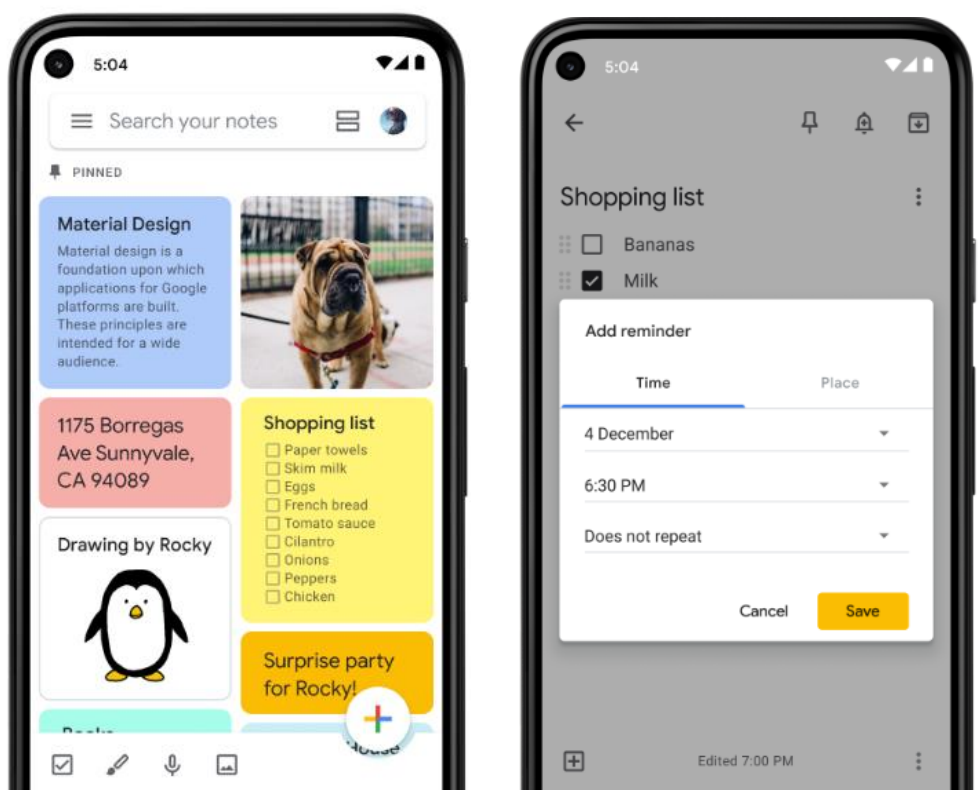


Рисунок 8 – Интерфейс Google Keep

В данном сервисе есть гибкие настройки времени:

- конкретная дата и время;
- повторяющиеся напоминания;
- геолокационные напоминания.

Не менее популярным приложением среди заметок является Microsoft To Do [3] – это менеджер задач от Microsoft с удобными напоминаниями. В котором можно ставить задачи с дедлайнами, повторяющиеся задачи и

синхронизация с Outlook и почтой. Уведомления приходят на телефон и компьютер. Интерфейс программы представлен на рисунке 9.

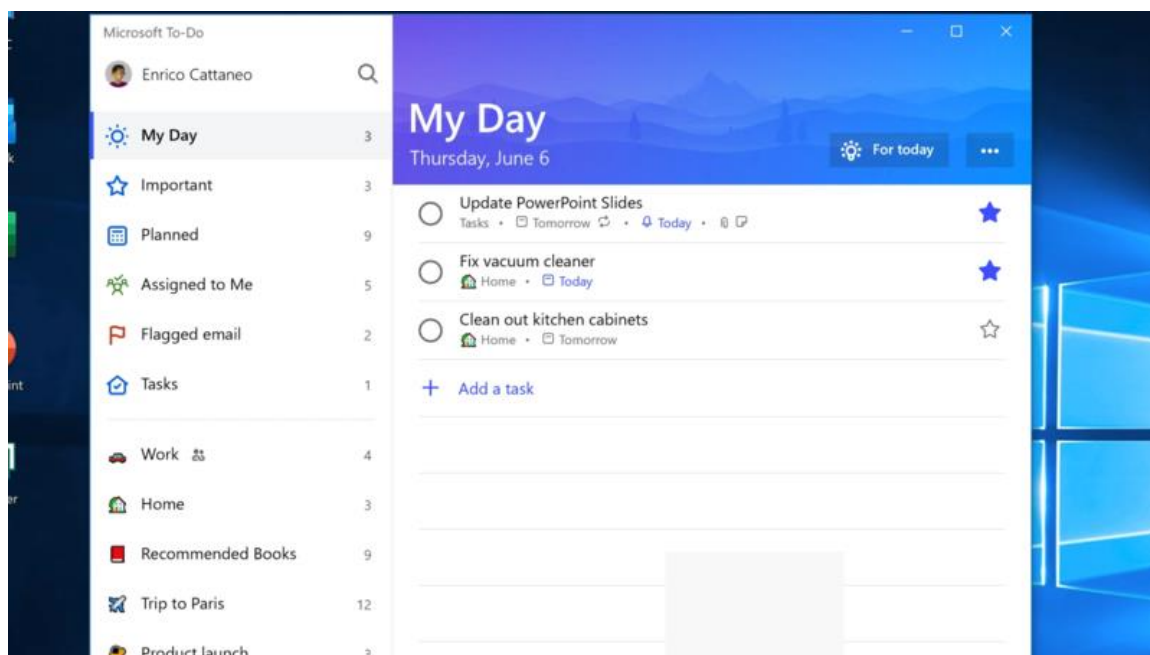


Рисунок 9 – Интерфейс Microsoft To Do

После тестирования сервисов для напоминаний, включая Google Keep, Microsoft To Do и Telegram-ботов, были сделаны следующие выводы:

- Google Keep [2] показал себя как оптимальное решение для быстрых личных напоминаний благодаря простоте использования и интеграции с сервисами Google. Однако для сложного планирования задач он не подходит.
- Microsoft To Do [3] оказался более мощным инструментом для управления задачами, особенно в экосистеме Microsoft, но требует учетной записи и менее удобен для геолокационных напоминаний.

А также, если говорить о телеграм-боте оказалось, что @уапаромпуц_bot – лучший выбор для простых, но гибких напоминаний, если не нужна интеграция с календарями. @skeddybot – удобнее за счёт кнопочного интерфейса и веб-доступа, но функционал всё ещё ограничен. @to_list_do_bot – слишком примитивен, подойдёт только для очень краткосрочных задач. А также была составлена таблица 1 с преимуществами и недостатками.

Таблица 1 – Преимущества и недостатки телеграм-ботов и дополнительных сервисов для напоминаний

Задачи	@yanapomny_bot	@skeddybot	@to_list_do_bot	Google Keep	Microsoft To Do
Простота использования	+	+	-	+	+
Синхронизация с календарями	-	-	-	+	+
Поддержка нескольких языков	-	+	-	+	+
Создание голосовых напоминаний	-	-	-	+	-
Гибкие настройки	+	+	-	+	+
Веб-версия	-	+	-	+	+
Повторяющиеся задачи	+	+	-	+	+
Кроссплатформенность	Только Telegram	Telegram+ Web	Только Telegram	Все платформы	Все платформы

1.3 Постановка задачи

Постановка задачи для Telegram-бота "ReminderBot" (@botzapomni_Bot) была следующей:

1.3.1 Назначение бота

Telegram-бот предназначен для создания, управления и отправки напоминаний пользователям в указанное время. В отличие от аналогов, имеются разделения на добавление напоминаний.

1.3.2 Функциональные требования

С помощью кнопки «Добавить напоминание»:

- обычное напоминание, которое придет в конкретную дату и время;

- с предупреждением, напоминание придет за два дня и за один день, до случившегося события;

- с повторением, напоминание можно создать на выбор, а точнее: «каждый час»; «каждый день»; «каждую неделю».

1.3.3 Команды бота

В случае того, что в худшем из вариантов использования телеграм-бота не будут работать кнопки управления, возможно, как и в любом другом боте, воспользоваться командами, которые представлены ниже:

- /start – начало работы, показывает меню;
- /add – создать новое напоминание;
- /list – показать список активных напоминаний;
- /delete [ID] – удалить напоминание по его ID;
- /help – справка по использованию бота.

1.3.4 Технические требования

Для разработки телеграм-бота был выбран язык программирования – C#. В качестве среды разработки использовалась Visual Studio 2022, в которой были применены следующие библиотеки для Telegram:

- Telegram.Bot;
- Microsoft.EntityFrameworkCore.

В качестве планировщика задач был выбран – Quartz.NET, связке с базой данных – SQLite.

1.3.5 Дополнительные возможности

- ReplyKeyboardButton – кнопки, которые появляются в нижней панели ввода;
- InlineKeyboardButton – кнопки, встроенные в сообщение.

1.3.6 Критерии успешной реализации

Можно понять, только если бот работает стабильно, напоминания приходят точно в срок, а также если для работы доступен удобный и понятный интерфейс.

1.3.7 Минимальные системные требования

Для пользователей имеются минимальные системные требования, такие как:

- устройство – смартфон/ПК/планшет с поддержкой Telegram;
- ОС: Android 7+, iOS 13+, Windows 10+, macOS 10.15+;
- Telegram: Версия 7.4+ (для корректной работы голосовых сообщений);
- интернет: Мобильный интернет (3G+/LTE) или Wi-Fi.

Для разработки были выявлены следующие минимальные системные требования:

- IDE: Visual Studio 2022, JetBrains Rider, VS Code;
- SDK: .NET 6.0.

Ниже на рисунке 10 представлена UML-диаграмма использования телеграм-бота.



Рисунок 10 – UML-диаграмма использования

1.4 Архитектура приложения

1.4.1 Общая структура

Приложение построено по многослойной архитектуре с четким разделением ответственностей:

- слой взаимодействия с пользователем, что включает в себя обработку команд Telegram и отправка уведомлений;
- бизнес-логика, в нее входит управление напоминаниями (создание, редактирование, удаление);
- слой данных представляет собой работу с базой данных (SQLite) и хранение информации о напоминаниях;
- в слой планирования задач входит Quartz.NET для отложенных уведомлений.

1.4.2 Основные компоненты

Компоненты необходимые для бесперебойной и стабильной работы Telegram-бота представлены в таблице 2.

Таблица 2 – Основные компоненты

Компоненты	Назначение	Технологии
BotBackgroundService	Фоновый сервис для обработки сообщений и команд пользователя.	Telegram.Bot, .NET Hosting
ReminderService	Основная логика работы с напоминаниями (парсинг, валидация, планирование).	Quartz.NET, C#
DatabaseService	Взаимодействие с базой данных.	SQLite, Entity Framework
ReminderJob	Задача Quartz для отправки уведомлений в указанное время.	Quartz.NET

1.4.3 Взаимодействие компонентов

Пользователь взаимодействует с ботом, отправляя команду на создание напоминания. Бот, в свою очередь, передаёт эту команду в фоновый сервис,

который занимается дальнейшей обработкой. Этот сервис вызывает компонент напоминаний — ReminderService.

ReminderService отвечает за основную бизнес-логику: он сохраняет информацию о напоминании с помощью сервиса работы с базой данных и параллельно планирует выполнение напоминания через систему планирования задач Quartz.NET.

Quartz.NET по расписанию запускает задачу — ReminderJob, которая инициирует отправку уведомления. Для этого ReminderJob обращается обратно к боту, чтобы тот отправил сообщение пользователю.

Таким образом, модули взаимодействуют последовательно по логике: от пользователя — к боту, от бота — в обработчик, оттуда — к сервису напоминаний, затем к базе и планировщику, и в назначенное время происходит обратный путь — от планировщика к боту и обратно к пользователю. UML-диаграмма взаимодействия представлена ниже на рисунке 11.

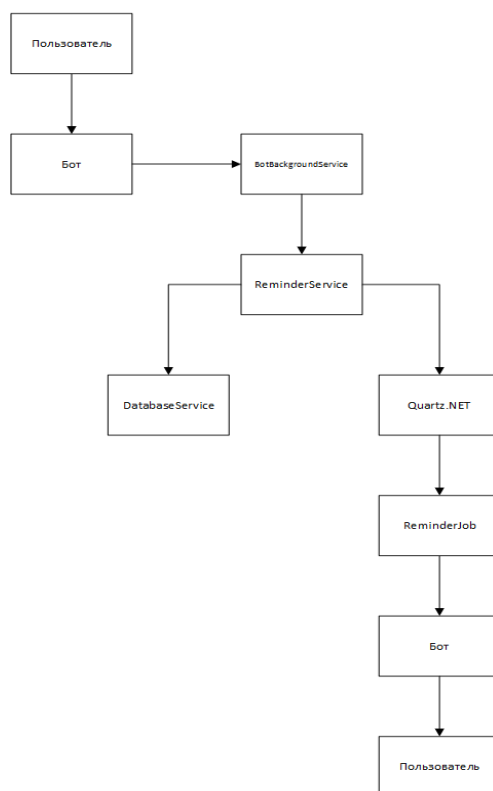


Рисунок 11 – UML-диаграмма взаимодействия компонентов

1.5 Выбор инструментов разработки

1.5.1 Язык программирования

C# (.NET 8.0) [4] – этот язык программирования включает в себя высокую производительность, полную поддержку асинхронного программирования, а также наличие богатой экосистемы библиотек.

1.5.2 Библиотеки и фреймворки

В разрабатываемом Telegram-боте для управления напоминаниями применяются следующие ключевые библиотеки, каждая из которых выполняет важную функцию в архитектуре приложения:

Telegram.Bot – данная библиотека предоставляет удобный интерфейс для работы с Telegram Bot API. Она отвечает за:

- прием и обработку входящих сообщений от пользователей;
- отправку текстовых уведомлений и медиафайлов;
- управление клавиатурами и inline-режимом.

Библиотека значительно упрощает интеграцию бота с Telegram, предоставляя готовые методы для всех основных операций.

Quartz.NET [5] – мощный планировщик задач, который используется для:

- точного выполнения напоминаний в указанное пользователем время;
- поддержки повторяющихся задач (ежедневные, еженедельные напоминания);
- обеспечения надежности - задачи сохраняются между перезапуском приложения.

Quartz.NET позволяет гибко настраивать расписание через триггеры и гарантирует своевременное выполнение задач.

Microsoft.Data.Sqlite [6] – это легковесный ADO.NET провайдер для работы с SQLite, который:

- обеспечивает взаимодействие с локальной файловой базой данных;

- позволяет выполнять SQL-запросы для хранения и извлечения данных о напоминаниях;

- поддерживает транзакции для обеспечения целостности данных.

Эта библиотека выбрана за свою простоту и эффективность для небольших проектов.

Entity Framework Core – объектно-реляционный маппер (ORM), который:

- позволяет работать с базой данных через объекты C#;

- автоматизирует создание и миграцию схемы базы данных;

- предоставляет LINQ-интерфейс для написания типобезопасных запросов.

Entity Framework значительно ускоряет разработку, избавляя от необходимости писать сырые SQL-запросы.

Каждая из этих библиотек была выбрана исходя из требований проекта: необходимость надежной работы с Telegram API, точного планирования задач, простого локального хранения данных и удобного доступа к ним. Вместе они образуют мощную, но легковесную основу для Telegram-бота, обеспечивая все необходимые функции без избыточного усложнения архитектуры.

1.5.3 Среда разработки

Для создания Telegram-бота использовалась современная среда разработки Microsoft Visual Studio 2022 — профессиональная IDE для платформы .NET, предоставляющая полный набор инструментов для эффективной работы над проектом [7].

Microsoft Visual Studio 2022 была выбрана в качестве основной среды разработки для создания Telegram-бота по нескольким ключевым причинам.

Во-первых, Visual Studio обеспечивает полную поддержку .NET 8.0 — целевой платформы проекта. Это гарантирует совместимость со всеми современными функциями языка C# и .NET, включая оптимизации производительности и улучшенную работу с асинхронным кодом.

Во-вторых, среда предлагает глубокую интеграцию с системой контроля версий Git, что значительно упрощает командную разработку и управление изменениями в коде. Встроенные инструменты позволяют эффективно вести разработку даже в рамках распределённой команды.

Одним из наиболее важных преимуществ является мощный отладчик Visual Studio. Он предоставляет все необходимые инструменты для поиска и исправления ошибок: точки останова, пошаговое выполнение кода, а также мониторинг переменных в реальном времени с возможностью изменения их значений прямо во время отладки.

Для управления зависимостями проекта в Visual Studio используется встроенный NuGet-менеджер. Этот инструмент позволяет удобно устанавливать, обновлять и удалять пакеты, такие как Telegram.Bot, Quartz.NET и другие, прямо из интерфейса IDE, без необходимости ручного редактирования файлов проекта.

1.5.4 Настройка среды разработки Visual Studio 2022

Перед началом работы нужно убедиться в том, что установлены все инструменты и компоненты, для дальнейшей настройки среды разработки. Проверить можно через панель инструментов, далее «Получить средства и компоненты» и просмотреть установленные компоненты. Должно быть установлено:

- .NET 6 SDK (или новее);
- ASP.NET Core Runtime;
- NuGet Package Manager.

Следующим шагом будет создание проекта. Выбрав кнопку создать проект последует выбор шаблона на языке программирования C# «Консольное приложение (.NET Core)». После того, как создали проект в правой части экрана, существует компонент интерфейса «обозреватель решений», показано на рисунке 12.

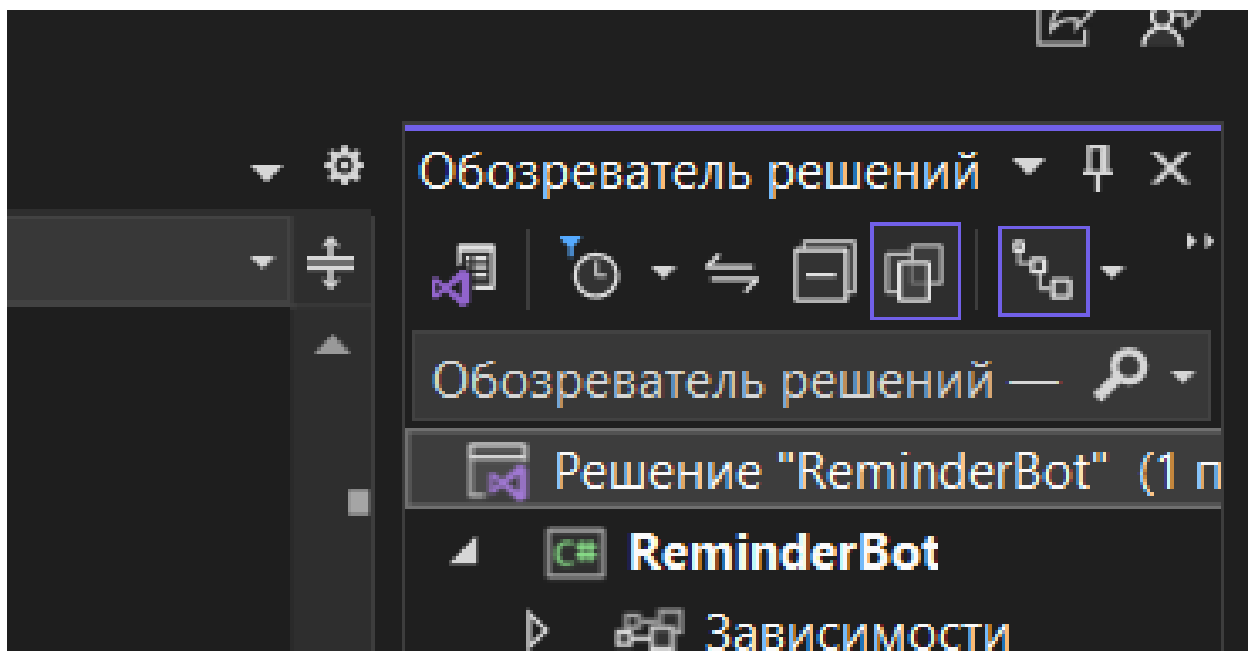


Рисунок 12 – Обзорщик решений

Кликаем правой кнопкой по нему и выбираем пункт «Управление пакетами NuGet для решения...», изображено на рисунке 13.

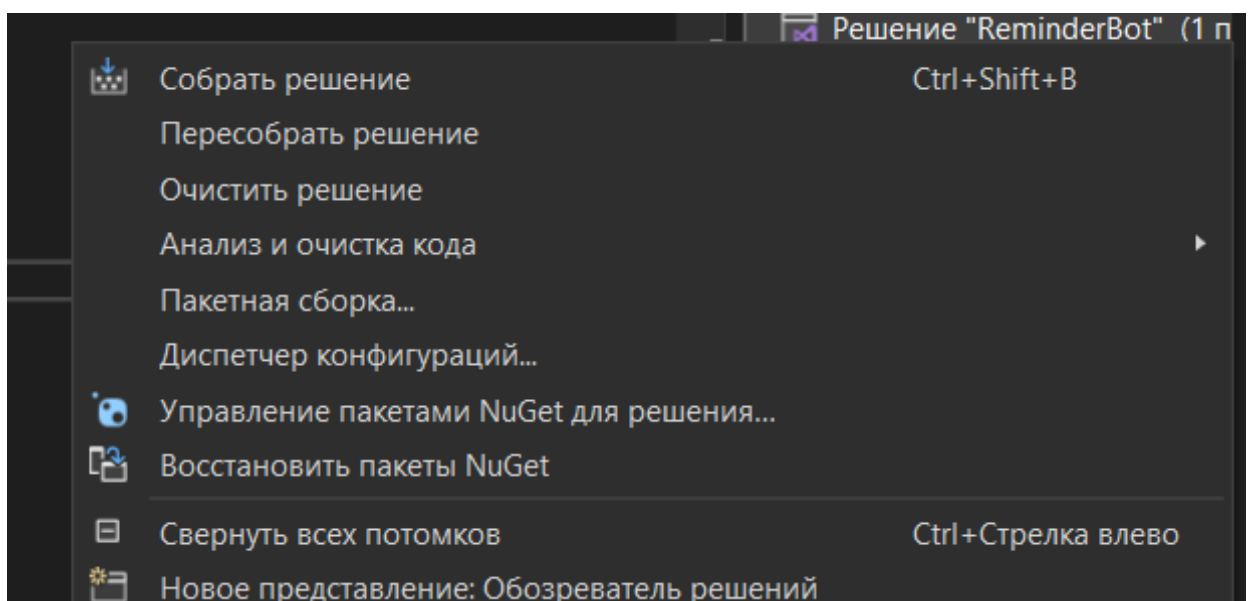


Рисунок 13 – Меню инструментов решений проекта

Перед нами расстилается интерфейс «NuGet – решения», с помощью которого возможно установить или дооснастить различные библиотеки,

настройки API и другие компоненты, необходимые для телеграм-бота, показано на рисунке 14.

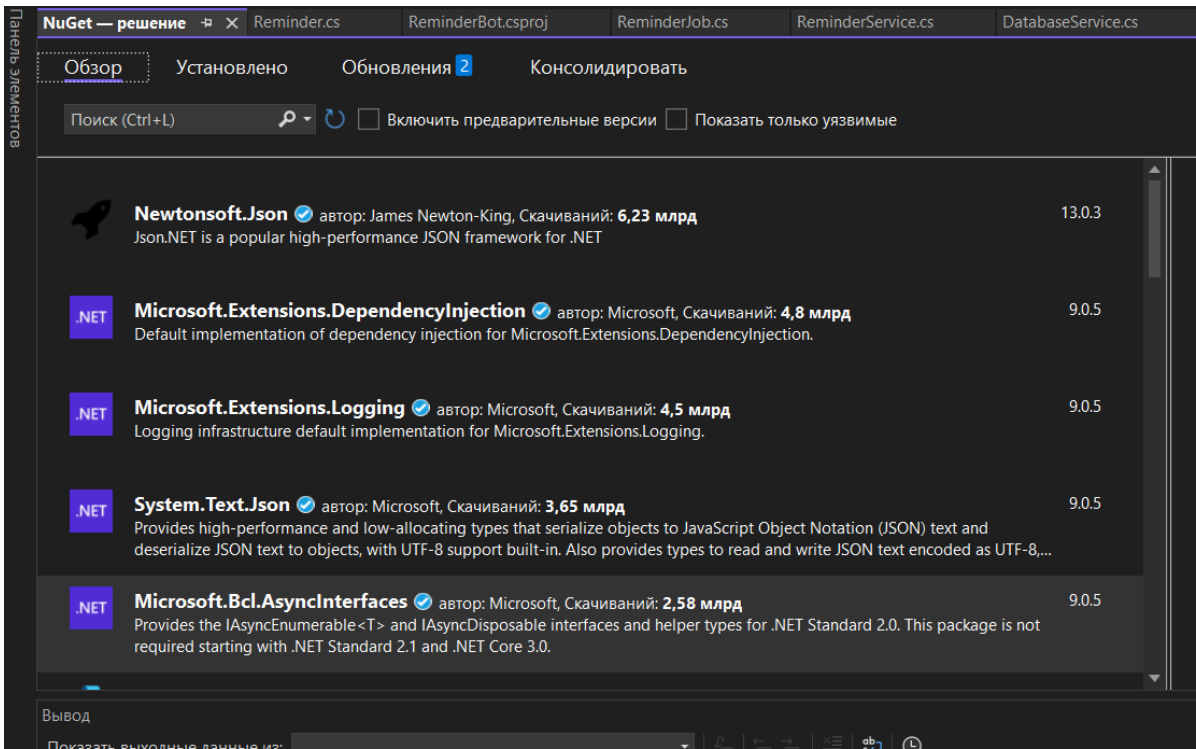


Рисунок 14 – Интерфейс «NuGet – решения»

В поисковой строке вбиваем «Telegram.Bot» и устанавливаем в первую очередь данную библиотеку, можно увидеть на рисунке 15.

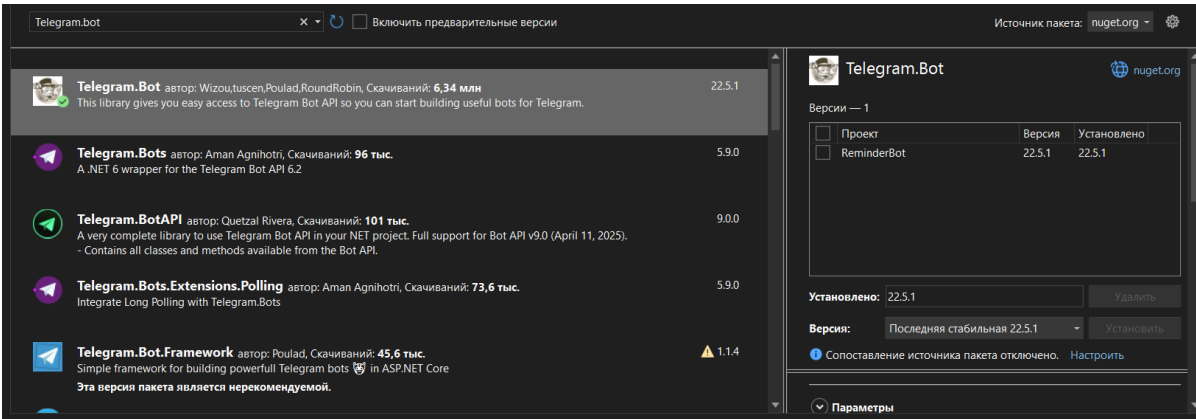


Рисунок 15 – Установка Telegram.Bot – библиотеки

Telegram.Bot — это официальная .NET-библиотека для работы с Telegram Bot API.

Она предоставляет:

- полный доступ к Bot API;
- асинхронную модель (поддержка `async/await`);
- встроенные механизмы для парсинга входящих сообщений и работы с клавиатурами;
- поддержку .NET 6+/7+/8+ (кроссплатформенность: Windows, Linux, macOS).

Существуют также и аналоги данной библиотеки, например, такие как:

- Telegram.Bot.Extensions [8] – это дополнение к основной библиотеке, для упрощения работы с Long Polling;
- WTelegramClient [9] – библиотека для работы с MTProto API, а значит не только для ботов, а для кастомных клиентов Telegram. Из-за чего появляются минусы, к примеру: сложность настройки и нестабильность.

В связи с чем, выбор пал на «Telegram.Bot», так как данная библиотека поддерживается сообществом разработчиков Telegram и регулярно обновляется, и подходит как для простых ботов (уведомления), так и для сложных (бот-систем с оплатой, голосовым вводом и т.д.).

Далее для локального хранения данных бота (напоминаний, пользовательских настроек), потребуется установка базы данных, а точнее Microsoft.Data.Sqlite, так как это легковесная технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных» для работы с SQLite в .NET-приложениях.

Ключевыми параметрами, исходя из чего выбор пал именно на эту библиотеку состоят в том, что:

- поддерживаются `async` или `await` режимы для всех операций;
- кроссплатформенная для всех операционных систем;
- интегрируется с Entity Framework Core;
- поддерживаются транзакции и миграции.

Данная библиотека представлена на рисунке 16.

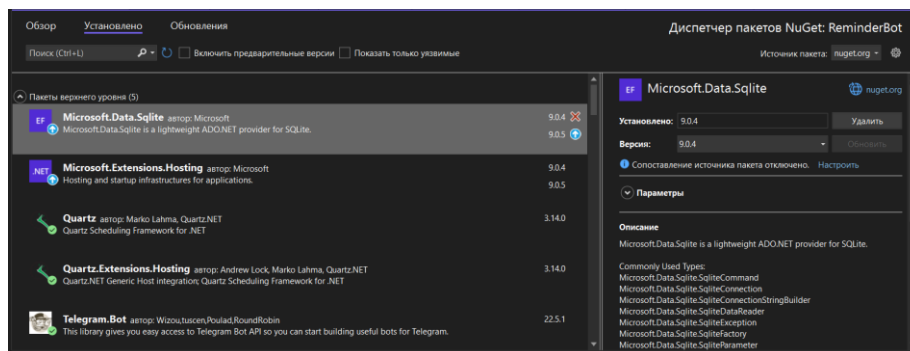


Рисунок 16 – База данных Microsoft.Data.Sqlite

Следующим шагом была произведена установка библиотеки Microsoft.Extensions.Hosting для создания управляемых хостов (hosted services) в .NET.

Главные возможности включают в себя:

- главный хост приложения (IHost) — управляет жизненным циклом;
- фоновые сервисы (IHostedService) — для долгоживущих задач (например, обработка сообщений бота);
- встроенная DI (внедрение зависимостей) — не нужно подключать отдельные контейнеры;
- конфигурация — поддержка appsettings.json, переменных окружения.

А также используется для фоновых задач, в нашем случае для Quartz-планировщика и для консольных приложений с долгим сроком жизни (боты, демоны). На эту библиотеку можно посмотреть на рисунке 17.

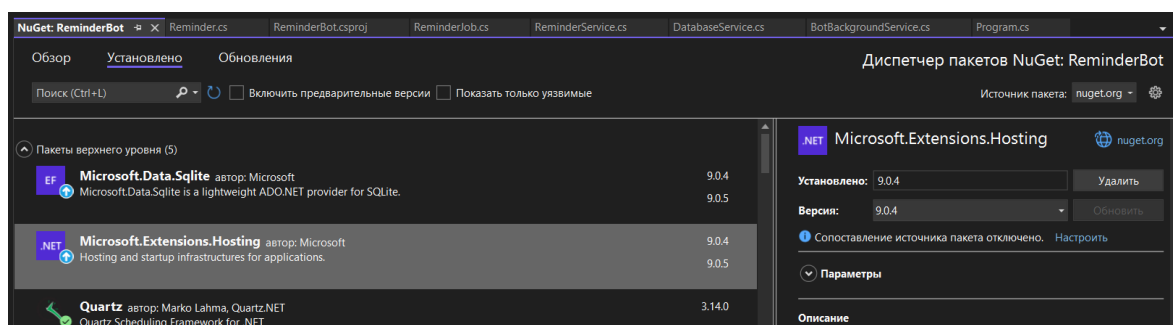


Рисунок 17 – Библиотека Microsoft.Extensions.Hosting

Была установлена вышеупомянутая библиотека для планирования задач – Quartz с поддержкой cron-выражений и сложных триггеров, показано на рисунке 18.

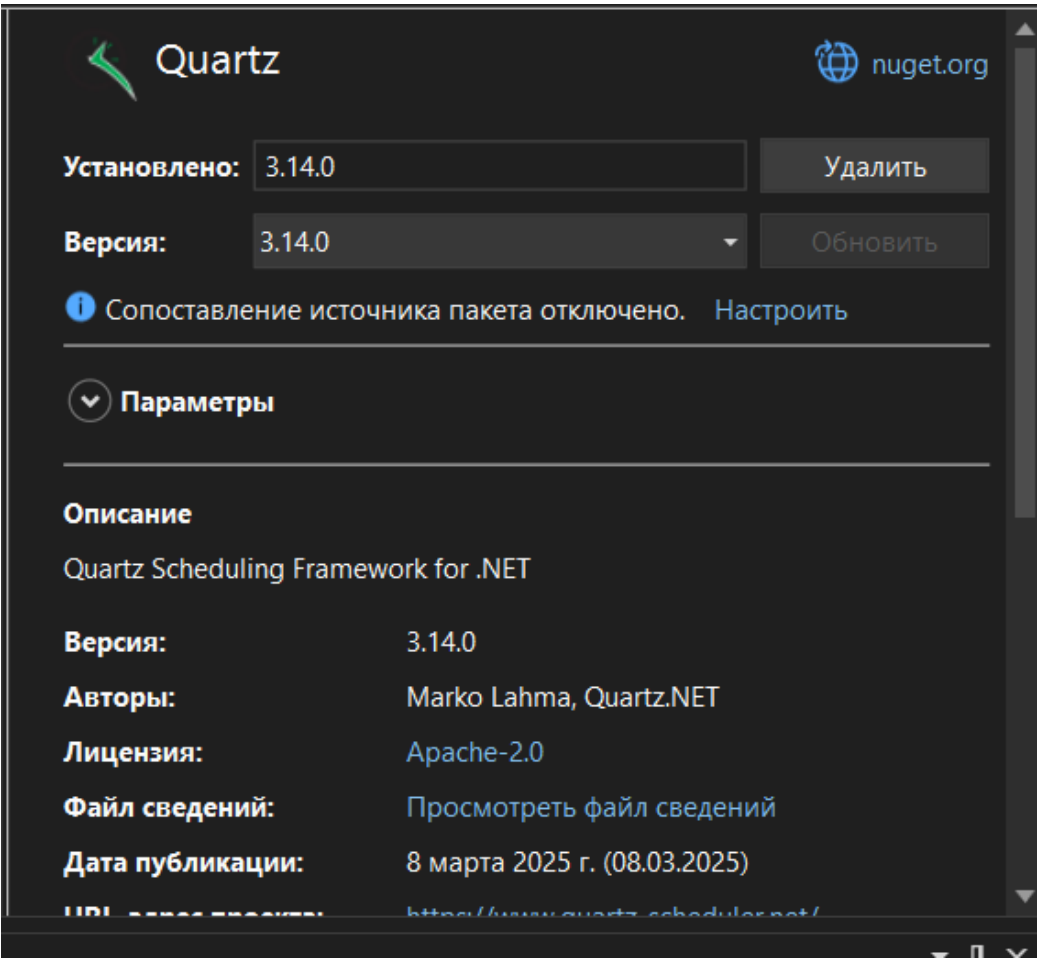


Рисунок 18 – Библиотека Quartz - планировщик

Библиотека Quartz представляет собой набор инструментов для отправки напоминаний в заданное время, а также если нужны повторяющиеся задачи (например, ежедневные уведомления), отлично поможет для реализации с ними.

Ключевые возможности:

- гибкое расписание — cron, интервалы, одноразовые задачи;
- кластеры — распределенное выполнение;
- асинхронность — все методы поддерживают async/await.

Для интеграции Quartz с Microsoft.Extensions.Hosting было установлено расширение Quartz.Extensions.Hosting, что позволит автоматический запуск планировщика при старте хоста и управление жизненным циклом, а точнее корректное завершение задач при остановке приложения.

Установка происходит тем же способом, через диспетчера пакетов NuGet, показано на рисунке 19.

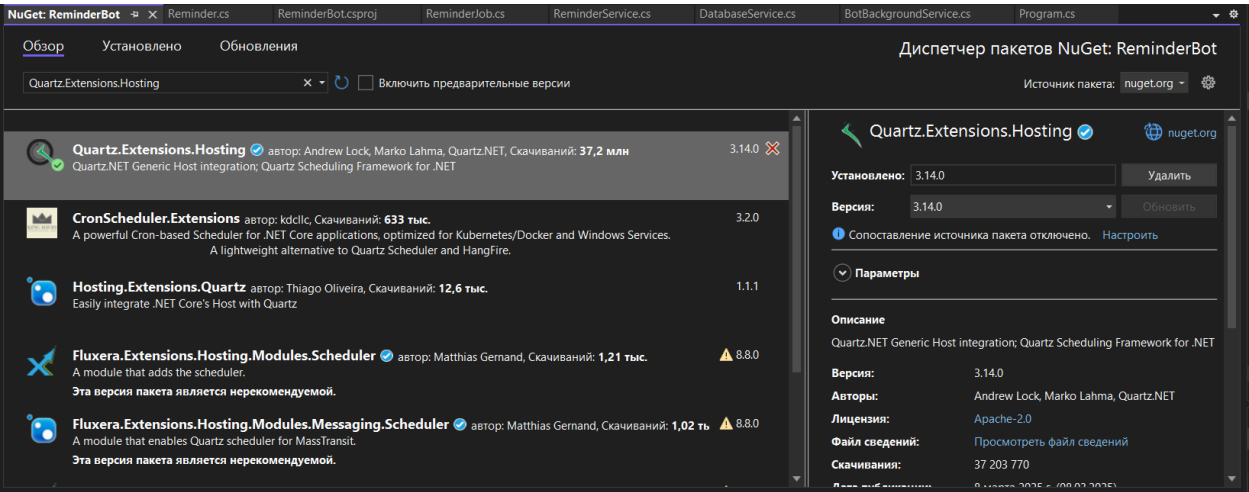


Рисунок 19 – Установка расширения Quartz.Extensions.Hosting

2 Проектирование и разработка

Инициализация системы происходит при запуске приложения. Сначала создается хост, который настраивает и запускает все необходимые сервисы. Основу составляют несколько ключевых компонентов: клиент для работы с Telegram API, сервис для взаимодействия с базой данных, сервис управления напоминаниями и планировщик задач Quartz. Также активируется фоновый сервис для обработки сообщений от пользователей.

Листинг 1 – Создание хоста

```
var builder = Host.CreateDefaultBuilder(args);
```

Эта строка создает стандартный хост для .NET приложения. `Host.CreateDefaultBuilder()` – метод, который настраивает хост со стандартными параметрами, включая: «Конфигурацию из `appsettings.json`», «Логирование», «Внедрение зависимостей».

Листинг 2 – Метод «ConfigureServices»

```
builder.ConfigureServices((context, services) =>
{
    //Регистрация сервисов
});
```

Метод `ConfigureServices` используется для регистрации всех сервисов, которые будут доступны через систему внедрения зависимостей.

Листинг 3 – Создание экземпляра сервиса

```
services.AddSingleton<ITelegramBotClient>(_ => new
TelegramBotClient("TOKEN"));
```

`AddSingleton` создает единственный экземпляр сервиса на все время работы приложения. `ITelegramBotClient` – интерфейс клиента Telegram Bot API. `new TelegramBotClient("TOKEN")` – создает экземпляр клиента с указанным токеном бота.

					ПГУ1.090301.48.001 ПЗ	Лист
						28
Изм.	Лист	№ докум.	Подпись	Дата		

Листинг 4 – Регистрация сервиса для базы данных

```
services.AddSingleton<DatabaseService>();
```

Регистрирует сервис для работы с базой данных как singleton, чтобы обеспечить единое подключение к БД на все время работы приложения.

Листинг 5 – Регистрация «ReminderService»

```
services.AddSingleton<ReminderService>();
```

Основной сервис для работы с напоминаниями также регистрируется как singleton.

Листинг 6 – Конфигурация планировщика

```
services.AddQuartz(q =>
{
    q.UseMicrosoftDependencyInjectionJobFactory();
    q.UseSimpleTypeLoader();
    q.UseInMemoryStore();
});
```

Конфигурирует планировщик задач Quartz.NET:

- UseMicrosoftDependencyInjectionJobFactory - позволяет использовать DI для jobs;
- UseSimpleTypeLoader - простой загрузчик типов;
- UseInMemoryStore - хранение задач в памяти.

Листинг 7 – Регистрация фонового сервиса

```
services.AddQuartzHostedService(q => q.WaitForJobsToComplete = true);
```

Регистрирует Quartz как фоновый сервис с настройкой: «WaitForJobsToComplete = true», что значит – ожидать завершения всех задач при остановке.

Листинг 8 – Регистрация основного сервиса

```
services.AddHostedService<BotBackgroundService>();
```

Регистрирует основной фоновый сервис бота, который будет: «Обрабатывать входящие сообщения», «Управлять жизненным циклом бота», «Координировать работу других сервисов».

2.1 Основные компоненты

2.1.1 BotBackgroundService

Первый компонент – BotBackgroundService. Это фоновый сервис, отвечающий за основную работу бота. Он запускает процесс получения обновлений от Telegram API, настраивает обработчики входящих сообщений и возможных ошибок. При успешном запуске он выводит информацию о готовности бота к работе. Сервис обрабатывает основные команды пользователей, такие как старт и меню, отображает главное меню с кнопками интерфейса и перенаправляет запросы в соответствующие сервисы для дальнейшей обработки.

Листинг 9 – метод «ExecuteAsync»

```
protected override async Task ExecuteAsync(CancellationToken  
stoppingToken)  
{  
    /.../  
}
```

Метод ExecuteAsync – основной рабочий цикл фонового сервиса бота. Это переопределенный метод из базового класса BackgroundService, который выполняется при запуске хоста. Он отвечает за:

- инициализацию системы получения updates от Telegram;
- настройку обработчиков входящих сообщений и ошибок;
- запуск фонового процесса прослушивания;
- контроль жизненного цикла сервиса.

CancellationToken stoppingToken – токен отмены, который используется для корректного завершения работы сервиса при остановке приложения.

Листинг 10 – Создание объекта конфигурации

```
var receiverOptions = new ReceiverOptions
{
    AllowedUpdates = Array.Empty<UpdateType>()
};
```

Создает объект конфигурации для получения updates:

– AllowedUpdates = Array.Empty<UpdateType>() означает, что бот будет получать все типы updates.

Листинг 11 – Настройка фонового процесса «updateHandler»

```
_botClient.StartReceiving(
    updateHandler: HandleUpdateAsync,
    errorHandler: HandlePollingErrorAsync,
    receiverOptions: receiverOptions,
    cancellationToken: stoppingToken);
```

Настраивает фоновый процесс получения сообщений: «updateHandler: HandleUpdateAsync» – метод, который будет вызываться для каждого входящего update; «errorHandler: HandlePollingErrorAsync» – метод для обработки ошибок при получении updates; «receiverOptions» – передает настроенные параметры получения; «cancellationToken» – токен для остановки получения.

Листинг 12 – Асинхронный запрос

```
var me = await _botClient.GetMe(stoppingToken);
Console.WriteLine($"Бот @{me.Username} запущен и готов к работе!");
```

В данных строчках кода производится асинхронный запрос к Telegram API для получения информации о боте, где GetMe возвращает объект с username, id и другими данными бота, и выводит в консоль сообщение о успешном запуске

2.1.2 ReminderService

Второй важный компонент – «ReminderService». Это центральный сервис для работы с напоминаниями. Он выполняет добавление новых напоминаний с автоматическим определением даты и времени из текста, предоставляет функции управления списком напоминаний, включая их просмотр, удаление и редактирование. Сервис интегрирован с планировщиком Quartz для установки времени срабатывания уведомлений. Он взаимодействует с базой данных для сохранения и получения информации, проверяет корректность вводимых пользователем данных и формирует ответы о результатах выполнения операций.

Листинг 13 – Метод «AddReminderAsync»

```
public async Task AddReminderAsync(long chatId, string
messageText)
{
    /.../
}
```

Метод AddReminderAsync отвечает за добавление нового напоминания. Это асинхронный метод, который принимает:

- long chatId – идентификатор чата с пользователем;
- string messageText – текст сообщения с напоминанием и датой.

Листинг 14 – Парсинг даты и времени

```
if (!TryParseRussianDateTime(dateString, out var remindAt))
{
    await _botClient.SendMessage(chatId, "✗ Неверная дата.
Формат: 25 декабря 18:30");
    return;
}
```

Данный блок выполняет парсинг даты и времени из текста сообщения:

- TryParseRussianDateTime – метод для преобразования строки с датой в русском формате в объект DateTime;

- если парсинг не удался, отправляется сообщение об ошибке с примером правильного формата;
- метод завершает выполнение при неудачном парсинге.

Листинг 15 – Создание задачи для планировщика

```
var job = JobBuilder.Create<ReminderJob>()
    .WithIdentity(jobKey)
    .UsingJobData("chatId", chatId)
    .UsingJobData("text", text)
    .Build();
```

Создается задача для планировщика Quartz:

- JobBuilder.Create<ReminderJob>() – создает задачу типа ReminderJob;
- WithIdentity(jobKey) – устанавливает уникальный идентификатор задачи;
- UsingJobData – передает параметры в задачу (chatId и текст напоминания);
- Build() – завершает конфигурацию и создает объект задачи.

Листинг 16 – Сохранение напоминания в базу данных

```
await _db.AddReminderAsync(chatId, text, remindAt, jobKey);
```

Производится сохранение напоминания в базу данных:

- _db.AddReminderAsync – асинхронный метод сервиса работы с базой данных;
- передаются параметры: chatId, текст напоминания, время срабатывания и ключ задачи Quartz;
- ожидается завершение операции сохранения (await).

2.1.3 DatabaseService

Третий компонент – DatabaseService. Он отвечает за все операции с базой данных SQLite. В его функции входит создание и инициализация базы данных, формирование необходимых таблиц, выполнение всех основных операций с данными - добавление, чтение, обновление и удаление

информации о напоминаниях. Сервис поддерживает асинхронные операции для эффективной работы, хранит полные данные о напоминаниях, включая идентификаторы чатов, текст сообщений, время срабатывания и служебные ключи, а также обеспечивает надежность хранения информации и обработку возможных ошибок.

Листинг 17 – Метод «AddReminderAsync»

```
public async Task AddReminderAsync(long chatId, string text,
DateTime remindAt, string jobKey)
{
    /.../**
}
```

Метод `AddReminderAsync` выполняет асинхронное сохранение напоминания в базу данных. Это ключевой метод для персистентного хранения данных о напоминаниях.

Параметры метода:

- `long chatId`: уникальный идентификатор чата пользователя в Telegram;
- `string text`: текст напоминания, которое нужно сохранить;
- `DateTime remindAt`: дата и время срабатывания напоминания;
- `string jobKey`: уникальный идентификатор связанной задачи в планировщике Quartz.

Листинг 18 – Создание SQL-запроса

```
cmd.CommandText = @"
INSERT INTO Reminders (ChatId, Text, RemindAt, JobKey)
VALUES (@chatId, @text, @remindAt, @jobKey)";
```

Создает параметризованный SQL-запрос для:

- вставки новой записи в таблицу `Reminders`;
- указания всех необходимых полей (`ChatId`, `Text`, `RemindAt`, `JobKey`);
- использования параметров вместо прямых значений для безопасности.

Листинг 19 – Настройка параметров запроса

					ПГУ1.090301.48.001 ПЗ	Лист
						34
Изм.	Лист	№ докум.	Подпись	Дата		

```
cmd.Parameters.AddWithValue("@chatId", chatId);
cmd.Parameters.AddWithValue("@text", text);
cmd.Parameters.AddWithValue("@remindAt", remindAt.ToString("O"));
cmd.Parameters.AddWithValue("@jobKey", jobKey);
```

Настраивает параметры запроса:

- «chatId»: привязывает идентификатор чата;
- «text»: привязывает текст напоминания;
- «remindAt»: преобразует дату в строку формата ISO8601;
- «jobKey»: привязывает ключ задачи Quartz.

Листинг 20 – Выполнение асинхронной SQL-команды

```
await cmd.ExecuteNonQueryAsync();
```

Асинхронно выполняет SQL-команду: «ExecuteNonQueryAsync» используется для операций INSERT/UPDATE/DELETE; ключевое слово `await` обеспечивает неблокирующее выполнение; возвращает `Task` для ожидания завершения операции.

2.1.4 ReminderJob

Четвертый компонент – `ReminderJob`. Этот класс непосредственно занимается отправкой запланированных уведомлений. Он получает данные о напоминании из системы планирования, отправляет сообщение пользователю в указанное время через Telegram API, обрабатывает возможные ошибки при отправке. Класс интегрирован с основным ботом через передачу клиента Telegram в системе планирования и гарантирует доставку напоминаний пользователям.

`IJobExecutionContext context` – контекст выполнения задачи, предоставляемый Quartz.NET. Содержит:

- данные задачи (`JobDataMap`);
- ссылку на планировщик;
- информацию о времени выполнения;
- возможность получить дополнительные сервисы.

Листинг 21 – Присвоение уникального идентификатора

```
var chatId = data.GetLong("chatId");  
var text = data.GetString("text");
```

- chatId – уникальный идентификатор чата Telegram, куда нужно отправить напоминание;
- text - сохраненный текст напоминания, который ввел пользователь;
- данные извлекаются из JobDataMap.

Листинг 22 – Извлечение клиента Telegram из контекста планировщика

```
var botClient =  
(ITelegramBotClient) context.Scheduler.Context.Get("botClient");
```

- клиент Telegram извлекается из контекста планировщика;
- это тот же экземпляр, что используется во всем приложении;
- позволяет избежать создания нового подключения.

Листинг 23 – Асинхронная отправка сообщения

```
await botClient.SendMessage(  
    chatId: chatId,  
    text: $"🔔 Напоминание: {text}");
```

- асинхронная отправка сообщения через Telegram Bot API
- форматированный текст с эмодзи для лучшей визуализации
- указание конкретного чата через chatId

Все эти компоненты работают вместе, обеспечивая полный цикл работы с напоминаниями – от момента их создания пользователем до своевременной отправки уведомлений. Система построена по модульному принципу, где каждый компонент выполняет строго определенные функции, что облегчает поддержку и развитие бота.

2.2 Тестирование

Для проведения тестирования телеграм-бота «Напоминания» были произведены следующие действия:

					ПГУ1.090301.48.001 ПЗ	Лист
						36
Изм.	Лист	№ докум.	Подпись	Дата		

- функциональное тестирование;
- тестирование пользовательского интерфейса;
- тестирование обработки ошибок;
- интеграционное тестирование взаимодействия компонентов.

А также была поставлена цель тестирования, чтобы проверить корректность работы основных функций системы, включая: создание напоминаний с указанием текста и даты; хранение и отображение списка активных напоминаний; удаление напоминаний; отправку уведомлений в указанное время; работу пользовательского интерфейса.

В качестве сервера выступает персональный компьютер на операционной системе Windows. Запустив программу, в частности серверную часть на платформе Visual Studio 2022, открылось окно командной строки. В данном окне можно увидеть запущенные технологии для создания задач, а точнее библиотека Quartz. Также для открытия хоста, был запущен Microsoft.Hosting. При успешном запуске сервера, мы наблюдаем уведомление, что бот запущен и готов к работе, что показано на рисунке 20.

```

C:\Users\Ростислав\source\repos\ReminderBot\ReminderBot\bin\Debug\net8.0\ReminderBot.exe
info: Quartz.Core.SchedulerSignalerImpl[0]
      Initialized Signaller of type: Quartz.Core.SchedulerSignalerImpl
info: Quartz.Core.QuartzScheduler[0]
      Quartz Scheduler created
info: Quartz.Core.QuartzScheduler[0]
      JobFactory set to: Quartz.Simpl.MicrosoftDependencyInjectionJobFactory
info: Quartz.Simpl.RAMJobStore[0]
      RAMJobStore initialized.
info: Quartz.Impl.StdSchedulerFactory[0]
      Quartz Scheduler 3.14.0.0 - 'QuartzScheduler' with instanceId 'NON_CLUSTERED' initialized
info: Quartz.Impl.StdSchedulerFactory[0]
      Using thread pool 'Quartz.Simpl.DefaultThreadPool', size: 10
info: Quartz.Impl.StdSchedulerFactory[0]
      Using job store 'Quartz.Simpl.RAMJobStore', supports persistence: False, clustered: False
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Quartz.Core.QuartzScheduler[0]
      Scheduler QuartzScheduler_$_NON_CLUSTERED started.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\Ростислав\source\repos\ReminderBot\ReminderBot\bin\Debug\net8.0
Бот @botzaromni_Bot запущен и готов к работе!
  
```

Рисунок 20 – Запуск серверной части телеграм-бота

После успешного запуска сервера, откроем приложение Telegram и воспользуемся непосредственно самим ботом. Тестирование производилось на мобильном устройстве под операционной системой IOS 18.4.1, с актуальной версией Telegram 11.11.

Перед нами расстилается минималистичный интерфейс с находящейся внизу кнопкой «Start», сигнализирующая о том, что мы можем начать работу с данным ботом, изображено на рисунке 21.



Рисунок 21 – Начальный интерфейс телеграм-бота.

Запускаем бота, в этом нам поможет вышеупомянутая кнопка «Start», и видим, как нас встречает экран приветствия, на котором описаны возможности бота, а также если обратить внимание на нижнюю панель, то можно увидеть появление новых кнопок, для взаимодействия с интерфейсом. Ведь в современных реалиях важно, чтобы интерфейс программы и различных сервисов был прост в эксплуатации, дабы сэкономить время и не писать команды целиком. Для этого в боте были добавлены кнопки: «Добавить напоминание», «Мои напоминания» и «Помощь», что показывает рисунок 22.

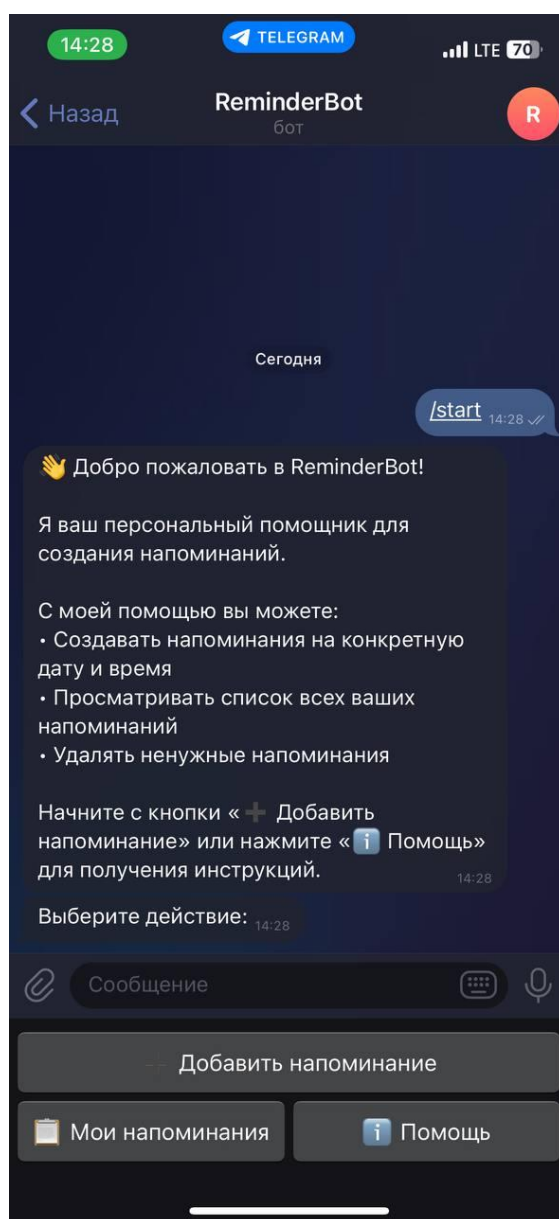


Рисунок 22 – Экран приветствия

После ознакомления с начальным интерфейсом телеграм-бота, воспользуемся кнопкой «Добавить напоминание», и выберем режим напоминания.

Нажав на кнопку, можно увидеть сообщение от бота, которое показывает интерфейс с inline-кнопками, где можно выбрать какой тип напоминания нас интересует. Данные возможности показаны на рисунке 23.

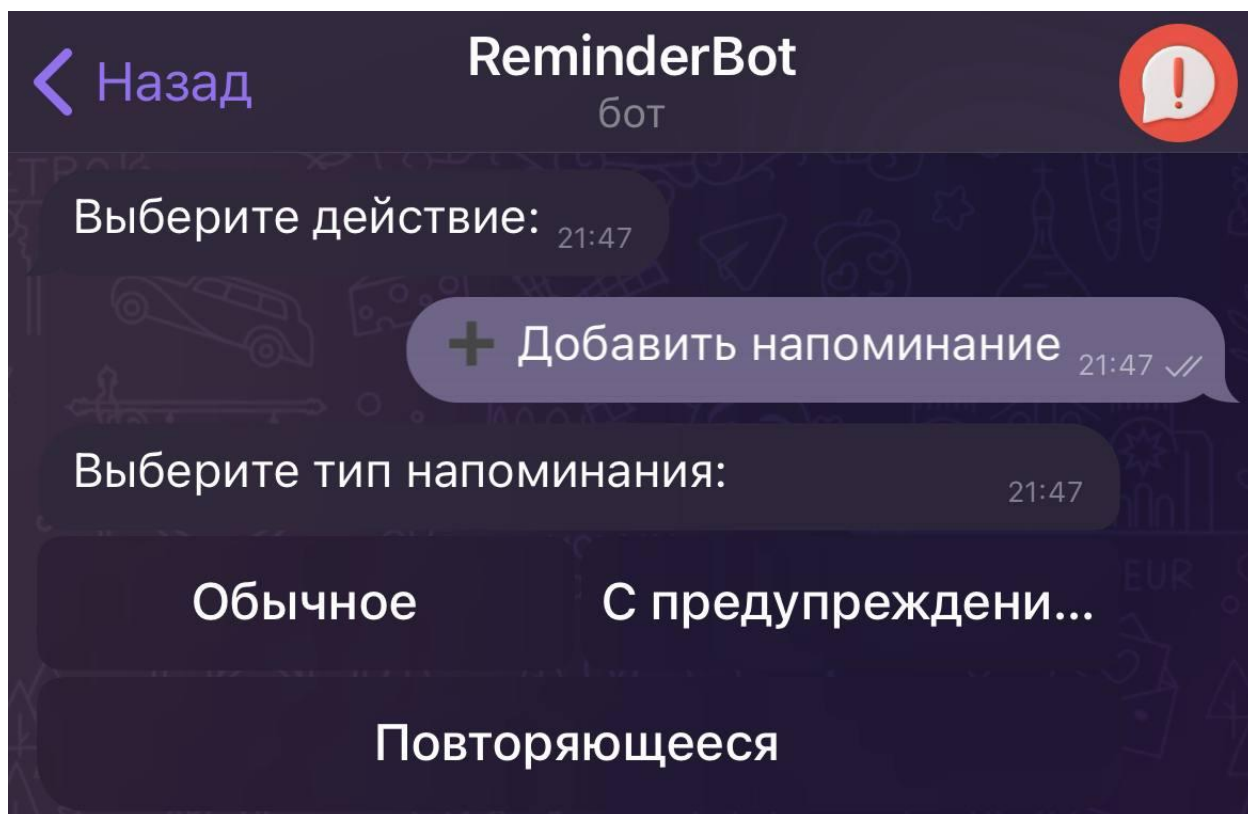


Рисунок 23 – Ознакомление с интерфейсом добавления напоминаний

Выберем создание обычных напоминаний и увидим пример того, какой формат записи он поддерживает, что поможет пользователю безошибочно создать новое напоминание. А также если посмотреть на нижнюю панель, то можно увидеть новую кнопку в интерфейсе «Назад», что поможет откатить действия и вернуться на главное меню.

После ознакомление с примером, введем заметку для записи напоминаний, а также дату и время в том формате, который описал бот. Нажмем кнопку отправить, для отправки сообщения и увидим следующее

сообщение от бота: «Напоминание добавлено», что свидетельствует о том, что запись произведена успешно, а также описано, что именно добавлено в него и на какое время ждать уведомление, изображено на рисунке 24.

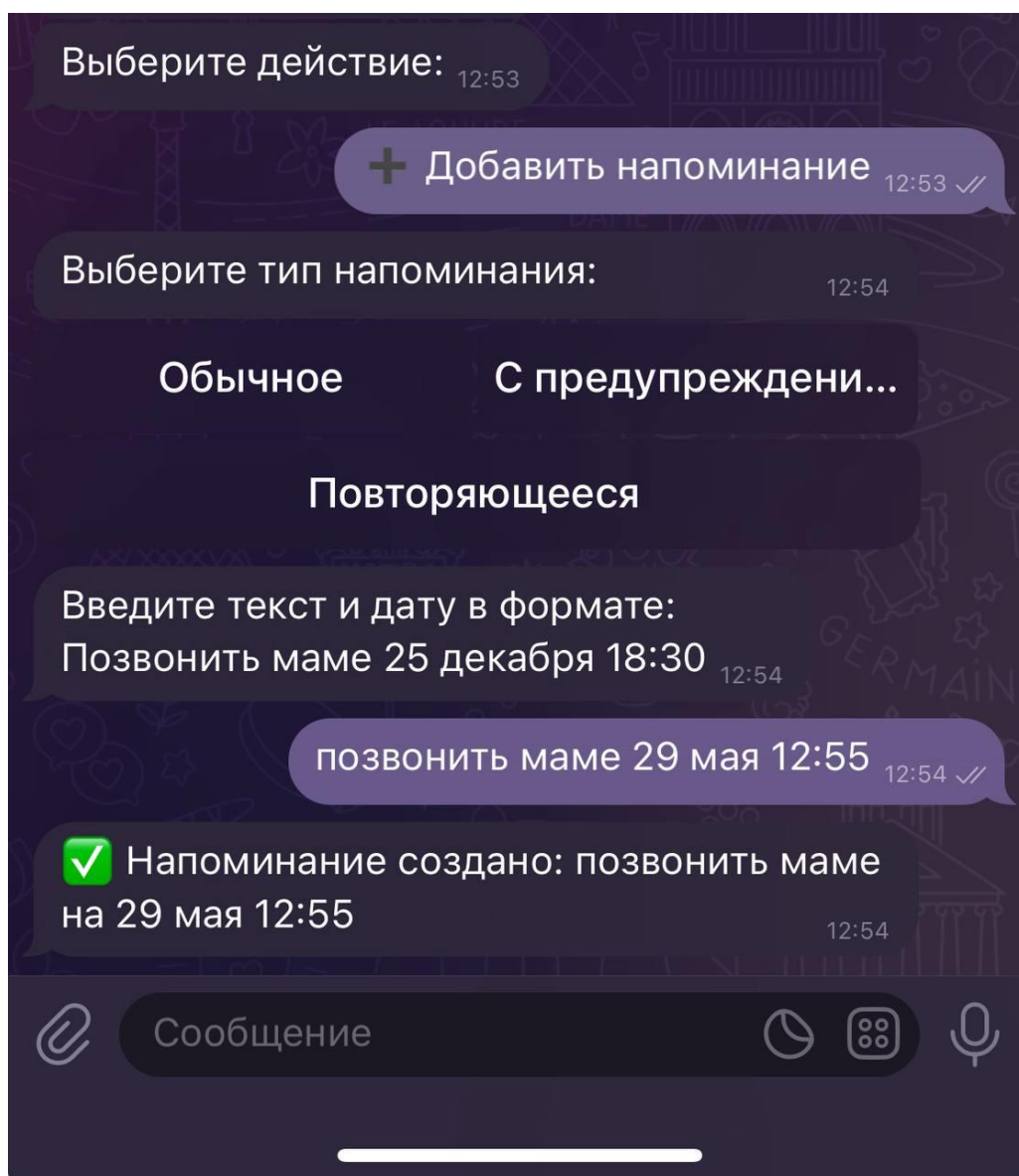


Рисунок 24 – Создание напоминание в телеграм-боте

После записи напоминания, можно просмотреть список всех созданных напоминаний, кликнув по кнопке «Мои напоминания». Бот отправит сообщение в диалог, где покажет все записи, хранящиеся в базе данных, которые еще не были завершены. И предложит удалить напоминание, кнопкой, которая показана на нижней панели. Если же данное действие не

соответствует требованиям пользователя, можно нажать на кнопку «Назад» и вернуться на главное меню. Увидеть вышеперечисленное можно на рисунке 25.

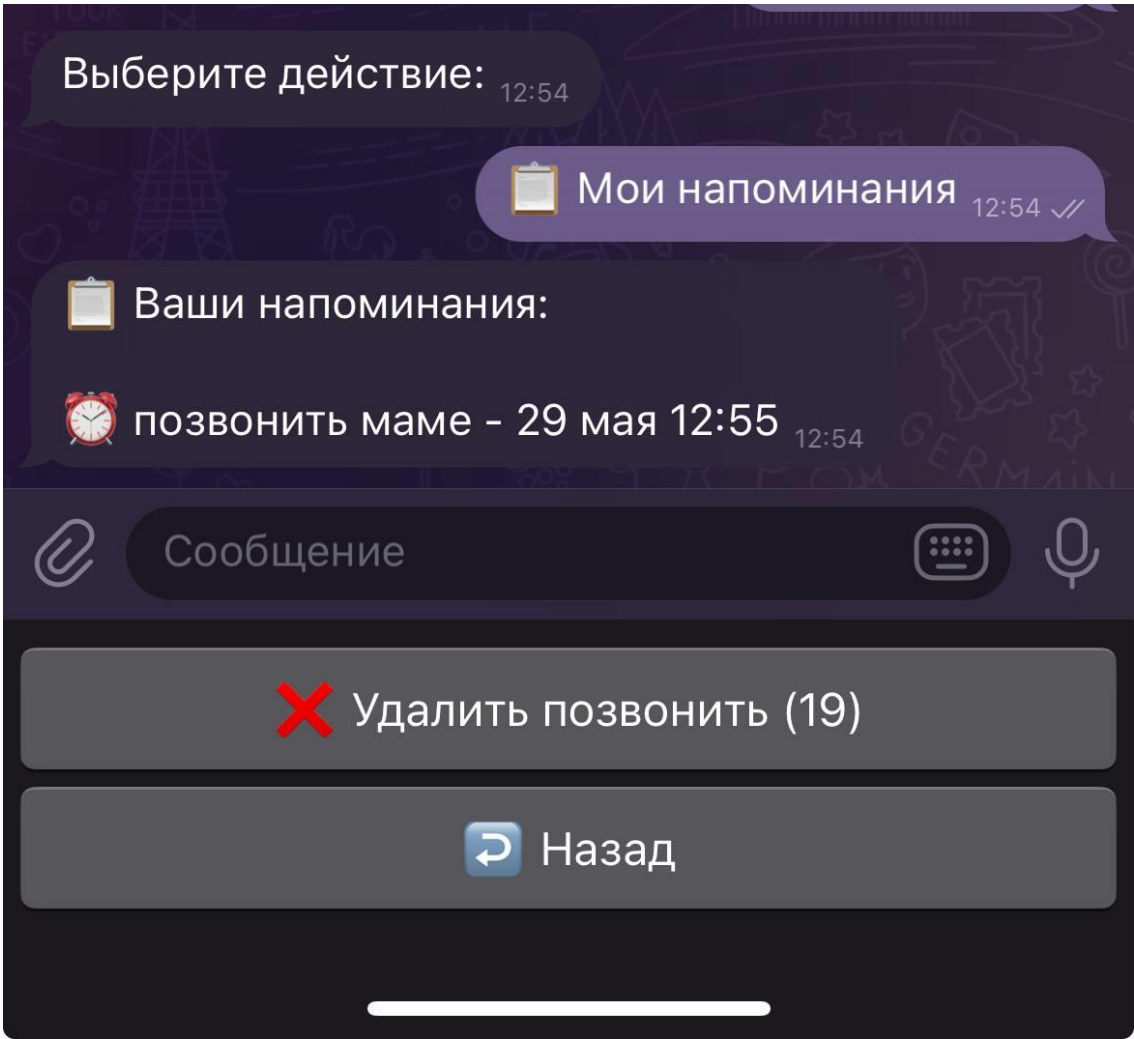


Рисунок 25 – Список напоминаний

В конечном итоге, после того, когда наступит дата и время, та, что была указана в напоминании – поступит уведомление от бота, с тем текстом, который был записан в сообщении и отправлено ему. Данное уведомление можно увидеть в диалоге с ботом или на экране уведомлений мобильного устройства. После чего, оно удалится автоматически из списка напоминаний и базы данных, в которой оно хранилось. Данный фрагмент показан на рисунке 26.

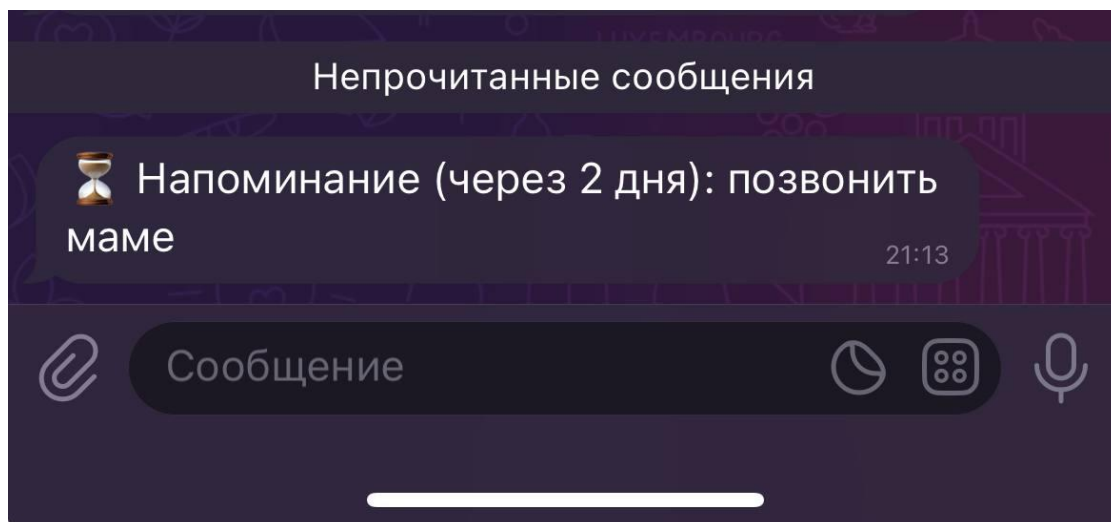


Рисунок 28 – Уведомление от бота, о напоминании с предупреждением

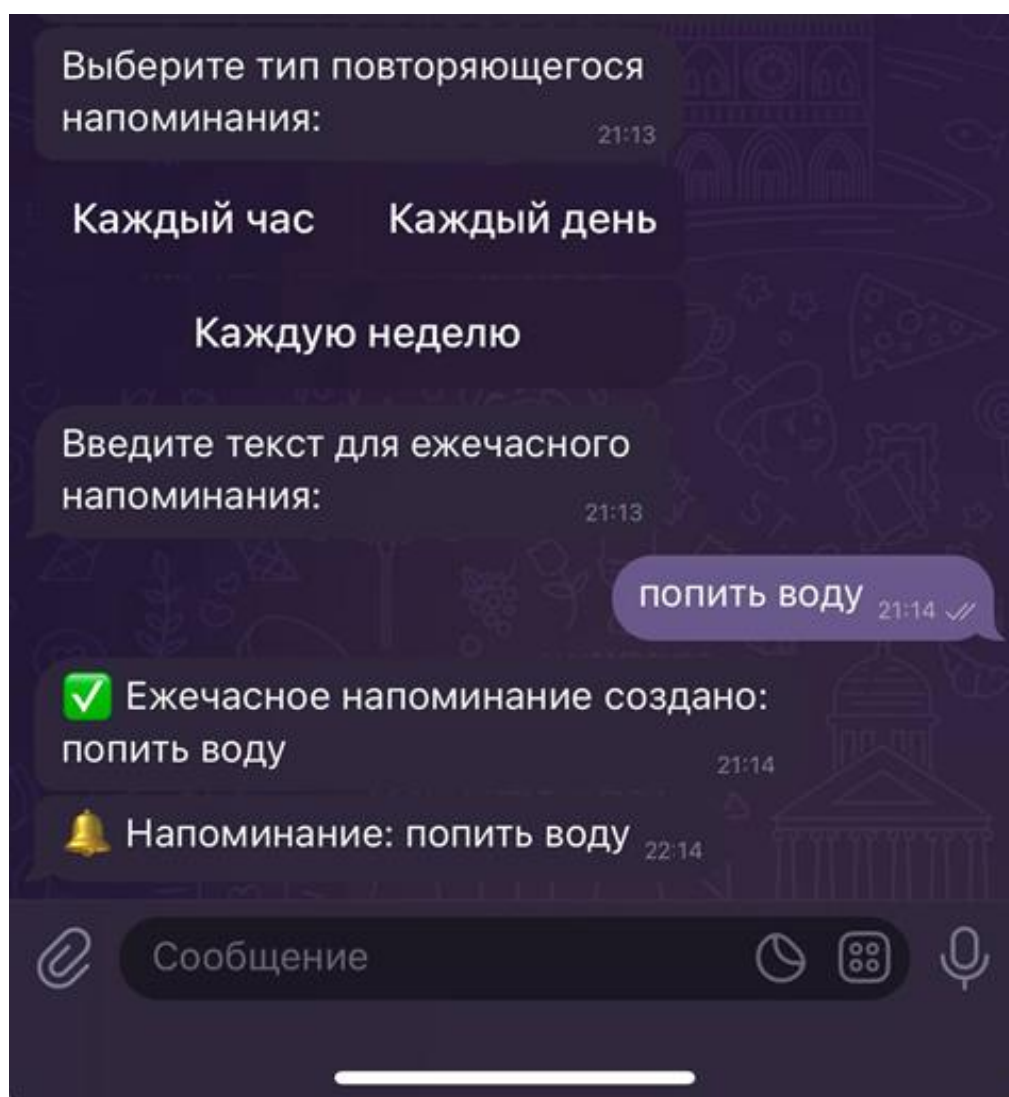


Рисунок 29 – Создание и результат повторяющегося напоминания

Вернувшись на главное меню телеграм-бота, воспользуемся кнопкой «Помощь», что откроет список для помощи с эксплуатацией бота. Подробная инструкция описывает такие вопросы как:

- добавление напоминаний;
- просмотр напоминаний;
- удаление напоминаний;
- навигация.

Данные возможности можно увидеть на рисунке 30 и изучить перед эксплуатацией телеграм-бота.

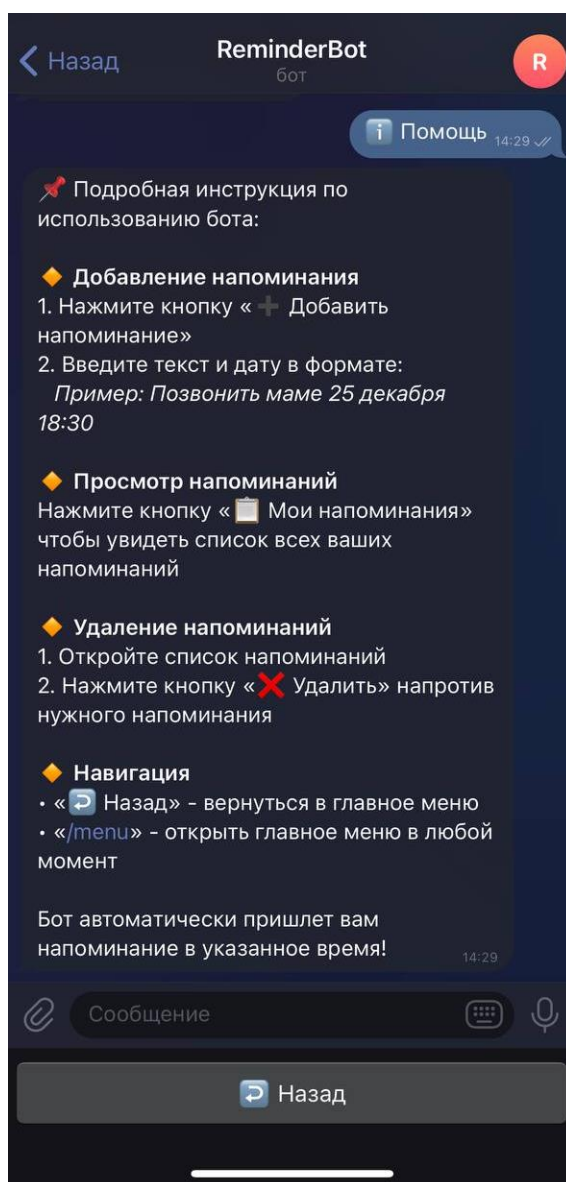


Рисунок 30 – Подробная инструкция по использованию бота

3 Безопасность жизнедеятельности

3.1 Правила безопасности при использовании ПК

Правила выполнения работ с ПК обязаны обеспечивать минимизацию основных рисков, перечисленных в ГОСТ 12.0.003-2015. В первую очередь к ним относятся:

- повышенная температура отдельных элементов персонального компьютера;
- монотонность трудового процесса;
- высокий уровень зрительных нагрузок;
- вероятность поражения статическим электричеством, генерируемым техникой;
- недостаточная освещенность в зоне выполнения операций у компьютера;
- высокий уровень напряженности электрического и магнитного полей в зоне установки компьютера;
- другие факторы.

Полностью избавиться от их действия, конечно, не получится, однако работодатель обязан стремиться к тому, чтобы сократить воздействие этих факторов на сотрудников, обеспечивая возможность восстановления нормального самочувствия и работоспособности уже к началу следующей рабочей смены без применения дополнительных средств, например, массажа или медикаментозной терапии.

3.2 Правила безопасности при использовании смартфона

Для того, чтобы предотвратить возможные поломки и повреждения смартфона и сопутствующего оборудования, и уменьшить потенциальные

					ПГУ1.090301.48.001 ПЗ	Лист
						46
Изм.	Лист	№ докум.	Подпись	Дата		

риски возгорания или удара током, необходимо серьезно отнестись к условиям, представленным ниже:

- запрещено самостоятельно открыть смартфон, зарядное устройство и инструмент;

- не сдавливайте ваш смартфон, зарядное устройство и инструмент и не допускайте возможного поражения, а также колющих и режущих повреждений;

- не закрывайте зону внешних коммуникаций и электроснабжения, держите смартфон подальше от открытого огня и горящего табака, избегайте попадания воды или огня на них;

- не используйте телефон и чашу рядом с водой или жидкостью, не прикасайтесь к ним мокрыми руками;

- если телефон или зарядное устройство намокли, во избежание повреждений не пытайтесь сушить их в духовке, микроволновой печи или феном;

- не роняйте, не бросайте и не сгибайте свой мобильный телефон и посуду;

- не используйте телефон и тарелку, если они неисправны, а также после повреждения или падения.

Расстояние между корпусом и телефоном должно составлять не менее 1 см.

Чтобы предотвратить риски (а также потому, что это является нарушением закона в большинстве стран), не используйте телефон во время вождения, во время работы с производственным инструментом, а также при выполнении других потенциально опасных действий.

Не используйте телефон и зарядное устройство при температуре окружающей среды ниже 0°C и выше 40°C. Оптимальный температурный диапазон для хранения: от 0°C до 40°C.

					ПГУ1.090301.48.001 ПЗ	Лист
						47
Изм.	Лист	№ докум.	Подпись	Дата		

Во время нормальной работы телефон и лоток могут нагреваться. Вскоре (в том числе косвенно) при контакте устройств с кожей – это может привести к неприятным ощущениям и может оставить ожог.

Телефоны могут передавать и принимать сигналы LTE, WCDMA, GSM, Wi-Fi и Bluetooth, а также могут содержать информацию об электромагнитных полях, которые могут мешать работе кардиостимуляторов и других имплантированных медицинских устройств. Владелец должен соблюдать следующие условия:

Не кладите мобильный телефон в карман (или другие вещи), который находится рядом с кардиостимулятором или другим имплантированным медицинским устройством.

Если у вас есть какие-либо вопросы о совместимости вашего телефона с таким устройством, проконсультируйтесь с врачом.

Если вы считаете, что мобильное устройство мешает работе обычного кардиостимулятора или другого медицинского устройства, выключите телефон и обратитесь к врачу.

Меры предосторожности при использовании зарядного устройства

При отсоединении пластины от розетки тяните непосредственно к дому, а не по кабелю.

Когда вы отвечаете на телефонный звонок, убедитесь, что к адаптеру питания легко получить доступ и он подключен к розетке рядом с телефоном.

Подключайте к телефону только те устройства, которые имеют логотип USB-IF или прошли тестирование на соответствие этому стандарту.

Чтобы заменить аккумулятор, обратитесь в сервисный центр. Использование батареи, не соответствующей стандарту, может привести к возгоранию, взрыву, утечке воды и другим опасным побочным эффектам.

Если из аккумулятора вытекает вода, не допускайте ее попадания в глаза, на кожу и одежду. Если жидкость попала вам в глаза, не трите их, а промойте чистой водой и немедленно обратитесь к врачу.

Держите протекающую батарею подальше от огня, чтобы избежать возгорания или взрыва.

Замена телефона и аккумулятора устройства должна производиться в соответствии с местным законодательством. Не выбрасывайте электронные устройства вместе с бытовыми отходами. Неправильная утилизация может привести к пожару, взрыву и другим опасным побочным эффектам.

Не используйте и не храните мобильный телефон и устройство в пыльных, прокуренных, влажных и загрязненных помещениях, а также в зоне воздействия магнитного поля – это может привести к повреждению или короткому замыканию внутри помещения.

Держите телефон подальше от источников тепла и не подвергайте его воздействию прямых солнечных лучей. Не оставляйте телефон в автомобиле и других местах, где температура может превышать 60 °C, например, на приборной панели автомобиля, на подоконнике, рядом с вентиляционными отверстиями системы отопления, а также за стеклом, которое подвергается длительному воздействию прямых солнечных лучей или сильное ультрафиолетовое излучение. Это приводит к поломке телефона, перегреву аккумулятора, а также может привести к взрыву или возгоранию.

Не используйте телефон и не храните его вблизи легковоспламеняющихся и взрывающихся вещей (например, на заправочных станциях, складах горючего или химических заводах). Не используйте телефон во время взрывных работ и в потенциально взрывоопасном месте, например, на нижней палубе судна, вблизи горючих и химических соединений в условиях хранения или транспортировки, а также в атмосфере присутствуют частицы химических веществ или пыли (в том числе риса и железа). В подобных случаях одна искра может привести к взрыву или пожару и, как следствие, к травмам и даже смерти.

					ПГУ1.090301.48.001 ПЗ	Лист
						49
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе работы был успешно проведен анализ существующих решений для управления напоминаниями и наглядно показано, что Telegram-боты обладают рядом преимуществ, таких как доступность, простота использования и интеграция с мессенджером. Однако они уступают специализированным сервисам, в функциональности, включая поддержку синхронизации с календарями. В следствии чего был разработан и протестирован свой телеграм-бот для управления напоминаниями. Бот демонстрирует высокую эффективность в выполнении своих основных функций: создание, хранение и своевременная отправка напоминаний. Модульная архитектура системы, включающая такие компоненты, как BotBackgroundService, ReminderService, DatabaseService и ReminderJob, обеспечивает гибкость и легкость поддержки кода.

Тестирование подтвердило корректность работы всех функций бота, включая обработку пользовательских команд, интеграцию с базой данных и планировщиком задач Quartz. Пользовательский интерфейс был оценен как интуитивно понятный и удобный, что соответствует современным требованиям к подобным приложениям.

В перспективе возможно расширение функционала, например, добавление напоминаний с помощью голосовых сообщений и интеграции с другими сервисами.

Таким образом, проект достиг поставленных целей и может служить основой для дальнейшего развития.

					ПГУ1.090301.48.001 ПЗ	Лист
						50
Изм.	Лист	№ докум.	Подпись	Дата		

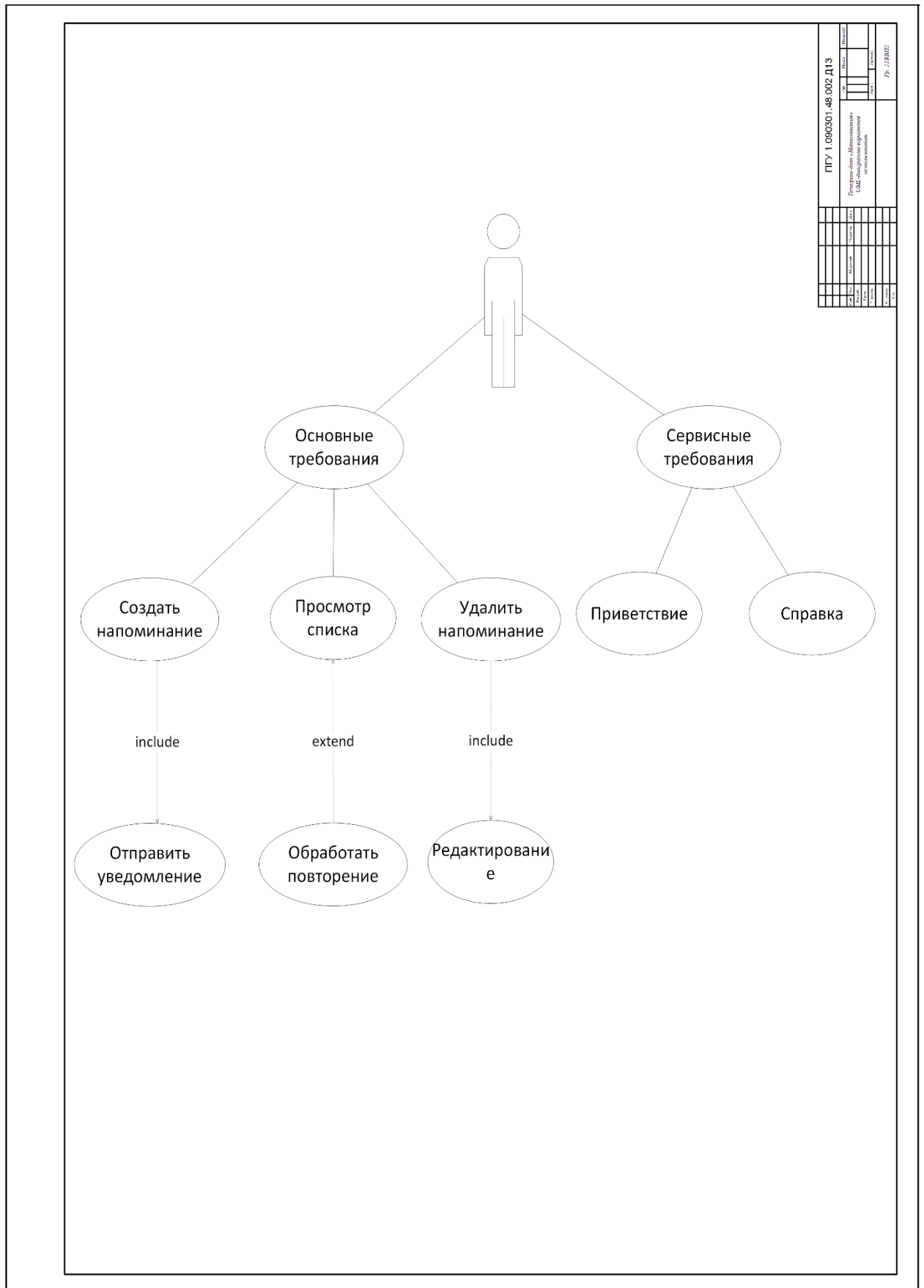
Список использованных источников

1. Чат-боты, и как нам с этим поможет Microsoft. URL: <https://habr.com/ru/companies/pvs-studio/articles/311592/>
(дата обращения 06.05.2025)
2. Google Keep. URL: <https://keep.google.com/>
(дата обращения 09.05.2025)
3. Microsoft To Do. URL: <https://to-do.office.com/tasks/>
(дата обращения 09.05.2025)
4. Telegram Bot C#. URL: <https://ru.stackoverflow.com/questions/1562257/telegram-bot-c> (дата обращения 11.05.2025)
5. Quartz 3 Quick Start. URL: <https://www.quartzscheduler.net/documentation/quartz-3.x/quick-start.html> (дата обращения 16.05.2025)
6. Обзор Microsoft.Data.Sqlite. URL: <https://learn.microsoft.com/ru-ru/dotnet/standard/data/sqlite/?tabs=net-cli> (дата обращения 16.05.2025)
7. Visual Studio 2022. URL: <https://visualstudio.microsoft.com/ru/vs/>
(дата обращения 16.05.2025)
8. Библиотека Telegram.Bot.Extensions.Polling. URL: <https://github.com/TelegramBots/Telegram.Bot.Extensions.Polling> (дата обращения 17.05.2025)
9. Telegram Client API library. URL: <https://github.com/wiz0u/WTelegramClient> (дата обращения 17.05.2025)
10. Telegram.bot. .NET Client for Telegram Bot. API. URL: <https://github.com/TelegramBots/Telegram.Bot> (дата обращения 18.05.2025)
11. Microsoft Learn – Официальная документация по C#, .NET и Bot Framework. URL: <https://learn.microsoft.com/ru-ru/azure/bot-service/bot-service-quickstart-create-bot?view=azure-bot-service-4.0&tabs=csharp%2Cvs>
(дата обращения 18.05.2025)
12. Microsoft Docs – Официальная документация. URL: <https://learn.microsoft.com/en-us/dotnet/fundamentals/> (дата обращения 25.05.2025)

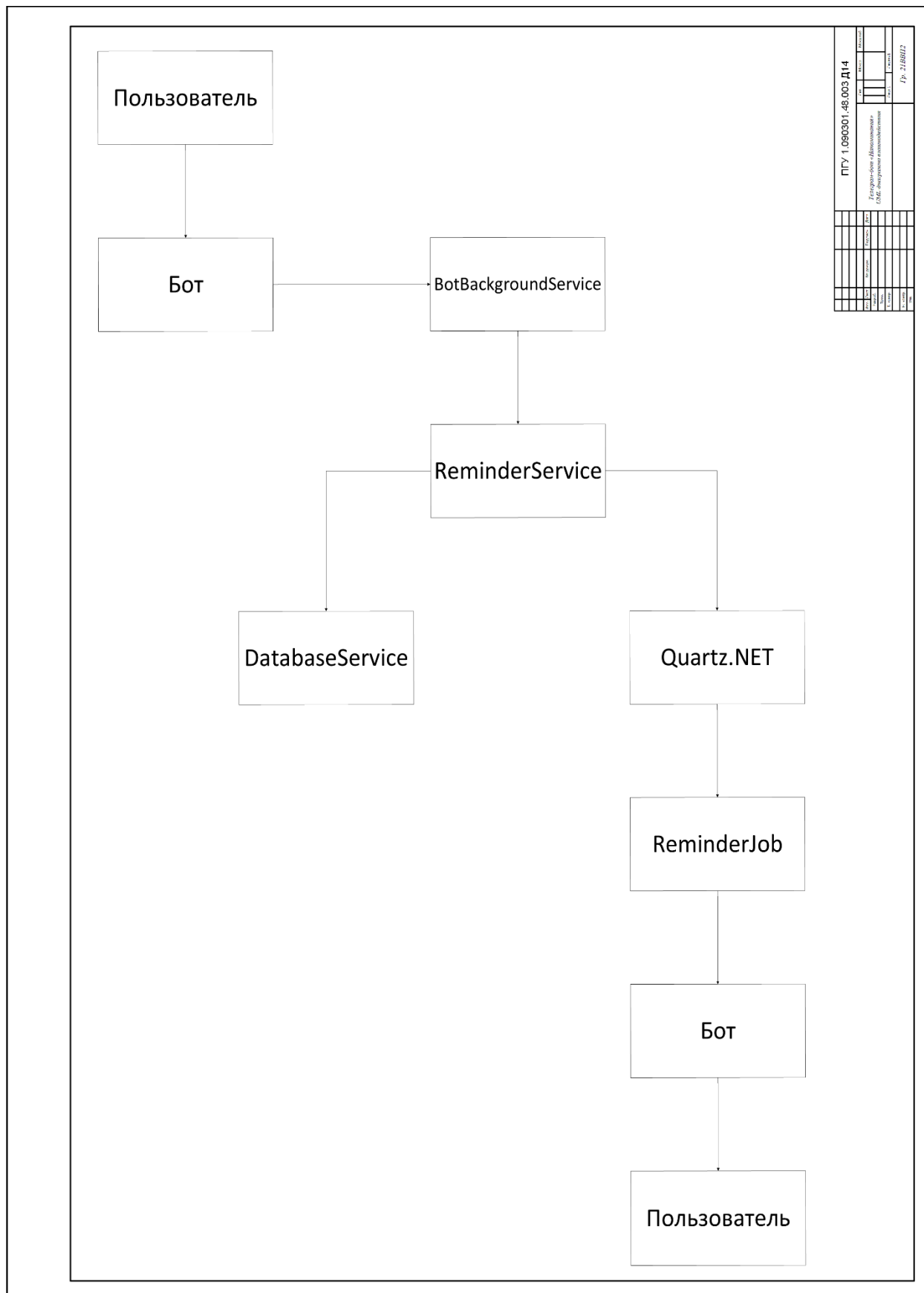
Приложение А
(Обязательное)
«Телеграм-бот “Напоминания”»
UML – диаграммы

					<i>ПГУ1.090301.48.001 ПЗ</i>	Лист
Изм.	Лист	№ докум.	Подпись	Дата		52

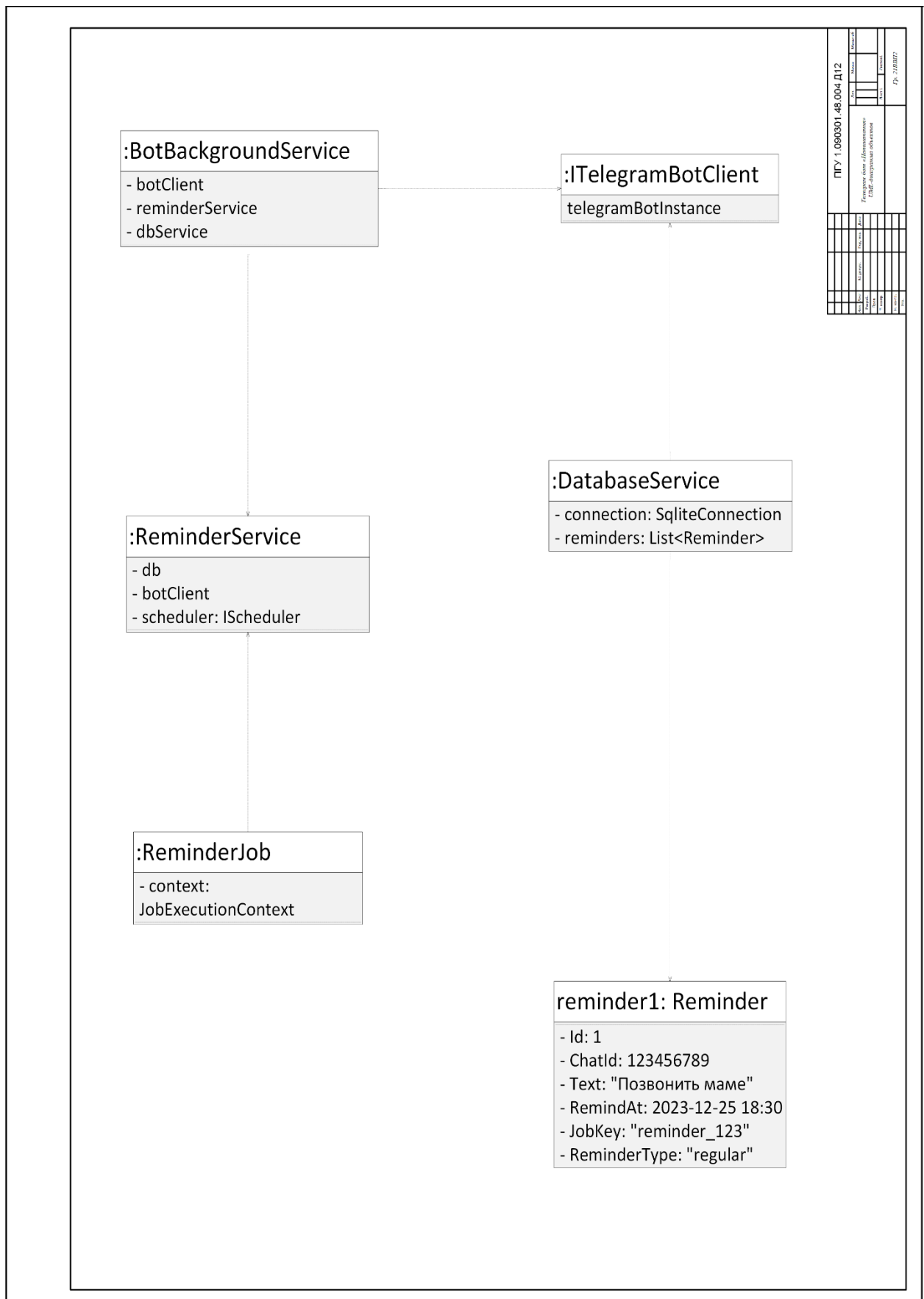
Приложение А.1 – UML – диаграмма вариантов использования



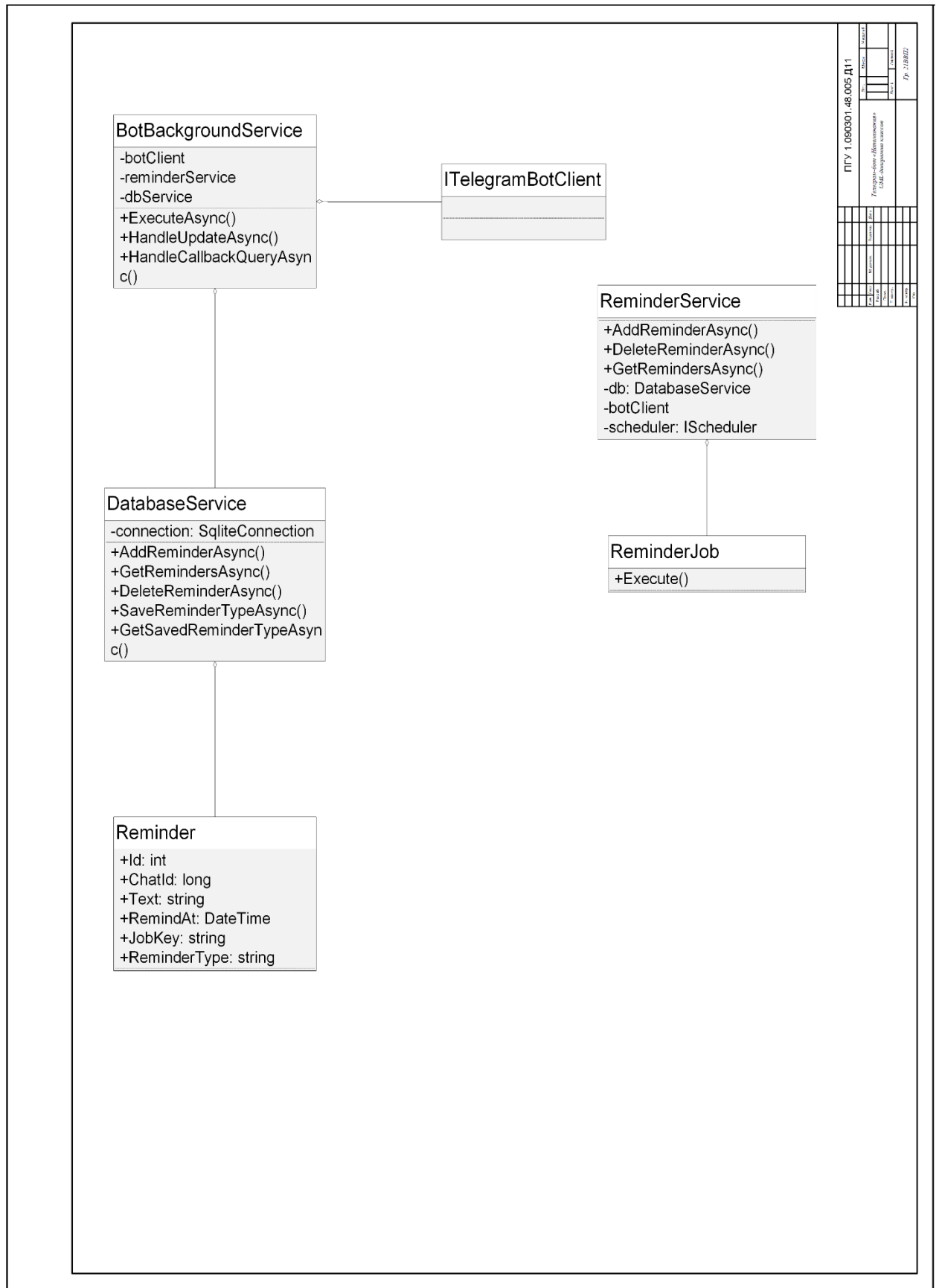
Приложение А.2 – UML – диаграмма взаимодействия компонентов



Приложение А.3 – UML – диаграмма объектов



Приложение А.4 – UML – диаграмма классов



ПГУ 1.090301.48.005 Д11									
Техническое задание на разработку									
UML-диаграмма классов									
№	Имя	Дата	Стр.	Всего стр.	Всего листов	Всего листов	Всего листов	Всего листов	Всего листов
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9
10	10	10	10	10	10	10	10	10	10
11	11	11	11	11	11	11	11	11	11
12	12	12	12	12	12	12	12	12	12
13	13	13	13	13	13	13	13	13	13
14	14	14	14	14	14	14	14	14	14
15	15	15	15	15	15	15	15	15	15
16	16	16	16	16	16	16	16	16	16
17	17	17	17	17	17	17	17	17	17
18	18	18	18	18	18	18	18	18	18
19	19	19	19	19	19	19	19	19	19
20	20	20	20	20	20	20	20	20	20
21	21	21	21	21	21	21	21	21	21
22	22	22	22	22	22	22	22	22	22
23	23	23	23	23	23	23	23	23	23
24	24	24	24	24	24	24	24	24	24
25	25	25	25	25	25	25	25	25	25
26	26	26	26	26	26	26	26	26	26
27	27	27	27	27	27	27	27	27	27
28	28	28	28	28	28	28	28	28	28
29	29	29	29	29	29	29	29	29	29
30	30	30	30	30	30	30	30	30	30
31	31	31	31	31	31	31	31	31	31
32	32	32	32	32	32	32	32	32	32
33	33	33	33	33	33	33	33	33	33
34	34	34	34	34	34	34	34	34	34
35	35	35	35	35	35	35	35	35	35
36	36	36	36	36	36	36	36	36	36
37	37	37	37	37	37	37	37	37	37
38	38	38	38	38	38	38	38	38	38
39	39	39	39	39	39	39	39	39	39
40	40	40	40	40	40	40	40	40	40
41	41	41	41	41	41	41	41	41	41
42	42	42	42	42	42	42	42	42	42
43	43	43	43	43	43	43	43	43	43
44	44	44	44	44	44	44	44	44	44
45	45	45	45	45	45	45	45	45	45
46	46	46	46	46	46	46	46	46	46
47	47	47	47	47	47	47	47	47	47
48	48	48	48	48	48	48	48	48	48
49	49	49	49	49	49	49	49	49	49
50	50	50	50	50	50	50	50	50	50
51	51	51	51	51	51	51	51	51	51
52	52	52	52	52	52	52	52	52	52
53	53	53	53	53	53	53	53	53	53
54	54	54	54	54	54	54	54	54	54
55	55	55	55	55	55	55	55	55	55
56	56	56	56	56	56	56	56	56	56
57	57	57	57	57	57	57	57	57	57
58	58	58	58	58	58	58	58	58	58
59	59	59	59	59	59	59	59	59	59
60	60	60	60	60	60	60	60	60	60
61	61	61	61	61	61	61	61	61	61
62	62	62	62	62	62	62	62	62	62
63	63	63	63	63	63	63	63	63	63
64	64	64	64	64	64	64	64	64	64
65	65	65	65	65	65	65	65	65	65
66	66	66	66	66	66	66	66	66	66
67	67	67	67	67	67	67	67	67	67
68	68	68	68	68	68	68	68	68	68
69	69	69	69	69	69	69	69	69	69
70	70	70	70	70	70	70	70	70	70
71	71	71	71	71	71	71	71	71	71
72	72	72	72	72	72	72	72	72	72
73	73	73	73	73	73	73	73	73	73
74	74	74	74	74	74	74	74	74	74
75	75	75	75	75	75	75	75	75	75
76	76	76	76	76	76	76	76	76	76
77	77	77	77	77	77	77	77	77	77
78	78	78	78	78	78	78	78	78	78
79	79	79	79	79	79	79	79	79	79
80	80	80	80	80	80	80	80	80	80
81	81	81	81	81	81	81	81	81	81
82	82	82	82	82	82	82	82	82	82
83	83	83	83	83	83	83	83	83	83
84	84	84	84	84	84	84	84	84	84
85	85	85	85	85	85	85	85	85	85
86	86	86	86	86	86	86	86	86	86
87	87	87	87	87	87	87	87	87	87
88	88	88	88	88	88	88	88	88	88
89	89	89	89	89	89	89	89	89	89
90	90	90	90	90	90	90	90	90	90
91	91	91	91	91	91	91	91	91	91
92	92	92	92	92	92	92	92	92	92
93	93	93	93	93	93	93	93	93	93
94	94	94	94	94	94	94	94	94	94
95	95	95	95	95	95	95	95	95	95
96	96	96	96	96	96	96	96	96	96
97	97	97	97	97	97	97	97	97	97
98	98	98	98	98	98	98	98	98	98
99	99	99	99	99	99	99	99	99	99
100	100	100	100	100	100	100	100	100	100

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.48.001 ПЗ

Лист
56

Приложение Б

(Обязательное)

«Телеграм-бот “Напоминания”»

Листинг программы

					ПГУ1.090301.48.001 ПЗ	Лист
						57
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «BotBackgroundService.cs»

```

using Microsoft.Extensions.Hosting;
using Telegram.Bot;
using Telegram.Bot.Polling;
using Telegram.Bot.Types;
using Telegram.Bot.Types.Enums;
using Telegram.Bot.Types.ReplyMarkups;
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Linq;

public class BotBackgroundService : BackgroundService
{
    private readonly ITelegramBotClient _botClient;
    private readonly ReminderService _reminderService;
    private readonly DatabaseService _dbService;

    public BotBackgroundService(ITelegramBotClient botClient,
        ReminderService reminderService, DatabaseService dbService)
    {
        _botClient = botClient;
        _reminderService = reminderService;
        _dbService = dbService;
    }

    protected override async Task ExecuteAsync(CancellationToken
        stoppingToken)
    {
        var receiverOptions = new ReceiverOptions
        {
            AllowedUpdates = new[] { UpdateType.Message,
UpdateType.CallbackQuery }
        };

        _botClient.StartReceiving(
            updateHandler: HandleUpdateAsync,
            errorHandler: HandlePollingErrorAsync,
            receiverOptions: receiverOptions,
            cancellationToken: stoppingToken);

        var me = await _botClient.GetMe(stoppingToken);
        Console.WriteLine($"Бот @{me.Username} запущен и готов к
работе!");

        while (!stoppingToken.IsCancellationRequested)
            await Task.Delay(1000, stoppingToken);
    }

    private async Task HandleUpdateAsync(ITelegramBotClient
        botClient, Update update, CancellationToken cancellationToken)
    
```

					ПГУ1.090301.48.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

```

        {
            if (update.CallbackQuery != null)
            {
                await HandleCallbackQueryAsync(botClient,
update.CallbackQuery, cancellationToken);
                return;
            }

            if (update.Message is not { Text: { } messageText }
message)
                return;

            var chatId = message.Chat.Id;
            Console.WriteLine($"Получено сообщение от {chatId}:
{messageText}");

            try
            {
                switch (messageText)
                {
                    case "/start":
                        await SendWelcomeMessage(chatId,
cancellationToken);
                        break;

                    case "/menu":
                        await ShowMainMenu(chatId,
cancellationToken);
                        break;

                    case "+ Добавить напоминание":
                        await botClient.SendMessage(
                            chatId: chatId,
                            text: "Выберите тип напоминания:",
                            replyMarkup:
GetReminderTypeSelectionMarkup(),
                            cancellationToken: cancellationToken);
                        break;

                    case "📅 Мои напоминания":
                        await SendRemindersListAsync(chatId,
cancellationToken);
                        break;

                    case "ℹ️ Помощь":
                        await SendHelpMessage(chatId,
cancellationToken);
                        break;

                    case "🔙 Назад":

```

```

        await ShowMainMenu(chatId,
cancellationToken);
        break;

        case string s when s.StartsWith("✕ Удалить"):
            await HandleReminderDeletion(chatId, s,
cancellationToken);
            break;

        default:
            await HandleDefaultMessage(chatId,
messageText, cancellationToken);
            break;
    }
}
catch (Exception ex)
{
    Console.WriteLine($"Ошибка обработки сообщения:
{ex}");
    await botClient.SendMessage(
        chatId: chatId,
        text: "Произошла ошибка при обработке вашего
запроса.",
        replyMarkup: GetBackButtonMarkup(),
        cancellationToken: cancellationToken);
}

private async Task SendWelcomeMessage(long chatId,
CancellationToken cancellationToken)
{
    await _botClient.SendMessage(
        chatId: chatId,
        text: "👋 Добро пожаловать в ReminderBot - вашего
надежного помощника в управлении временем!\n\n" +
            "📌 <b>Основные возможности бота:</b>\n\n" +
            "• Создание <b>разовых</b> напоминаний на
конкретную дату и время\n" +
            "• Напоминания с <b>предварительными с
уведомлениями</b> (за 1 и 2 дня)\n" +
            "• <b>Повторяющиеся</b> напоминания
(ежечасные, ежедневные, еженедельные)\n" +
            "• Удобный просмотр всех активных
напоминаний\n" +
            "• Простое управление через интерактивное
меню\n\n" +
            "💡 <b>Как начать работу:</b>\n" +
            "1. Нажмите «➕ Добавить напоминание»\n" +
            "2. Выберите тип напоминания\n" +

```

```

+
        "3. Введите текст и дату (если требуется)\n\n"

        "📖 Для получения более подробной информации
нажмите « Помощь»",
        parseMode: ParseMode.Html,
        replyMarkup: GetMainMenuMarkup(),
        cancellationToken: cancellationToken);
    }

    private async Task
    HandleCallbackQueryAsync(ITelegramBotClient botClient,
    CallbackQuery callbackQuery, CancellationToken
    cancellationToken)
    {
        var chatId = callbackQuery.Message.Chat.Id;
        var data = callbackQuery.Data;

        try
        {
            string botMessage = null;
            string reminderType = null;

            switch (data)
            {
                case "regular_reminder":
                    botMessage = "Введите текст и дату в
формате:\nПозвонить маме 25 декабря 18:30";
                    reminderType = "regular";
                    break;

                case "pre_reminder":
                    botMessage = "Введите текст и дату в
формате:\nПозвонить маме 25 декабря 18:30\n\nБот пришлет
напоминание за 2 дня и за 1 день до указанной даты.";
                    reminderType = "pre_reminder";
                    break;

                case "recurring_reminder":

                    await botClient.SendMessage(
                        chatId: chatId,
                        text: "Выберите тип повторяющегося
напоминания:",
                        replyMarkup:
    GetRecurringReminderTypesMarkup(),
                        cancellationToken: cancellationToken);
                    await
    botClient.AnswerCallbackQuery(callbackQuery.Id,
    cancellationToken: cancellationToken);
                    return;
            }
        }
    }

```

```

        case "hourly_reminder":
            botMessage = "Введите текст для ежечасного
напоминания:";
            reminderType = "hourly_reminder";
            break;

        case "daily_reminder":
            botMessage = "Введите текст для ежедневного
напоминания:";
            reminderType = "daily_reminder";
            break;

        case "weekly_reminder":
            botMessage = "Введите текст для
еженедельного напоминания:";
            reminderType = "weekly_reminder";
            break;
    }

    if (botMessage != null)
    {
        await botClient.SendMessage(
            chatId: chatId,
            text: botMessage,
            replyMarkup: GetBackButtonMarkup(),
            cancellationToken: cancellationToken);

        await _dbService.SaveReminderTypeAsync(chatId,
reminderType);
    }

    await
botClient.AnswerCallbackQuery(callbackQuery.Id,
cancellationToken: cancellationToken);
}
catch (Exception ex)
{
    Console.WriteLine($"Ошибка обработки callback:
{ex}");
    await botClient.SendMessage(
        chatId: chatId,
        text: "Произошла ошибка при обработке вашего
запроса.",
        replyMarkup: GetBackButtonMarkup(),
        cancellationToken: cancellationToken);
}
}

private async Task ProcessDefaultMessage(long chatId, string
messageText, Cancellation token cancellationToken)
{
    try

```

```

        {
            var reminderType = await
_dbService.GetSavedReminderTypeAsync(chatId);

            if (reminderType == null)
            {
                if (messageText.Split(' ').Length >= 4)
                {
                    reminderType = "regular";
                }
                else
                {
                    await _botClient.SendMessage(
                        chatId: chatId,
                        text: "Неизвестная команда. Используйте
меню.",
                        replyMarkup: GetMainMenuMarkup(),
                        cancellationTokens: cancellationTokens);
                    return;
                }
            }

            Console.WriteLine($"Создание напоминания типа:
{reminderType}");
            await _reminderService.AddReminderAsync(chatId,
messageText, reminderType);

            await
_dbService.ClearSavedReminderTypeAsync(chatId);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Ошибка обработки сообщения:
{ex}");
            await _botClient.SendMessage(
                chatId: chatId,
                text: "Произошла ошибка при обработке вашего
запроса.",
                replyMarkup: GetBackButtonMarkup(),
                cancellationTokens: cancellationTokens);
        }
    }

    private async Task HandleDefaultMessage(long chatId, string
messageText, CancellationToken cancellationTokens)
    {
        try
        {
            var reminderType = await
_dbService.GetSavedReminderTypeAsync(chatId);

```

```

        if (reminderType == null)
        {
            if (messageText.Split(' ').Length >= 4)
            {
                reminderType = "regular";
            }
            else
            {
                await _botClient.SendMessage(
                    chatId: chatId,
                    text: "Неизвестная команда. Используйте
меню.",
                    replyMarkup: GetMainMenuMarkup(),
                    cancellationToken: cancellationToken);
                return;
            }
        }

        Console.WriteLine($"Создание напоминания типа:
{reminderType}");
        await _reminderService.AddReminderAsync(chatId,
messageText, reminderType);

        await
_dbService.ClearSavedReminderTypeAsync(chatId);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Ошибка обработки сообщения:
{ex}");
        await _botClient.SendMessage(
            chatId: chatId,
            text: "Произошла ошибка при обработке вашего
запроса.",
            replyMarkup: GetBackButtonMarkup(),
            cancellationToken: cancellationToken);
    }
}

private bool TryParseRussianDateTime(string dateString, out
DateTime result)
{
    result = DateTime.MinValue;
    try
    {
        {
            var months = new Dictionary<string, int>
            {
                {"января", 1}, {"февраля", 2}, {"марта", 3},
{"апреля", 4},
                {"мая", 5}, {"июня", 6}, {"июля", 7},
{"августа", 8},

```

```

        {"сентября", 9}, {"октября", 10}, {"ноября",
11}, {"декабря", 12}
        };

        var parts = dateString.Split(' ');
        if (parts.Length != 3) return false;

        if (!int.TryParse(parts[0], out var day)) return
false;
        if (!months.TryGetValue(parts[1].ToLower(), out var
month)) return false;
        if (!TimeSpan.TryParse(parts[2], out var time))
return false;

        var now = DateTime.Now;
        result = new DateTime(now.Year, month, day,
time.Hours, time.Minutes, 0);

        if (result < now)
            result = result.AddYears(1);

        return true;
    }
    catch
    {
        return false;
    }
}

private ReplyKeyboardMarkup GetBackButtonMarkup()
{
    return new ReplyKeyboardMarkup(new[] { new
KeyboardButton("🔙 Назад") }) { ResizeKeyboard = true };
}

private async Task SendRemindersListAsync(long chatId,
Cancellation token cancellationToken)
{
    var reminders = await
_dbService.GetRemindersAsync(chatId);

    if (!reminders.Any())
    {
        await _botClient.SendMessage(chatId, "📧 У вас нет
активных напоминаний",
            replyMarkup: GetBackButtonMarkup(),
            cancellationToken: cancellationToken);
        return;
    }
}

```

```

        var message = "📅 Ваши напоминания:\n\n" +
string.Join("\n", reminders
        .OrderBy(r => r.RemindAt)
        .Select(r => $"🕒 {r.Text} - {r.RemindAt:dd MMMM
HH:mm}"));

        var buttons = reminders
        .Select(r => new[] { new KeyboardButton($"✖ Удалить
{r.Text.Split(' ').FirstOrDefault()} ({r.Id})") })
        .Concat(new[] { new[] { new KeyboardButton("🏠
Назад") } });

        await _botClient.SendMessage(
            chatId: chatId,
            text: message,
            replyMarkup: new ReplyKeyboardMarkup(buttons) {
                ResizeKeyboard = true },
            cancellationTokens: cancellationTokens);
    }

    private async Task HandleReminderDeletion(long chatId,
string messageText, CancellationToken cancellationToken)
    {
        try
        {
            var start = messageText.IndexOf('(') + 1;
            var end = messageText.IndexOf(')');
            if (start <= 0 || end <= 0 || end <= start)
                throw new FormatException();

            if (!int.TryParse(messageText.Substring(start, end -
start), out var reminderId))
                throw new FormatException();

            var reminder = (await
_dbService.GetRemindersAsync(chatId)).FirstOrDefault(r => r.Id
== reminderId);
            if (reminder == null)
                throw new KeyNotFoundException();

            await
_reminderService.DeleteReminderAsync(reminder.JobKey);
            await _botClient.SendMessage(chatId, $"✅
Напоминание удалено: {reminder.Text}",
                replyMarkup: GetBackButtonMarkup(),
                cancellationTokens: cancellationTokens);
            await SendRemindersListAsync(chatId,
cancellationTokens);
        }
        catch (Exception ex)
    }

```

```

        {
            Console.WriteLine($"Ошибка при удалении напоминания:
{ex}");
            await _botClient.SendMessage(chatId, "❌ Не удалось
удалить напоминание",
                replyMarkup: GetBackButtonMarkup(),
                cancellationToken: cancellationToken);
        }
    }

    private async Task ShowMainMenu(long chatId,
CancellationToken cancellationToken)
    {
        await _botClient.SendMessage(
            chatId: chatId,
            text: "Выберите действие:",
            replyMarkup: GetMainMenuMarkup(),
            cancellationToken: cancellationToken);
    }

    private ReplyKeyboardMarkup GetMainMenuMarkup()
    {
        return new ReplyKeyboardMarkup(new[]
        {
            new[] { new KeyboardButton("✚ Добавить
напоминание") },
            new[] { new KeyboardButton("📅 Мои напоминания"), new
KeyboardButton("ℹ️ Помощь") }
        })
        { ResizeKeyboard = true };
    }

    private async Task SendHelpMessage(long chatId,
CancellationToken cancellationToken)
    {
        await _botClient.SendMessage(
            chatId: chatId,
            text: "🔗 Подробная инструкция по использованию
бота:\n\n" +
                "💎 <b>Добавление напоминания</b>\n" +
                "1. Нажмите кнопку «✚ Добавить напоминание»\n"
            +
                "2. Введите текст и дату в формате:\n" +
                "    <i>Пример: Позвонить маме 25 декабря
18:30</i>\n\n" +
                "💎 <b>Просмотр напоминаний</b>\n" +
                "Нажмите кнопку «📅 Мои напоминания» чтобы
увидеть список всех ваших напоминаний\n\n" +
                "💎 <b>Удаление напоминаний</b>\n" +

```

```

        "1. Откройте список напоминаний\n" +
        "2. Нажмите кнопку «X Удалить» напротив\n\n" +
        "♦ <b>Навигация</b>\n" +
        "• « Назад» - вернуться в главное меню\n" +
        "• «/menu» - открыть главное меню в любой\n\n" +
        "Бот автоматически пришлет вам напоминание в\n\n" +
        "указанное время!",
        parseMode: ParseMode.Html,
        replyMarkup: GetBackButtonMarkup(),
        cancellationToken: cancellationToken);
    }

    private InlineKeyboardMarkup
    GetReminderTypeSelectionMarkup()
    {
        return new InlineKeyboardMarkup(new[]
        {
            new[]
            {
                InlineKeyboardButton.WithCallbackData("Обычное",
                "regular_reminder"),
                InlineKeyboardButton.WithCallbackData("С
                предупреждением", "pre_reminder")
            },
            new[]
            {
                InlineKeyboardButton.WithCallbackData("Повторяющееся",
                "recurring_reminder")
            }
        });
    }

    private InlineKeyboardMarkup
    GetRecurringReminderTypesMarkup()
    {
        return new InlineKeyboardMarkup(new[]
        {
            new[]
            {
                InlineKeyboardButton.WithCallbackData("Каждый час",
                "hourly_reminder"),
                InlineKeyboardButton.WithCallbackData("Каждый день",
                "daily_reminder")
            },
            new[]
            {
                InlineKeyboardButton.WithCallbackData("Каждую
                неделю", "weekly_reminder")
            }
        });
    }

```

```

    }
    });
}

private Task HandlePollingErrorAsync(ITelegramBotClient
botClient, Exception exception, CancellationToken
cancellation_token)
{
    Console.WriteLine($"Ошибка Telegram Bot:
{exception.Message}");
    return Task.CompletedTask;
}
}

```

Файл «DatabaseService.cs»

```

using Microsoft.Data.Sqlite;
using ReminderBot;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

public class DatabaseService : IDisposable
{
    private readonly SqliteConnection _connection;

    public DatabaseService()
    {
        _connection = new SqliteConnection("Data
Source=reminders.db");
        InitializeDatabase();
    }

    private void InitializeDatabase()
    {
        _connection.Open();
        using var cmd = _connection.CreateCommand();

        cmd.CommandText = @"
CREATE TABLE IF NOT EXISTS ReminderTypes (
ChatId INTEGER PRIMARY KEY,
ReminderType TEXT NOT NULL
)";
        cmd.ExecuteNonQuery();

        cmd.CommandText = @"
CREATE TABLE IF NOT EXISTS Messages (
Id INTEGER PRIMARY KEY AUTOINCREMENT,
ChatId INTEGER NOT NULL,
Text TEXT NOT NULL,
IsFromBot BOOLEAN NOT NULL,

```

```

        CreatedAt TEXT NOT NULL DEFAULT CURRENT_TIMESTAMP
    );
    cmd.ExecuteNonQuery();

    cmd.CommandText = @"
CREATE TABLE IF NOT EXISTS Reminders (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    ChatId INTEGER NOT NULL,
    Text TEXT NOT NULL,
    RemindAt TEXT NOT NULL,
    JobKey TEXT NOT NULL UNIQUE,
    ReminderType TEXT NOT NULL DEFAULT 'regular'
);
    cmd.ExecuteNonQuery();
}

public async Task AddReminderAsync(long chatId, string text,
DateTime remindAt, string jobKey, string reminderType)
{
    try
    {
        using var cmd = _connection.CreateCommand();
        cmd.CommandText = @"
INSERT INTO Reminders (ChatId, Text, RemindAt,
JobKey, ReminderType)
VALUES (@chatId, @text, @remindAt, @jobKey,
@reminderType)";

        cmd.Parameters.AddWithValue("@chatId", chatId);
        cmd.Parameters.AddWithValue("@text", text);
        cmd.Parameters.AddWithValue("@remindAt",
remindAt.ToString("O"));
        cmd.Parameters.AddWithValue("@jobKey", jobKey);
        cmd.Parameters.AddWithValue("@reminderType",
reminderType);

        await cmd.ExecuteNonQueryAsync();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Ошибка при добавлении
напоминания: {ex}");
        throw;
    }
}

public async Task SaveReminderTypeAsync(long chatId, string
reminderType)
{
    try
    {

```

```

        using var cmd = _connection.CreateCommand();
        cmd.CommandText = @"
            INSERT OR REPLACE INTO ReminderTypes (ChatId,
ReminderType)
            VALUES (@chatId, @reminderType)";

        cmd.Parameters.AddWithValue("@chatId", chatId);
        cmd.Parameters.AddWithValue("@reminderType",
reminderType);

        await cmd.ExecuteNonQueryAsync();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Ошибка сохранения типа
напоминания: {ex}");
    }
}

public async Task<string> GetSavedReminderTypeAsync(long
chatId)
{
    try
    {
        using var cmd = _connection.CreateCommand();
        cmd.CommandText = "SELECT ReminderType FROM
ReminderTypes WHERE ChatId = @chatId";
        cmd.Parameters.AddWithValue("@chatId", chatId);

        return (await cmd.ExecuteScalarAsync())?.ToString();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Ошибка получения типа
напоминания: {ex}");
        return null;
    }
}

public async Task ClearSavedReminderTypeAsync(long chatId)
{
    try
    {
        using var cmd = _connection.CreateCommand();
        cmd.CommandText = "DELETE FROM ReminderTypes WHERE
ChatId = @chatId";
        cmd.Parameters.AddWithValue("@chatId", chatId);

        await cmd.ExecuteNonQueryAsync();
    }
    catch (Exception ex)

```

```

        {
            Console.WriteLine($"Ошибка очистки типа напоминания:
{ex}");
        }
    }

    public async Task<Reminder> GetReminderByJobKeyAsync(string
jobKey)
    {
        try
        {
            using var cmd = _connection.CreateCommand();
            cmd.CommandText = "SELECT Id, ChatId, Text,
RemindAt, JobKey, ReminderType FROM Reminders WHERE JobKey =
@jobKey";
            cmd.Parameters.AddWithValue("@jobKey", jobKey);

            using var reader = await cmd.ExecuteReaderAsync();
            if (await reader.ReadAsync())
            {
                return new Reminder
                {
                    Id = reader.GetInt32(0),
                    ChatId = reader.GetInt64(1),
                    Text = reader.GetString(2),
                    RemindAt =
DateTime.Parse(reader.GetString(3)),
                    JobKey = reader.GetString(4),
                    ReminderType = reader.GetString(5)
                };
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Ошибка при получении
напоминания: {ex}");
        }
        return null;
    }

    public async Task<List<Reminder>> GetRemindersAsync(long
chatId)
    {
        var reminders = new List<Reminder>();
        try
        {
            using var cmd = _connection.CreateCommand();
            cmd.CommandText = "SELECT Id, Text, RemindAt,
JobKey, ReminderType FROM Reminders WHERE ChatId = @chatId";
            cmd.Parameters.AddWithValue("@chatId", chatId);

```

```

        using var reader = await cmd.ExecuteReaderAsync();
        while (await reader.ReadAsync())
        {
            reminders.Add(new Reminder
            {
                Id = reader.GetInt32(0),
                Text = reader.GetString(1),
                RemindAt =
DateTime.Parse(reader.GetString(2)),
                JobKey = reader.GetString(3),
                ReminderType = reader.GetString(4)
            });
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Ошибка при чтении из БД: {ex}");
    }
    return reminders;
}

public async Task DeleteReminderAsync(string jobKey)
{
    try
    {
        using var cmd = _connection.CreateCommand();
        cmd.CommandText = "DELETE FROM Reminders WHERE
JobKey = @jobKey";
        cmd.Parameters.AddWithValue("@jobKey", jobKey);
        await cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Ошибка при удалении напоминания:
{ex}");
        throw;
    }
}

public void Dispose()
{
    _connection?.Close();
    _connection?.Dispose();
}
}

```

Файл «Program.cs»

```

using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Quartz;

```

					ПГУ1.090301.48.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

```

using Telegram.Bot;

var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices((context, services) =>
{
    services.AddSingleton<ITelegramBotClient>(_ =>
        new
TelegramBotClient("7420047915:AAF8tk2PQUVB2vOAFxub_5sC5wp_HYvYgO
M"));

    services.AddSingleton<DatabaseService>();
    services.AddSingleton<ReminderService>();

    services.AddQuartz(q =>
    {
        q.UseMicrosoftDependencyInjectionJobFactory();
        q.UseSimpleTypeLoader();
        q.UseInMemoryStore();
    });

    services.AddQuartzHostedService(q => q.WaitForJobsToComplete
= true);
    services.AddHostedService<BotBackgroundService>();
});

var host = builder.Build();

var reminderService =
host.Services.GetRequiredService<ReminderService>();
await reminderService.InitializeAsync();

await host.RunAsync();

```

Файл «Reminder.cs»

```

using System;

namespace ReminderBot
{
    public class Reminder
    {
        public int Id { get; set; }
        public long ChatId { get; set; }
        public string Text { get; set; }
        public DateTime RemindAt { get; set; }
        public string JobKey { get; set; }
        public string ReminderType { get; set; } = "regular";
    }
}

```

Файл «ReminderJob.cs»

```
using Quartz;
using System.Threading.Tasks;
using Telegram.Bot;

public class ReminderJob : IJob
{
    public async Task Execute(IJobExecutionContext context)
    {
        var data = context.MergedJobDataMap;
        var chatId = data.GetLong("chatId");
        var text = data.GetString("text");
        var jobKey = data.GetString("jobKey");
        var botClient =
            (ITelegramBotClient)context.Scheduler.Context.Get("botClient");
        var db =
            (DatabaseService)context.Scheduler.Context.Get("dbService");

        try
        {
            var reminder = await
                db.GetReminderByJobKeyAsync(jobKey);
            if (reminder == null) return;

            string message = reminder.ReminderType switch
            {
                "pre_reminder" when
                    context.Trigger.Key.Name.Contains("_2days") => $"⌚ Напоминание
                    (через 2 дня): {text}",
                "pre_reminder" when
                    context.Trigger.Key.Name.Contains("_1day") => $"⌚ Напоминание
                    (через 1 день): {text}",
                _ => $"🔔 Напоминание: {text}"
            };

            await botClient.SendMessage(chatId, message);

            // Удаляем только одноразовые напоминания после
            срабатывания
            if (reminder.ReminderType == "regular" &&
                context.Trigger.Key.Name == $"{jobKey}_trigger")
            {
                await db.DeleteReminderAsync(jobKey);
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Ошибка при отправке напоминания:
            {ex}");
        }
    }
}
```

					ПГУ1.090301.48.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }

}

```

Файл «ReminderService.cs»

```

using Quartz;
using Quartz.Impl;
using System;
using System.Threading.Tasks;
using Telegram.Bot;
using System.Collections.Generic;
using ReminderBot;

public class ReminderService : IAsyncDisposable
{
    private readonly DatabaseService _db;
    private readonly ITelegramBotClient _botClient;
    private IScheduler _scheduler;

    public ReminderService(DatabaseService db,
        ITelegramBotClient botClient)
    {
        _db = db;
        _botClient = botClient;
    }

    public async Task InitializeAsync()
    {
        var factory = new StdSchedulerFactory();
        _scheduler = await factory.GetScheduler();
        _scheduler.Context.Put("botClient", _botClient);
        _scheduler.Context.Put("dbService", _db);
        await _scheduler.Start();
    }

    public async Task AddReminderAsync(long chatId, string
        messageText, string reminderType)
    {
        try
        {
            Console.WriteLine($"Начало создания напоминания
                типа: {reminderType}");

            DateTime remindAt = DateTime.MinValue;
            string text = messageText;

            if (reminderType == "regular" || reminderType ==
                "pre_reminder")
            {
                var parts = messageText.Split(' ');
                if (parts.Length < 4)

```

```

        {
            await _botClient.SendMessage(chatId, "✗
Неверный формат. Пример: Поздравить с ДР 25 декабря 18:30");
            return;
        }

        text = string.Join(" ", parts.Take(parts.Length
- 3));
        var dateString = string.Join(" ",
parts.Skip(parts.Length - 3));

        if (!TryParseRussianDateTime(dateString, out
remindAt))
        {
            await _botClient.SendMessage(chatId, "✗
Неверная дата. Формат: 25 декабря 18:30");
            return;
        }

        if (remindAt < DateTime.Now)
        {
            await _botClient.SendMessage(chatId, "✗
Указанная дата и время уже прошли");
            return;
        }
    }

    Console.WriteLine($"Создание напоминания. Тип:
{reminderType}, Текст: {text}");
    if (reminderType == "hourly_reminder" ||
reminderType == "daily_reminder" || reminderType ==
"weekly_reminder")
    {
        Console.WriteLine($"Это повторяющееся
напоминание с интервалом: {reminderType}");
    }

    var jobKey = $"reminder_{Guid.NewGuid()}";
    await _db.AddReminderAsync(chatId, text,
        reminderType == "hourly_reminder" ||
reminderType == "daily_reminder" || reminderType ==
"weekly_reminder"
        ? DateTime.Now
        : remindAt,
        jobKey,
        reminderType);

    IJobDetail job = JobBuilder.Create<ReminderJob>()
        .WithIdentity(jobKey)
        .UsingJobData("chatId", chatId)
        .UsingJobData("text", text)

```

```

        .UsingJobData("jobKey", jobKey)
        .Build();

switch (reminderType)
{
    case "pre_reminder":
        var preTriggers = new List<ITrigger>
        {
            TriggerBuilder.Create()
                .WithIdentity($"{jobKey}_main")
                .StartAt(remindAt)
                .Build(),
            TriggerBuilder.Create()
                .WithIdentity($"{jobKey}_2days")
                .StartAt(remindAt.AddDays(-2))
                .Build(),
            TriggerBuilder.Create()
                .WithIdentity($"{jobKey}_1day")
                .StartAt(remindAt.AddDays(-1))
                .Build()
        };
        await _scheduler.ScheduleJob(job,
preTriggers, false);
        break;

    case "hourly_reminder":
        await _scheduler.ScheduleJob(job,
TriggerBuilder.Create()
            .WithIdentity($"{jobKey}_trigger")
            .StartNow()
            .WithSimpleSchedule(x =>
x.WithIntervalInHours(1).RepeatForever())
            .Build());
        break;

    case "daily_reminder":
        await _scheduler.ScheduleJob(job,
TriggerBuilder.Create()
            .WithIdentity($"{jobKey}_trigger")
            .StartNow()
            .WithSimpleSchedule(x =>
x.WithIntervalInHours(24).RepeatForever())
            .Build());
        break;

    case "weekly_reminder":
        await _scheduler.ScheduleJob(job,
TriggerBuilder.Create()
            .WithIdentity($"{jobKey}_trigger")
            .StartNow()

```

```

        .WithSimpleSchedule(x =>
x.WithIntervalInHours(168).RepeatForever())
        .Build());
        break;

        default: // regular
            await _scheduler.ScheduleJob(job,
TriggerBuilder.Create()
                .WithIdentity($"{jobKey}_trigger")
                .StartAt(remindAt)
                .Build());
            break;
    }

    await SendConfirmationMessage(chatId, text,
reminderType, remindAt);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Ошибка при создании напоминания:
{ex}");
        await _botClient.SendMessage(chatId, $"❌ Ошибка:
{ex.Message}");
    }

    private async Task SendConfirmationMessage(long chatId,
string text, string reminderType, DateTime remindAt)
    {
        var response = reminderType switch
        {
            "hourly_reminder" => $"✅ Ежечасное напоминание
создано: {text}",
            "daily_reminder" => $"✅ Ежедневное напоминание
создано: {text}",
            "weekly_reminder" => $"✅ Еженедельное напоминание
создано: {text}",
            "pre_reminder" => $"✅ Напоминание с уведомлениями
создано: {text} на {remindAt:dd MMMM HH:mm}",
            _ => $"✅ Напоминание создано: {text} на
{remindAt:dd MMMM HH:mm}"
        };

        await _botClient.SendMessage(chatId, response);
    }

    private bool TryParseRussianDateTime(string dateString, out
DateTime result)
    {
        result = DateTime.MinValue;
    }

```

```

        try
        {
            var months = new Dictionary<string, int>
            {
                {"января", 1}, {"февраля", 2}, {"марта", 3},
{"апреля", 4},
                {"мая", 5}, {"июня", 6}, {"июля", 7},
{"августа", 8},
                {"сентября", 9}, {"октября", 10}, {"ноября",
11}, {"декабря", 12}
            };

            var parts = dateString.Split(' ');
            if (parts.Length != 3) return false;

            if (!int.TryParse(parts[0], out var day)) return
false;
            if (!months.TryGetValue(parts[1].ToLower(), out var
month)) return false;
            if (!TimeSpan.TryParse(parts[2], out var time))
return false;

            var now = DateTime.Now;
            result = new DateTime(now.Year, month, day,
time.Hours, time.Minutes, 0);

            if (result < now)
                result = result.AddYears(1);

            return true;
        }
        catch
        {
            return false;
        }
    }

    public async Task<List<Reminder>> GetRemindersAsync(long
chatId)
    {
        return await _db.GetRemindersAsync(chatId);
    }

    public async Task DeleteReminderAsync(string jobKey)
    {
        try
        {
            var jobKeyObj = new JobKey(jobKey);
            if (await _scheduler.CheckExists(jobKeyObj))
                await _scheduler.DeleteJob(jobKeyObj);
        }
    }

```

```

        await _db.DeleteReminderAsync(jobKey);
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Ошибка при удалении напоминания:
{ex}");
        throw;
    }
}

public async ValueTask DisposeAsync()
{
    if (_scheduler != null)
        await _scheduler.Shutdown(true);
}
}

```

					ПГУ1.090301.48.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

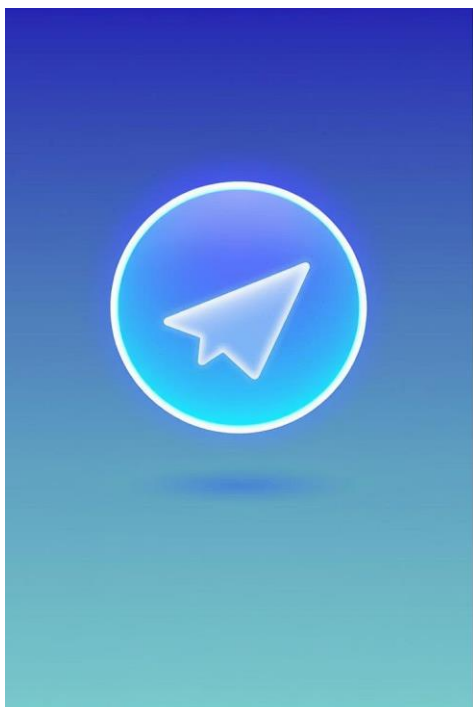
Приложение В

(Обязательное)

«Телеграм-бот “Напоминания”»

Презентация

					ПГУ1.090301.48.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		82



Telegram-бот «Напоминания»

Выполнил: студент группы 21ВВП2
Чабуев Ростислав Павлович

Научный руководитель:
К.т.н., доцент кафедры ВТ
Юрова Ольга Викторовна

Цель проекта



Разработка кроссплатформенного бота



Система напоминаний



Удобный пользовательский интерфейс



Надежное планирование задач

Основная цель проекта — предоставить пользователям надежный и удобный инструмент для управления своим расписанием, минимизируя риск забыть о важных встречах, задачах или событиях.

Изм.	Лист	№ докум.	Подпись	Дата

Актуальность



Популярность Telegram

Telegram является одним из ведущих мессенджеров с огромной пользовательской базой, что делает его идеальной платформой для разработки ботов.



Недостатки существующих решений

Многие стандартные приложения для напоминаний либо неудобны в использовании, либо перегружены избыточным функционалом, который отталкивает пользователей.



Кроссплатформенность и доступность

Telegram-бот предлагает простое, доступное и кроссплатформенное решение для управления напоминаниями, доступное на любом устройстве.

Актуальность проекта обусловлена возрастающей потребностью в эффективных инструментах для организации личного времени в современном мире. Мой бот способен решить эти проблемы, предлагая оптимальное сочетание функциональности и удобства.

Целевая аудитория



Студенты

Для эффективного управления учебным расписанием, сроками сдачи работ и подготовки к экзаменам.



Работники офисов

Для организации рабочих встреч, дедлайнов по проектам и управления задачами.



Люди с плотным графиком

Для тех, кто нуждается в постоянном контроле над своим временем и не хочет упустить важные события.



Пожилые пользователи

Для упрощения повседневных задач, таких как прием лекарств или важные звонки, без необходимости осваивать сложные приложения.

Мой Telegram-бот разработан для широкого круга пользователей, которые ценят свое время и ищут простое, но эффективное решение для управления напоминаниями. Он идеально подходит для тех, кто ведет активный образ жизни.



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.48.001 ПЗ

Лист

84

Анализ и аналоги

Изученные Telegram-боты

@yanapomnyu_bot: Простой бот для напоминаний.

@skeddybot: Бот с более продвинутыми функциями планирования.

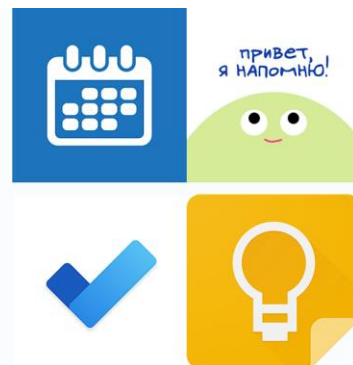
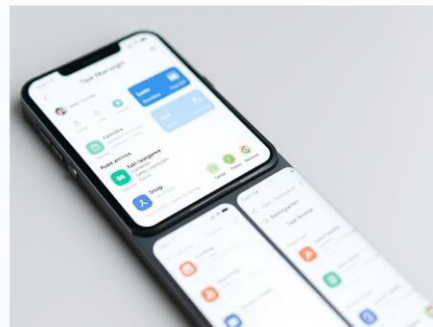
@to_list_do_bot: Бот для создания списков дел.

Приложения для заметок и задач

Google Keep: Облачное приложение для заметок и списков.

Microsoft To Do: Приложение для управления задачами.

В ходе анализа были изучены как существующие Telegram-боты, так и популярные приложения для управления задачами и напоминаниями. Это позволило выявить сильные и слабые стороны каждого решения, а также определить ключевые функциональные возможности, которые будут реализованы в моем боте для обеспечения максимальной эффективности и удобства пользователей.



Функциональность бота

Добавление напоминаний

Пользователи могут добавлять три типа напоминаний:

- Обычные: одноразовые уведомления.
- С предупреждением: с предварительным оповещением.
- Повторяющиеся: с заданным интервалом повторения.

Управление напоминаниями

Полный контроль над созданными напоминаниями:

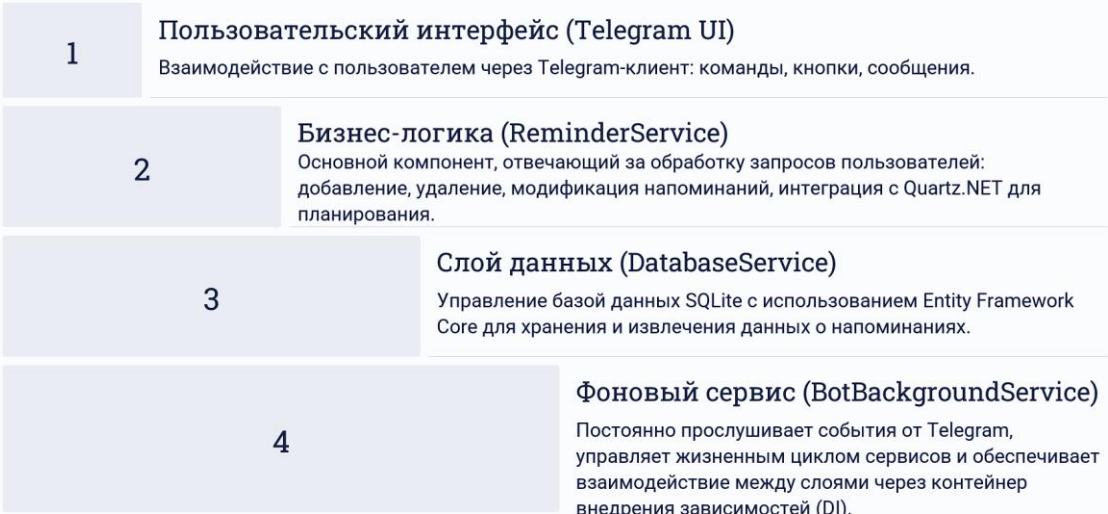
- Просмотр списка всех активных напоминаний.
- Удаление напоминаний, которые стали неактуальными.

Интерфейс и локализация

Удобный пользовательский опыт:

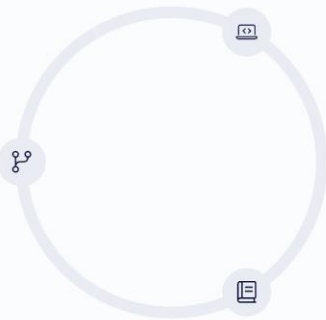
- Управление через текстовые команды и inline-кнопки.
- Полная поддержка русского языка.

Архитектура системы



Используемые технологии

Язык: C# (.NET 8.0)
Выбран благодаря своей производительности, надежности и широкой поддержке библиотек для создания современных приложений.



IDE: Visual Studio 2022
Интегрированная среда разработки, предоставляющая мощные инструменты для кодирования, отладки и тестирования.

Библиотеки

- **Telegram.Bot:** для взаимодействия с Telegram API.
- **Quartz.NET:** для планирования и выполнения задач по расписанию.
- **Microsoft.Data.Sqlite и Entity Framework Core:** для работы с базой данных SQLite.

Основные компоненты

BotBackgroundService	Отвечает за постоянное прослушивание сообщений от Telegram API и их первичную обработку.
ReminderService	Основная бизнес-логика: добавление, удаление, редактирование и планирование напоминаний.
DatabaseService	Обеспечивает взаимодействие с базой данных SQLite, сохранение и извлечение информации о напоминаниях.
ReminderJob	Класс, реализующий логику отправки уведомлений пользователям по расписанию, используя Quartz.NET.



Тестирование и интерфейс

Проведенные тестирования

Функциональное тестирование:

Проверка соответствия бота заявленным функциям, корректности добавления, изменения и удаления напоминаний, а также их своевременной отправки.

Интерфейсное тестирование:

Оценка удобства использования Telegram-интерфейса, интуитивности команд и работы inline-клавиатуры.

Тестирование и интерфейс

Начально окно приветствия:

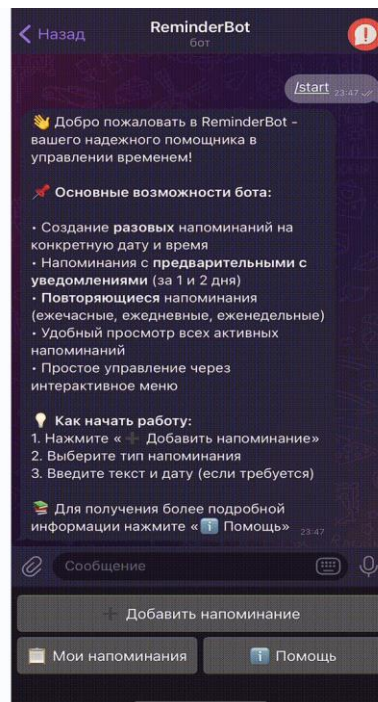
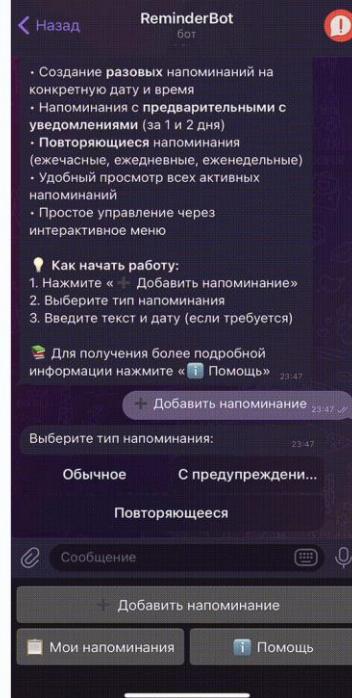
- Экран приветствия
- Основные возможности бота
- Краткое руководство по использованию



Тестирование и интерфейс

Добавление напоминаний:

- Выбор типа напоминаний
- Создание напоминания
- Уведомление от телеграм-бота



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.48.001 ПЗ

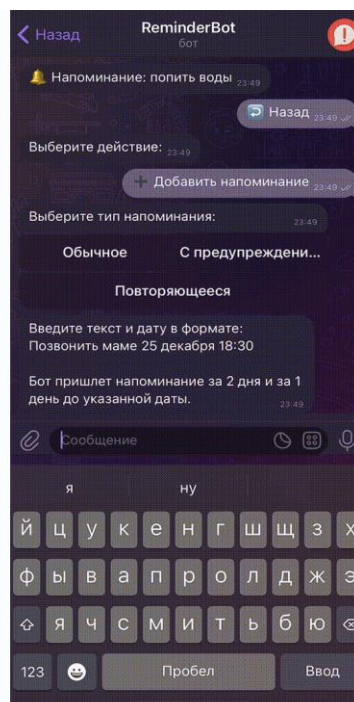
Лист

88

Тестирование и интерфейс

Добавление напоминания с предупреждением:

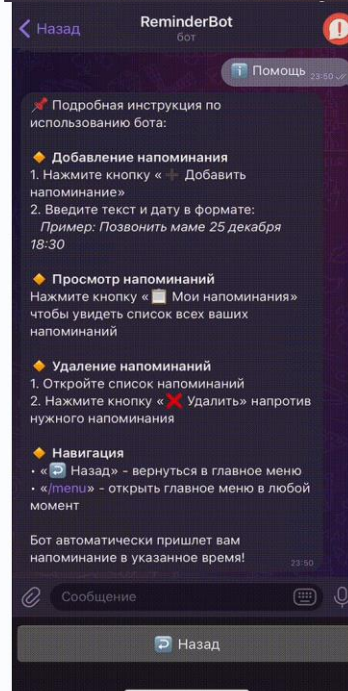
- Создание напоминания с предупреждением за 2 дня, до выполнения
- Просмотр созданных напоминаний
- Уведомление от телеграм-бота



Тестирование и интерфейс

Подробная инструкция по использованию бота:

- Добавление напоминания
- Просмотр напоминаний
- Удаление напоминаний



Вывод по тестированию

Проведенные тестирования подтвердили высокую надежность и удобство использования Telegram-бота «Напоминания». Интерфейс разработан таким образом, чтобы обеспечить максимальную простоту и ясность для любого пользователя, независимо от его опыта работы с мессенджерами и приложениями.

Возможность расширения:

- голосовые напоминания
- интеграция с календарями



ReminderBot

бот

Спасибо за внимание!

					ПГУ1.090301.48.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		