

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

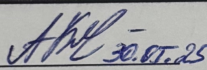
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

TELEGRAM-БОТ ДЛЯ ОРГАНИЗАЦИИ
УЧЕБНОГО ПРОЦЕССА

Студент

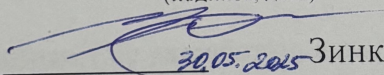
 30.05.2025

(подпись, дата)

Борисова Анастасия Витальевна

(ФИО полностью)

Руководитель

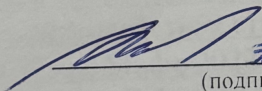
 30.05.2025

(подпись, дата)

Зинкин С.А.

(фамилия, инициалы)

Нормоконтролёр

 30.05.2025

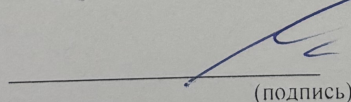
(подпись, дата)

Бычков А.С.

(фамилия, инициалы)

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

Заведующий кафедрой



(подпись)

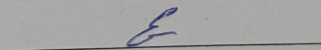
М.А. Митрохин

(фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от

11 июня 2025 г. № 26)

Секретарь ГЭК



(подпись)

Е.И. Гурин

(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
« 6 » 05 2025г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Борисова Анастасия Витальевна гр.21ВВП2 факультета вычислительной техники

2. Тема работы: Telegram-бот для организации учебного процесса
Тема утверждена приказом ПГУ № 609/0 от " 6 " 05 20 25 г.

3. Руководитель бакалаврской работы: д.т.н., профессор кафедры ВТ Зинкин С.А.

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

Разработать ассистента-помощника учебного процесса, представленную в виде Telegram-бота для упрощения взаимодействия студентов и преподавателей. Язык программирования C#, среда разработки Visual Studio 2022. Программа должна обладать интуитивно понятным интерфейсом в мессенджере Telegram и использовать следующие технологии:

1. Telegram

2. SQL

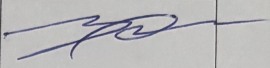
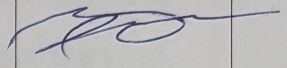
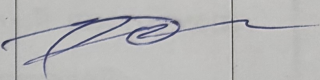
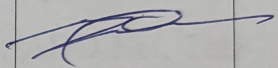
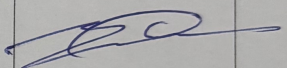
3. TelegramBotAPI

4. Net9

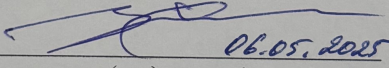
5. Перечень подлежащих разработке вопросов

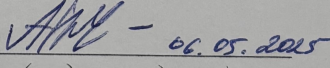
Требуется произвести анализ предметных областей: изучить существующие Телеграмм-боты в образовательной среде, выделить их достоинства и недостатки. А также анализ технических требований к системе, включая функциональные и нефункциональные аспекты. Выбрать язык программирования, среду разработки, СУБД.

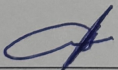
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	06.05.2025 – 08.05.2025	
2	Анализ требований	32	09.05.2025 – 12.05.2025	
3	Выбор технологий, применяемых при проектировании и разработке	40	13.04.2025 – 15.05.2025	
4	Разработка основной части программы	70	15.05.2025 – 24.05.2025	
5	Оформление пояснительной записки	54	25.05.2025 – 30.05.2025	

Дата выдачи задания " 6 " 05 2025 г.

Руководитель бакалаврской работы  06.05.2025
(подпись, дата)

Задание к исполнению принял студент  - 06.05.2025
(подпись, дата)

Работу к защите допустить
Декан факультета  05.06.2025
(подпись, дата)

Реферат

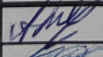
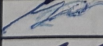
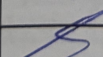
Бакалаврская работа содержит 180 страницы, 36 рисунков, 13 таблиц, 18 источников, 3 приложения.

ЧАТ-БОТ, MVS IDE, TELEGRAM BOT API, БАЗА ДАННЫХ, КОМАНДЫ, SQL SERVER

Цель работы – разработать ассистент-помощник в учебном процессе для преподавателей и студентов, для упрощения проведения учебных занятий.

В результате проделанной работы было разработано программное обеспечение, позволяющее обеспечить функциональный и надежный инструмент, который гарантирует комфортную работу студентов и преподавателей.

В ходе разработки ассистента-помощника для организации учебного процесса, использовались следующие технологии: с#, .Net 9, SQL, ADO.Net, Telegram.Bot API.

					ПГУ 1.090301.26.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Борисова А.В.		30.05.25			
Провер.		Зинкин С.А.		30.05.25			
Н. Контр		Бычков А.С.		30.05.25			
Утверд.		Митрохин М.А.		30.05.25			
					Лит.	Лист	Листов
						2	180
					ФВТ, каф. ВТ гр. 21ВВП2		

Содержание

Перечень сокращений и обозначений.....	4
Введение.....	5
1 Анализ предметной области	6
1.1 Анализ актуальности темы.....	6
1.2 Анализ существующих аналогов.....	7
1.3 Анализ функциональных требований	8
1.4 Анализ нефункциональных требований	12
2 Анализ архитектуры и структуры чат-бота.....	13
2.1 Описание архитектуры проекта.....	22
2.2 Выбор инструментов разработки	23
2.2.1 Выбор мессенджера для реализации чат-бота	23
2.2.2 Выбор языка программирования.....	27
2.2.3. Выбор СУБД.....	31
2.3 Выбранные технологии	32
3 Проектирование и разработка.....	34
3.1 Регистрация чат-бота	34
3.2 Проектирование БД.....	36
3.3 Описание программы.....	42
4 Тестирование чат-бота.....	54
Заключение	66
Список использованных источников	68
Приложение А – UML-диаграммы.....	70
Приложение Б – Листинг программы	75
Приложение В – Презентация.....	175

Перечень сокращений и обозначений

В настоящей работе применяются следующие сокращения и обозначения

БД – база данных

ИИ – искусственный интеллект

ПК – персональный компьютер

СНГ – содружество независимых государств

СУБД – система управления базами данных

ЯП – язык программирования

ACID (Atomicity, Consistency, Isolation, Durability) – требования к транзакционной системе, обеспечивающие наиболее надёжную и предсказуемую её работу

API (Application Programming Interface) – описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой

IDE (Integrated development environment) – интегрированная среда разработки

ID (Identifier) – идентификатор объекта

JSON (JavaScript Object Notation) – текстовый формат обмена данными

SQL (Structured Query Language) – язык структурированных запросов

UML (Unified Modeling Language) – унифицированный язык моделирования

Введение

В современном мире цифровые технологии играют ключевую роль в образовательном процессе. Автоматизация рутинных задач, упрощение коммуникации между студентами и преподавателями, а также оперативное получение информации являются важными аспектами эффективного обучения.

Данная выпускная квалификационная работа посвящена разработке Telegram-бота, предназначенного для автоматического оповещения студентов, предоставления информации о расписании занятий, дедлайнах, экзаменах и контрольных работах. Кроме того, бот предоставляет пользователям доступ к учебным материалам, контактными данным преподавателей и другим необходимым ресурсам.

Основной целью разработки является создание удобного, функционального и надежного инструмента, который обеспечит комфортную работу студентов и преподавателей. Важной особенностью проекта является поддержка различных ролей пользователей, таких как студент, преподаватель и администратор, каждая из которых обладает своим уровнем доступа и функциональными возможностями.

В рамках данной выпускной работы будут рассмотрены основные этапы проектирования и разработки Telegram-бота, включая выбор архитектуры, реализацию системы авторизации, организацию базы данных, обработку пользовательских запросов и обеспечение безопасности системы. Также будет проведено тестирование бота с целью выявления возможных ошибок и улучшения его функционала.

Разработка данного программного продукта позволит повысить эффективность образовательного процесса, снизить нагрузку на преподавателей за счет автоматизации некоторых процессов и обеспечить студентов актуальной информацией в удобном формате.

1 Анализ предметной области

Первым пунктом жизненного цикла разработки любой программы, является анализ предметной области и составление требований к продукту. Этот этап включает анализ и выбор функциональных и нефункциональных требований, необходимых для достижения целей проекта, а также обоснование актуальности разработки.

1.1 Анализ актуальности темы

В современных учебных заведениях взаимодействие между студентами и преподавателями может осуществляться через различные средства коммуникации, каждое из которых имеет свои преимущества и недостатки.

Исторически первым и самым распространенным является личное общение. При очной встрече, на парах или консультациях, у студентов всегда имеется возможность задать интересующие вопросы преподавателю. В случае возникновения вопросов вне университета, многие учащиеся обращаются ко второму по популярности способу - электронной почте. Однако, эффективность данного метода намного ниже. Виной тому большой объем входящих писем, где сообщение студента может легко затеряться или даже «улететь в спам», отсутствие гарантии, что письмо прочитано, а также возможные задержки в ответах. Оба эти способа не подходят для быстрого получения информации и решения возникающих вопросов.

Некоторые ВУЗы внедряют специализированные платформы для размещения учебных материалов и контактной информации, что значительно упрощает процесс обучения. Но многие процессы, такие как рассылка уведомлений о дедлайнах или оценках, по-прежнему требуют ручного вмешательства, что увеличивает нагрузку на преподавателей и административный персонал.

Разработка Telegram-бота для организации учебного процесса направлена на решение указанных проблем. Предлагаемое решение автоматизирует рассылку уведомлений о занятиях, экзаменах, заданиях, это позволит снизить нагрузку на преподавателей и администрацию. Чат-бот обеспечит удобный доступ к учебным материалам, расписанию и контактной информации как для обучающихся, так и для преподавателей. Кроме того, администраторы смогут эффективно управлять расписанием, домашними заданиями и системой оценивания.

1.2 Анализ существующих аналогов

Обзор аналогов является важной частью разработки программы, так как позволяет выявить преимущества и недостатки существующих решений, а также определить требования к разрабатываемому приложению.

Несмотря на то, что на сегодняшний день существует множество аналогов Телеграмм-ботов в образовательной среде, функционал большинства из них ограничивается новостным форумом. Например, чат-боты «ПензГТУ» (@PenzGTU58), «ННГУ» (@lobachevsky_university) и «РГУ» (@rgu_esenina) имеют только информационную рассылку о событиях и мероприятиях университета. Чат-боты для абитуриентов располагают более обширным спектром возможностей. Так, в «НИУ ВШЭ» (@media_hse_bot) и в «РАНХиГС» (@abiturientuiu_Bot), представлен следующий функционал:

Функционал «НИУ ВШЭ» (@media_hse_bot):

- регистрация;
- выбор направлений обучения;
- контактная информация.

Функционал «РАНХиГС» (@abiturientuiu_Bot):

- выбор направлений обучения;
- возможность оценить шансы на поступление;

- вывод всей необходимой информации о поступлении, общежитии и баллах;
- контактная информация.

В таблице 1 представлено сравнение существующих аналогов.

Таблица 1 – Сравнение аналогов

Название ВУЗа	Ссылка на чат-бот	Рассылка уведомлений	Регистрация	Отметка посещаемости	Выставление оценок	Контактная информация
ПензГТУ	@PenzGTU58	+	-	-	-	-
ННГУ	@lobachevsky_university	+	-	-	-	-
РГУ	@rgu_senina	+	-	-	-	-
НИУ ВШЭ	@media_hse_bot	-	+	-	-	+
РАНХиГС	@abiturientuiu_Bot	-	-	-	-	+

В результате анализа не удалось выявить полного аналога с заданным функционалом.

1.3 Анализ функциональных требований

Телеграмм-бот для организации учебного процесса – является программным обеспечением, которое направлено на автоматизацию некоторых процессов для студентов и преподавателей.

Telegram-бот должен обеспечивать следующие функциональные возможности:

- возможность регистрации в системе учебного заведения;
- авторизация в системе, для обеспечения большего уровня защиты пользовательских данных;
- осуществлять запланированную рассылка уведомлений студентам, например, об расписании занятий, экзаменах и дедлайнах;
- предоставление пользователям информации о программе курса, учебных материалах и контактных данных преподавателей;
- разграничение уровней доступа для ролей: студент, преподаватель, администратор;
- интерактивный интерфейс с поддержкой кнопок и команд;
- обработка пользовательских запросов в многопоточной среде.

Для представления и отображения всех функциональных возможностей разрабатываемой системы, были созданы UML-диаграммы вариантов использования. На рисунке 1 - UML-диаграмма вариантов использования для преподавателя, на рисунке 2 для студента и на рисунке 3 для администратора.

Функциональные требования для функционала преподавателя:

- авторизация в системе учебного заведения чат-бота;
- просмотр личной информации;
- выбор занятия из расписания;
- возможность отметить присутствующих студентов;
- возможность выставить оценки за занятия студентам;
- просмотр домашних заданий;
- возможность добавлять домашние задания;
- возможность просмотра и выставления оценок за ответы на домашние задания студентов;
- возможность рассылки уведомлений студентам.



Рисунок 1 – UML-диаграмма вариантов использования для преподавателя

Функциональные требования к функционалу студента:

- авторизация в системе учебного заведения чат-бота;
- просмотр личной информации;
- выбор занятия из расписания;
- просмотр полученных оценок;
- просмотр домашних заданий;
- возможность добавить ответ на домашнее задание.

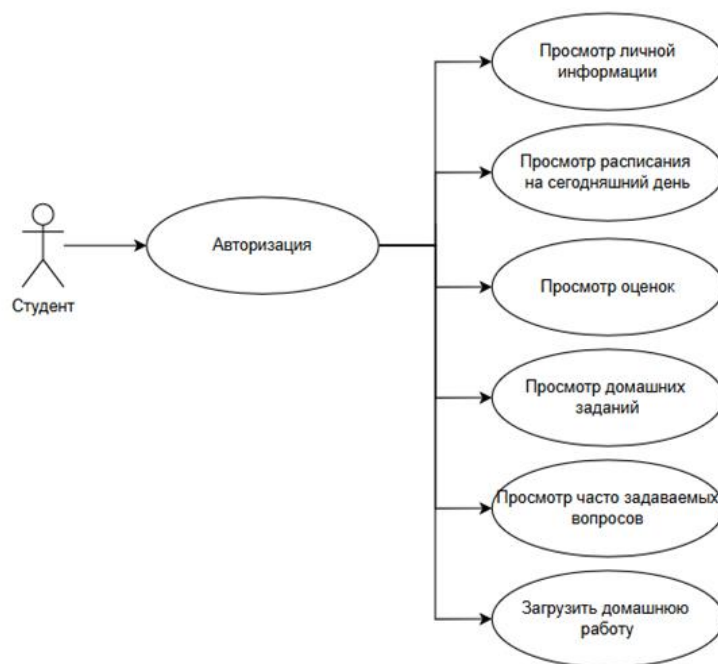


Рисунок 2 – UML-диаграмма вариантов использования для студента

Функциональные требования к функционалу администратора:

- авторизация в системе учебного заведения чат-бота;
- возможность войти под ролью студента для тестов функционала;
- возможность войти под ролью преподавателя для тестов функционала;
- просмотр всех таблиц базы данных;
- редактирование данных всех таблиц;
- составления расписания.

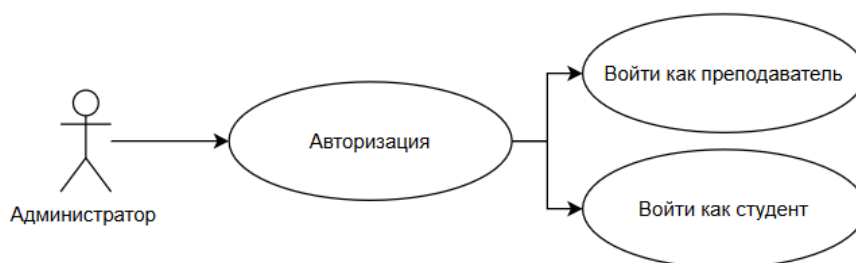


Рисунок 3 – UML-диаграмма вариантов использования для администратора

1.4 Анализ нефункциональных требований

Для функционирования данной системы требуется два устройства: клиент и сервер. Клиент – любое устройство с доступом к интернету, поддерживающее мессенджер Telegram нативно или через веб-браузер. Сервер – устройство с операционной системой Windows, Linux или MacOS с установленным необходимым программным обеспечением и доступом к интернету.

2 Анализ архитектуры и структуры чат-бота

Чат-боты могут быть классифицированы по нескольким критериям:

- по уровню сложности:

1) простые (правила на основе команд) – реагируют на заранее заданные команды, например, /help, /faq;

2) интерактивные (меню-боты) – работают через кнопки и меню без необходимости ввода команд;

3) интеллектуальные (ИИ-боты) – используют машинное обучение и обработку естественного языка (NLP) для общения с пользователями;

– по области применения:

1) информационные – предоставляют справочную информацию, уведомления;

2) помощники – помогают пользователям выполнять задачи, например, записываться на занятия;

3) обучающие – предоставляют образовательный контент, тестирование и задания;

– по способу взаимодействия:

1) текстовые – обмениваются текстовыми сообщениями;

2) голосовые – используют технологии синтеза речи (например, GoogleSpeech-to-Text);

3) гибридные – поддерживают и текстовое, и голосовое взаимодействие.

Разрабатываемый в рамках выпускного проекта чат-бот относится к информационно-обучающим, с меню-навигацией и текстовыми командами.

Чат-бот функционирует по заранее заданному алгоритму, обеспечивающему правильное взаимодействие пользователей с системой. Архитектуру проектируемого чат-бота можно охарактеризовать следующими компонентами:

– клиент;

- сервер;
- база данных.

Клиентская часть (frontend) — это часть программного обеспечения, которая взаимодействует с пользователем. Она отвечает за отображение интерфейса, обработку пользовательского ввода и передачу данных на сервер [1].

Основные функции клиентской части:

- взаимодействие с пользователем – клиентская часть предоставляет удобный и интуитивно понятный интерфейс;
- обработка ввода – анализирует введенные пользователем данные перед отправкой на сервер;
- запросы к серверу – отправляет данные на сервер с помощью API (например, REST или WebSocket);
- получение и отображение данных – принимает ответы сервера и обновляет интерфейс.

Серверная часть (backend) – это внутренняя часть системы, которая выполняет бизнес-логику, обрабатывает запросы от клиентов и взаимодействует с базой данных [1].

Основные функции серверной части:

- аутентификация и авторизация – проверка пользователей и контроль доступа к ресурсам;
- обработка логики приложения – выполнение бизнес-правил, расчетов и обработка запросов;
- взаимодействие с базой данных – хранение, извлечение, обновление и удаление данных;
- масштабируемость и отказоустойчивость – обеспечение надежной работы системы при высокой нагрузке;
- API и обмен данными – предоставление данных клиентской части через REST API, GraphQL или WebSocket.

База данных – это структурированное хранилище информации, предназначенное для эффективного хранения, поиска и управления данными [1].

Основные функции базы данных:

- хранение данных – систематизация информации в таблицах, документах или графах;
- обеспечение целостности – предотвращение дублирования, поддержка связей между записями;
- быстрый доступ к данным – оптимизированные запросы для извлечения нужной информации;
- безопасность – контроль доступа и защита данных от несанкционированного изменения.

Типы баз данных:

- реляционные (SQL) – данные хранятся в таблицах, связаны по ключам (PostgreSQL, MySQL, SQL Server);
- документо-ориентированные (NoSQL) – данные хранятся в JSON-документах (MongoDB, Firebase);
- графовые – используются в социальных сетях (Neo4j);
- ключ-значение (Key-Value) – применяются для кеширования (Redis).

База данных в Telegram-боте

Для хранения всей информации используется реляционная база данных SQL.

Структура взаимодействия СУБД, клиентской, серверной частей представлена на рисунке 4.

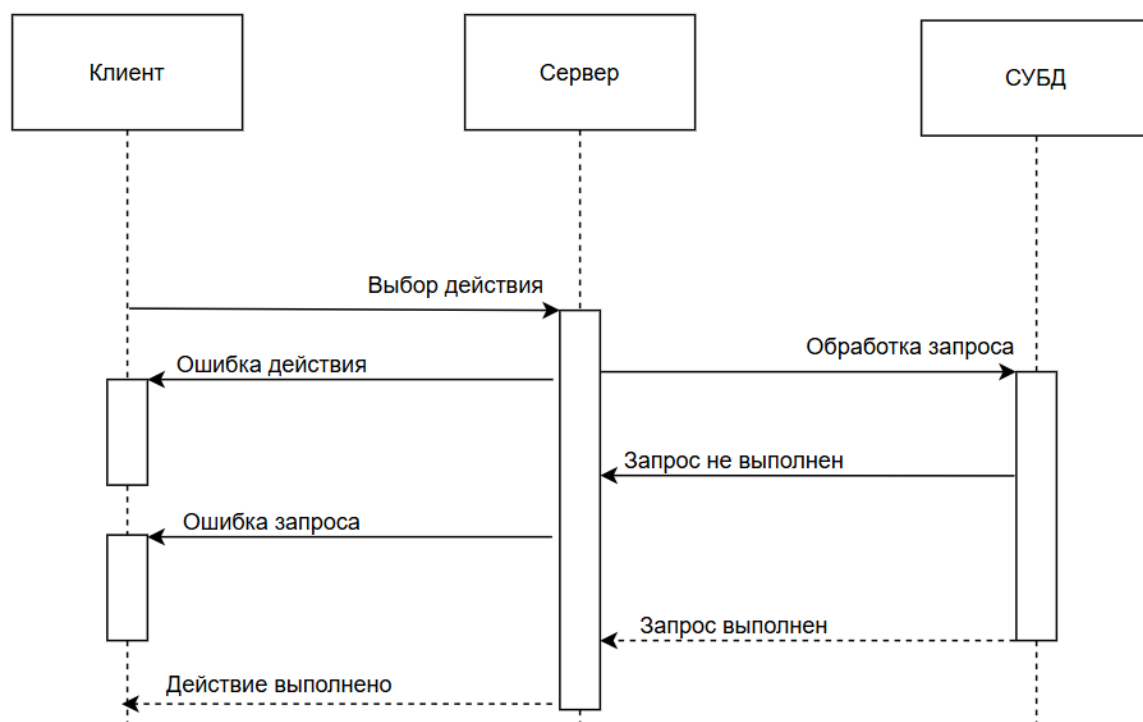


Рисунок 4 – UML-диаграмма последовательности

На рисунках 5-9 представлены диаграммы классов разработанного чат-бота.

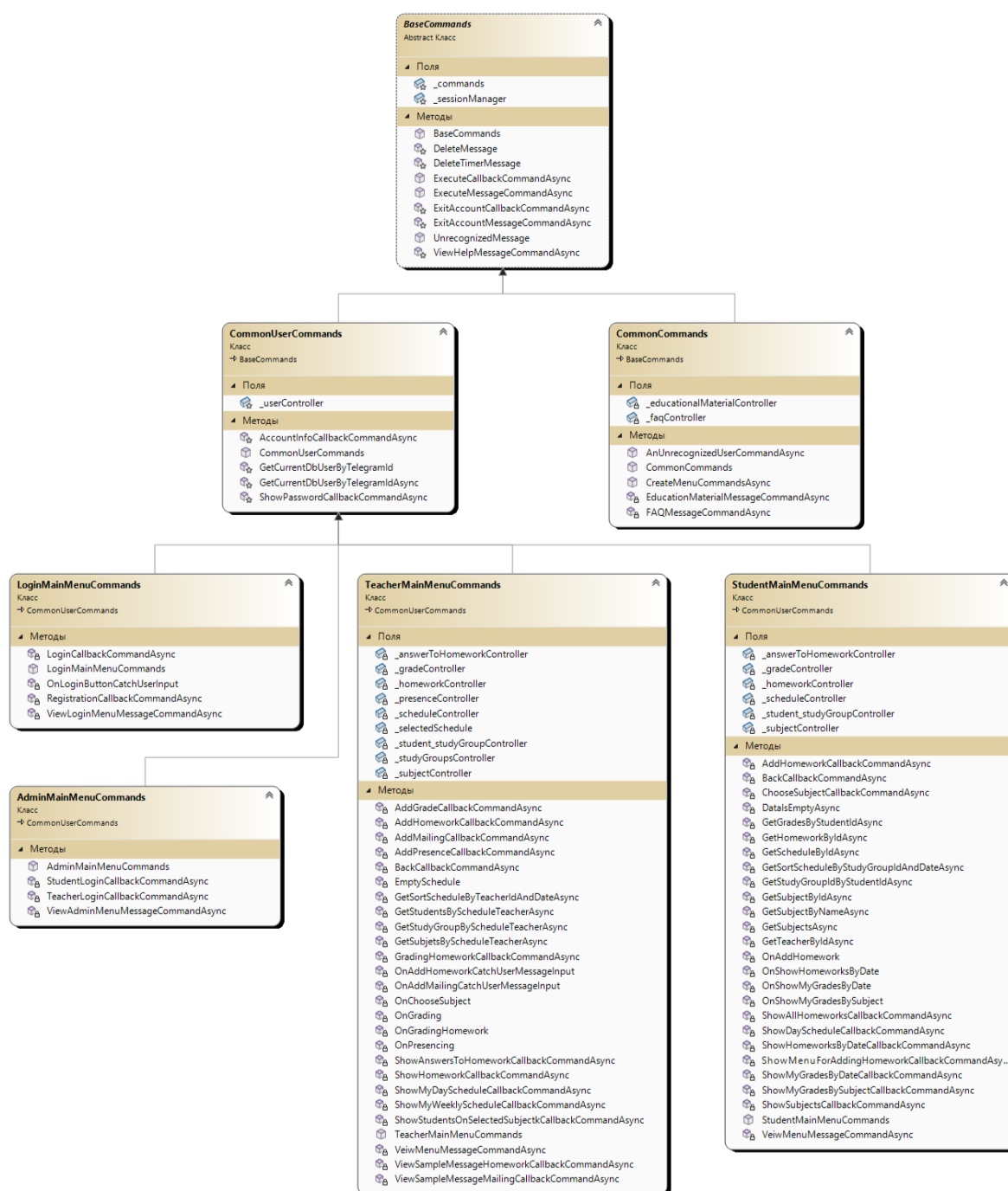


Рисунок 5 – Диаграмма классов команд



Рисунок 6 – Диаграмма классов работы с базой данных

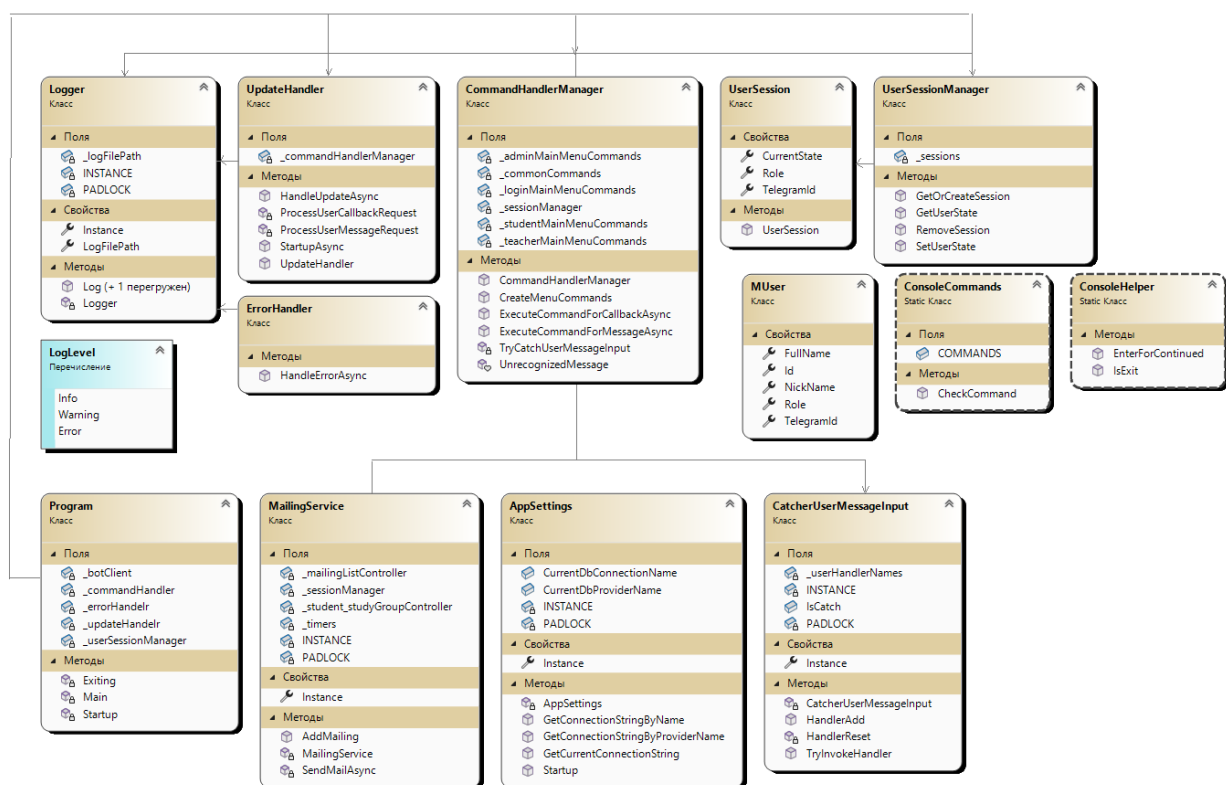


Рисунок 7 – Диаграмма основных классов

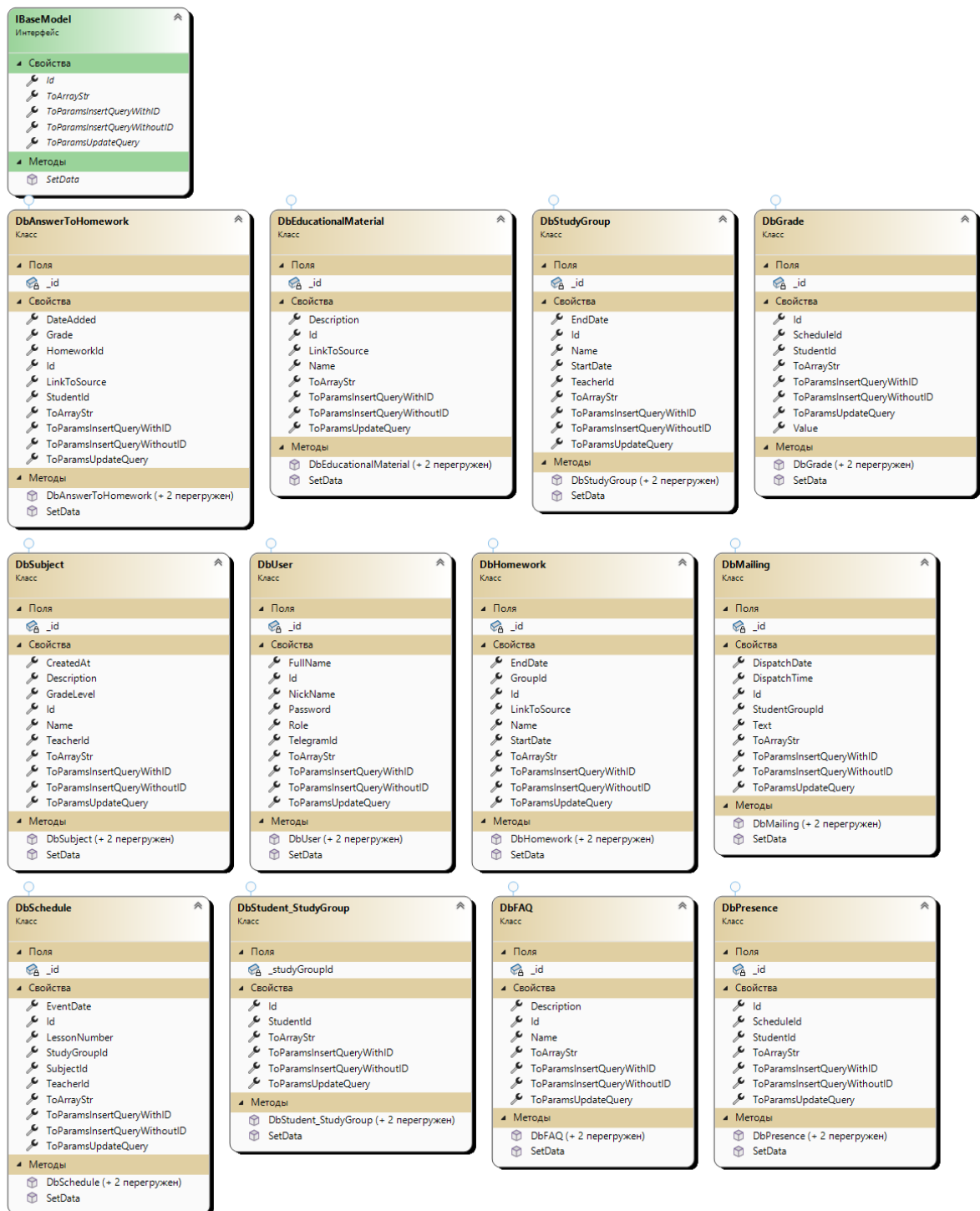


Рисунок 8 – Диаграмма классов базы данных

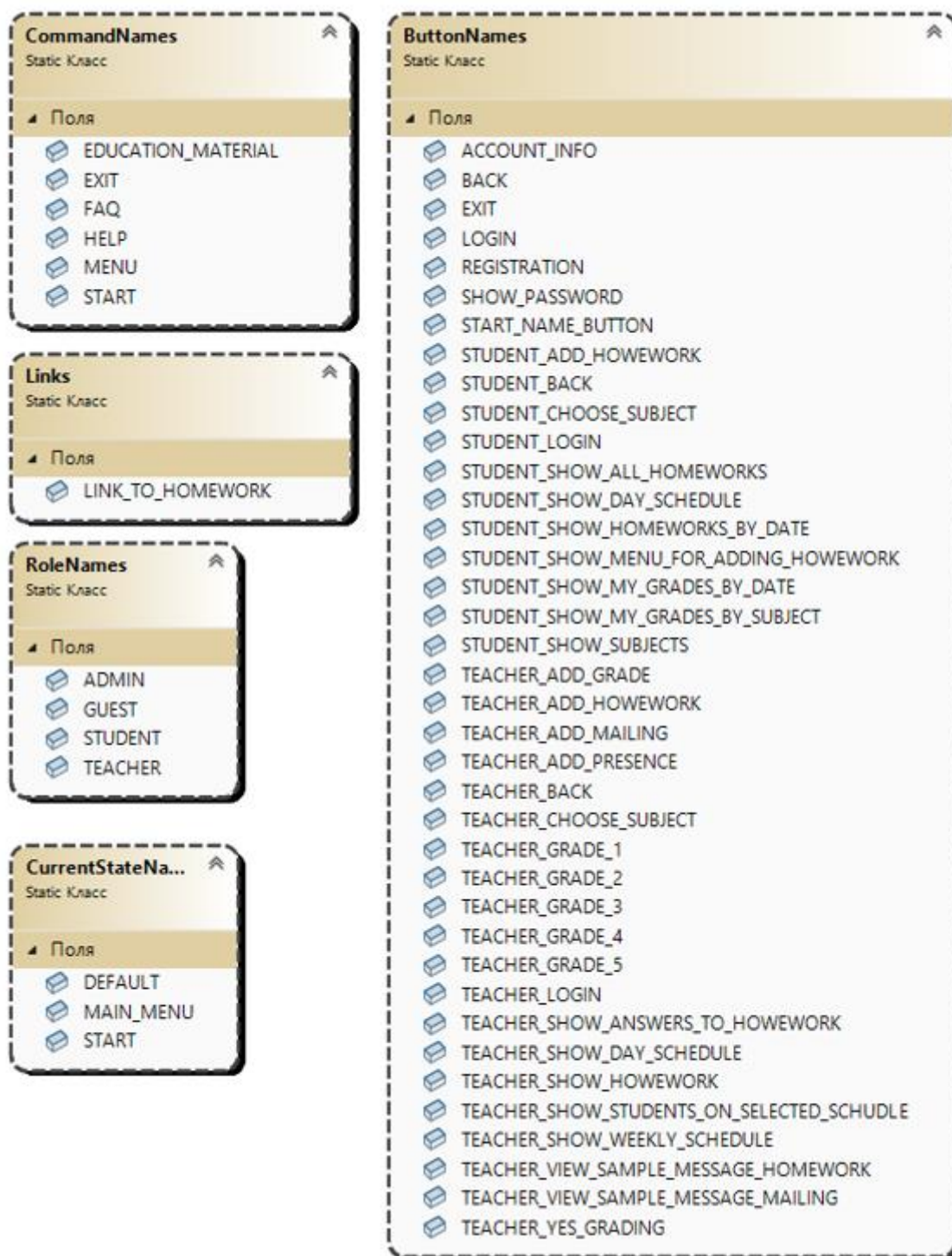


Рисунок 9 – Диаграмма классов, содержащих константы

Структура разработанного чат-бота включает следующие основные модули:

– модуль авторизации – проверяет учетные данные пользователей;

- модуль работы с данными – предоставляет доступ к данным, например, к расписанию, предметам, домашним заданиям и подобным;
- модуль рассылки уведомлений – отправляет напоминания пользователям;
- модуль управления пользователями – назначает и проверяет роли (студент, преподаватель, администратор);
- модуль хранения данных – использует базу данных для хранения информации;
- модуль обработки ввода – проверяет выбранные команды или введенный текст и выполняет требуемые действия.

Взаимодействие пользователя с ботом строится через команды или кнопки, обрабатываемые серверной частью.

2.1 Описание архитектуры проекта

Архитектура основана на многоуровневом подходе:

- уровень взаимодействия с Telegram API – принимает и отправляет сообщения;
- уровень логики – обработка команд, авторизация, обработка запросов;
- уровень хранения данных – база данных пользователей, расписания и учебных материалов;
- используется шаблон MVC (Model-View-Controller) для разделения логики бота. Это способ организации кода, который выделяет блоки, отвечающие за решение разных задач. Модель – компонент, который отвечает за данные и структуру приложения (DbUser – представляет модель пользователя (студента, преподавателя, администратора), DbSchedule – модель расписания занятий, DbSubject – модель учебного предмета и тд.). Представление – этот компонент отвечает за взаимодействие с пользователем (UpdateHandler – обрабатывает входящие сообщения и callback-запросы, CommandHandlerManager – управляет выполнением команд в зависимости от

роли пользователя (студент, преподаватель, администратор), CommonCommands, LoginMainMenuCommands, AdminMainMenuCommands, StudentMainMenuCommands, TeacherMainMenuCommands – содержат логику формирования ответных сообщений и кнопок для разных ролей). То есть код компонента view определяет внешний вид приложения и способы его использования. Контроллер отвечает за связь между model и view. (UserSessionManager – управляет сессиями пользователей (хранение текущего состояния и роли), MailingService – реализует логику рассылки уведомлений и тд.). Код компонента controller определяет, как сайт реагирует на действия пользователя.

Компоненты архитектуры:

- Telegram Bot API – интерфейс для взаимодействия с пользователями;
- Web API (.NET 9) – серверная часть, отвечающая за обработку запросов;
- база данных (SQL Server) – хранение информации;
- очереди запросов (Task, PoolThread, async) – обработка фоновых задач.

В процессе пользователь взаимодействует с чат-ботом через мессенджер Telegram. Интерфейс чат-бота реализован с помощью inline-кнопок, ввода текста и текстовых команд. Серверная часть обрабатывает выбранные действия пользователя, введенный текст или введенные текстовые команды и на базе проанализированного ввода запускает соответствующее действие.

2.2 Выбор инструментов разработки

2.2.1 Выбор мессенджера для реализации чат-бота

Чат-боты широко используются в различных мессенджерах для автоматизации общения с пользователями, предоставления услуг, обработки запросов и многого другого. Рассмотрим самые популярные мессенджеры, которые поддерживают разработку чат-ботов, а также их основные преимущества и недостатки.

Telegram — один из самых популярных мессенджеров с открытым API и удобными инструментами для создания ботов. Поддерживает Bot API, который позволяет легко разрабатывать чат-ботов на различных языках программирования (Python, C#, JavaScript и др.) [2].

Преимущества:

- простой API – удобный и хорошо документированный Telegram Bot API;
- отсутствие ограничений на число пользователей – можно обслуживать тысячи пользователей одновременно;
- поддержка Webhook и LongPolling – гибкие механизмы получения сообщений;
- гибкие интерфейсы – поддержка inline-кнопок, встроенных клавиатур, файлов, видео, голосовых сообщений и анимаций;
- без рекламы и спама – Telegram не монетизирует своих пользователей, нет навязчивой рекламы;
- бесплатность – полностью бесплатное API без ограничений.

Недостатки: Не все пользователи активно используют Telegram – особенно в странах, где доминируют другие мессенджеры (например, WhatsApp в Европе и Латинской Америке).

WhatsApp — один из самых популярных мессенджеров в мире, принадлежащий Meta (Facebook). Разработка ботов осуществляется через WhatsAppBusiness API, который предоставляет мощные возможности, но требует одобрения со стороны Meta [3].

Преимущества:

- огромная аудитория – миллиарды активных пользователей;
- высокая вовлеченность – пользователи часто проверяют WhatsApp, что повышает эффективность бота;
- поддержка мультимедиа – отправка изображений, аудио, видео и файлов;

– интеграция с CRM и e-commerce – удобна для бизнеса.

Недостатки:

– платный API – WhatsAppBusiness API требует оплаты за отправленные сообщения;

– требует одобрения – чтобы получить доступ к API, необходимо зарегистрировать бизнес и пройти верификацию;

– ограниченный функционал в сравнении с Telegram – меньше возможностей для кастомизации интерфейса.

Facebook Messenger — мессенджер от Meta, который поддерживает создание чат-ботов через Messenger API. Боты в Messenger активно используются бизнесом для автоматизации поддержки клиентов [4].

Преимущества:

– интеграция с Facebook – удобен для работы с бизнес-страницами Facebook;

– поддержка мультимедиа – отправка изображений, видео, аудио и файлов;

– возможность встроенной рекламы – можно продвигать чат-ботов через FacebookAds;

– платежи через Messenger – можно организовать продажу товаров и услуг.

Недостатки:

– сложность одобрения – необходимо пройти модерацию для работы с API;

– ограничения по рассылке – нельзя отправлять рекламные сообщения спустя 24 часа после последнего взаимодействия с пользователем;

– не везде популярен – FacebookMessenger популярен в США и Европе, но в некоторых странах уступает WhatsApp и Telegram.

Viber — популярный мессенджер, особенно востребованный в Восточной Европе и Азии. Поддерживает ViberBot API для создания чат-ботов [5].

Преимущества:

- большая пользовательская база в СНГ и Азии;
- поддержка RichMedia – кнопки, изображения, видео, стикеры;
- возможность создания публичных аккаунтов – чат-бот может взаимодействовать с широкой аудиторией.

Недостатки:

- ограничения на рассылки – массовые сообщения возможны только платно;
 - менее развитая экосистема API – по сравнению с Telegram и WhatsApp.
- Slack — корпоративный мессенджер, который активно используется для работы в командах. Поддерживает создание ботов через Slack API [6].

Преимущества:

- популярен в корпоративной среде – удобен для автоматизации рабочих процессов;
- интеграция с сервисами – поддержка интеграции с Trello, Jira, GitHub и другими инструментами;
- поддержка команд и интерактивных элементов – кнопки, опросы, вложения.

Недостатки:

- небольшая аудитория среди обычных пользователей;
- частично платные функции – для крупных организаций API может быть платным.

Discord — платформа для голосового и текстового общения, популярная среди геймеров и сообществ. Боты создаются через Discord API [7].

Преимущества:

- развитая система ролей – удобна для управления сообществами;

- поддержка голосовых команд – можно управлять ботом голосом;
- гибкий API – широкие возможности кастомизации.

Недостатки:

- низкая популярность вне игрового сегмента;
- требует сервер для работы – не так удобен, как Telegram или WhatsApp.

Выбор мессенджера для чат-бота зависит от его целей и аудитории.

- если нужен бесплатный, удобный и мощный мессенджер – лучше всего Telegram;
- для работы с бизнесом – WhatsApp Business API или Facebook Messenger;
- если аудитория в СНГ – Viber;
- для корпоративного сегмента – Slack;
- для игровых сообществ – Discord.

Telegram остаётся одним из лучших вариантов для создания ботов из-за бесплатного API, удобства разработки и гибкости интерфейсов.

2.2.2 Выбор языка программирования

Telegram предоставляет Bot API, который позволяет разрабатывать чат-ботов на различных языках программирования. Выбор языка зависит от требований к производительности, удобства разработки, доступности библиотек и опыта разработчика.

Python – один из самых популярных языков для разработки ботов благодаря простоте синтаксиса и обширной экосистеме библиотек [8].

Достоинства:

- легкость освоения – удобный и читаемый синтаксис;
- большое количество библиотек – pyTelegramBotAPI, aiogram, python-telegram-bot;
- отличная работа с асинхронностью – aiogram поддерживает async/await, что повышает производительность;

– быстрая разработка – подходит для прототипирования и быстрого запуска бота.

Недостатки:

– не самый быстрый язык – уступает C# и Go по производительности;
– может потреблять больше ресурсов – особенно при высоких нагрузках.

Когда использовать: для небольших ботов, MVP, автоматизации, обработки сообщений и интеграции с AI (ChatGPT, нейросети).

JavaScript (в частности, Node.js) – мощный язык для разработки серверных приложений, поддерживающий асинхронную обработку запросов[9].

Достоинства:

– асинхронность по умолчанию – отлично обрабатывает большое количество пользователей;
– популярность среди веб-разработчиков – удобная интеграция с веб-приложениями;
– богатый выбор библиотек – node-telegram-bot-api, telegraf.js;
– подходит для разработки Webhook-серверов – можно легко развернуть на Express.js.

Недостатки:

– не подходит для вычислений – низкая производительность в математических операциях;
– не всегда стабилен – при плохой обработке ошибок могут возникать утечки памяти.

Когда использовать: для веб-приложений, интеграции с базами данных, API, работы с асинхронными ботами.

C# – мощный объектно-ориентированный язык с высокой производительностью, подходящий для корпоративных решений и сложных телеграм-ботов [10, 11, 12].

Достоинства:

- высокая производительность – компилируемый язык, работает быстрее интерпретируемых языков (Python, JS);
- отличная поддержка многопоточности – Task, async/await;
- гибкая интеграция с базами данных – через EntityFramework, Dapper;
- поддержка облачных решений – легко развернуть в Azure или AWS.

Недостатки:

- более сложная разработка – требует больше кода, чем Python;
- меньшее количество готовых библиотек – есть Telegram.Bot, но сообщество меньше, чем у Python.

Когда использовать: для сложных корпоративных решений, API, высоконагруженных сервисов.

Java – надежный язык, популярный в корпоративной среде, подходит для масштабных проектов [14].

Достоинства:

- кроссплатформенность – работает на любых системах (Windows, Linux, Mac);
- высокая производительность – быстрее Python и JS;
- безопасность – строгая типизация, управление памятью.

Недостатки:

- сложность разработки – требует больше кода;
- медленнее разворачивается – нужно поднимать JVM.

Когда использовать: для корпоративных приложений, высоконагруженных ботов.

Go – язык с высокой производительностью, разработанный для многопоточных приложений [15].

Достоинства:

- очень высокая скорость выполнения – почти как у C++;

– отличная работа с многопоточностью – goroutines позволяют эффективно обрабатывать запросы;

– легкость деплоя – можно компилировать в единый бинарный файл.

Недостатки:

– меньше библиотек и документации – по сравнению с Python и JS;

– сложнее освоить новичкам.

Когда использовать: для высоконагруженных ботов, работающих с миллионами пользователей.

PHP – серверный язык, часто используемый для веб-разработки [16].

Достоинства:

– простота – легко интегрируется в существующие веб-приложения;

– хорошая работа с MySQL – подходит для простых ботов с БД;

– большое количество хостингов – легко развернуть на любом сервере.

Недостатки:

– низкая производительность – хуже, чем у Go, Java, C#;

– слабая многопоточность – не подходит для высоких нагрузок.

Когда использовать: для простых ботов, интеграции с сайтами, базами данных.

Выбор языка зависит от целей проекта:

– для быстрого старта → Python;

– для веб-разработки → JavaScript (Node.js);

– для высокопроизводительных решений → C#, Go;

– для корпоративных решений → Java;

– для простых интеграций с вебом → PHP.

Если нужен оптимальный баланс между простотой и мощностью → C# (Telegram.Bot) или Python (aiogram).

2.2.3 Выбор СУБД

Для хранения данных бота (пользовательская информация, логи, сообщения, настройки) можно использовать различные системы управления базами данных (СУБД). Выбор зависит от требований к производительности, масштабируемости и удобству работы с данными.

PostgreSQL – одна из самых популярных реляционных баз данных с открытым исходным кодом [17].

Достоинства:

- поддержка ACID-транзакций – надежность и целостность данных;
- гибкость – JSONB, геоданные, полнотекстовый поиск;
- высокая производительность – хорошо работает с большими объемами данных;
- отлично масштабируется – поддержка шардинга и репликации.

Недостатки:

- требует больше ресурсов по сравнению с MySQL;
- сложнее в настройке для начинающих.

Когда использовать: для сложных ботов с большим объемом данных, аналитикой и интеграцией с другими сервисами.

MySQL / MariaDB – классическая реляционная база данных, популярная в веб-разработке [18].

Достоинства:

- простота установки и настройки;
- хорошая производительность на чтение;
- большое количество готовых решений;
- совместимость с большинством хостингов.

Недостатки:

- меньше возможностей для работы с JSON;
- менее гибкая система индексов, чем в PostgreSQL.

Когда использовать: для небольших и средних проектов с традиционной структурой данных.

Microsoft SQL Server –проприетарная реляционная СУБД от Microsoft, активно используется в корпоративной среде [19].

Достоинства:

- высокая надежность – хорошо подходит для бизнеса;
- интеграция с .NET – удобна для C#-разработчиков;
- отличная аналитика и отчеты.

Недостатки:

- платная (есть бесплатная версия Express, но с ограничениями);
- требовательна к ресурсам.

Когда использовать: для корпоративных решений и интеграции с Windows-инфраструктурой.

2.3 Выбранные технологии

При выборе технологий учитывались:

- поддержка асинхронного взаимодействия – важно для быстрого ответа бота;
- совместимость с Telegram API – возможность работы с ботом через HTTP-запросы;
- простота масштабирования – возможность работы с большим числом пользователей.

Были выбраны следующие инструменты:

- язык программирования: C#
- фреймворк: .NET 9
- база данных: SQL Server
- библиотека для работы с Telegram: Telegram.Bot
- очереди запросов: Task, PoolThread для асинхронных запросов

Основные технологии, использованные в проекте:

- .NET Core – для построения серверной части;
- MyORM – самописная ORM для работы с базой данных, реализована самостоятельно, для легковесности и гибкости решения;
- Telegram.Bot – библиотека для интеграции с Telegram API;
- Docker – контейнеризация и деплой;
- Logger – самописная система для логирования в консоль и текстовый файл;
- UserManager – управление ролями пользователей для разделения состояния пользователя.

Эти технологии позволяют создать надежную и удобную в поддержке систему.

3 Проектирование и разработка

3.1 Регистрация чат-бота

Для функционирования Телеграмм чат-бота необходимо зарегистрировать информацию о создаваемом чат-боте. Для этого в Телеграмм реализован чат-бот @BotFather в котором происходит регистрация новых чат-ботов. Информация о данном чат-боте представлена на рисунке 10.



Рисунок 10 – Телеграмм чат-бот для управления чат-ботами

После начала работы с данным чат-ботом приходит сообщение, в котором есть информация о доступных командах, как показано на рисунке 11.

Далее необходимо зарегистрировать новый чат-бот путем вызова команды /new_bot и указать все данные, которые будет запрошены. В случае успешно введенных данных, новый чат-бот будет зарегистрирован и придет сообщение с информацией о чат-боте и его токене для подключения через код. Весь процесс показан на рисунке 12.

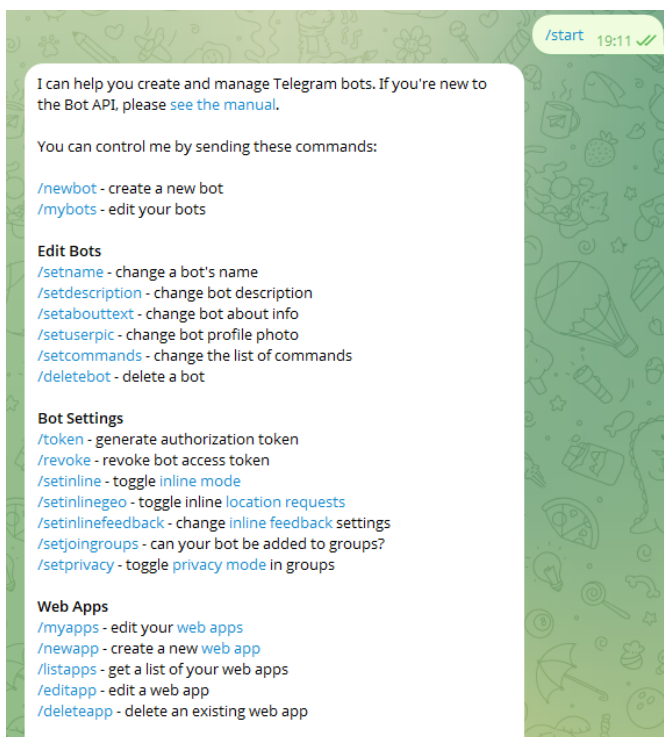


Рисунок 11 – Список доступных команд

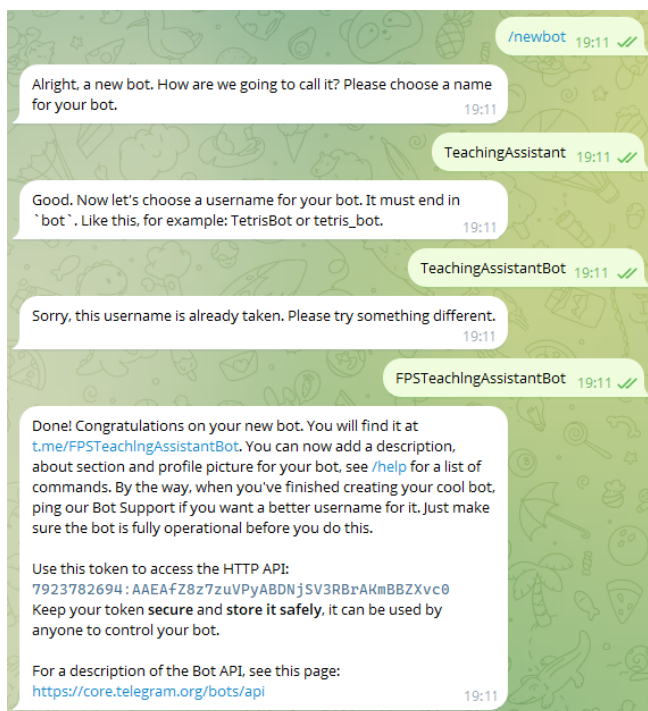


Рисунок 12 – Регистрация чат-бота

3.2 Проектирование БД

Для удобного и надежного хранения данных, используется СУБД.

Таблица User необходима для содержания информации о пользователях бота. Поля данной таблицы указаны в таблице 2.

Таблица 2 – Атрибуты таблицы “User”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
Id	integer	Идентификатор пользователя	РК	1
TelegramId	varchar	Идентификатор телеграмм клиента		111111111
NickName	varchar	Псевдоним телеграмм пользователя		Sample_name
FullName	varchar	ФИО пользователя		Иван Иванович Иванов
Password	varchar	Пароль пользователя		1111
Role	varchar	Роль в рамках сессии пользователя		Teacher

Таблица Grade необходима для содержания информации об оценках. Поля данной таблицы указаны в таблице 3.

Таблица 3 – Атрибуты таблицы “Grade”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
Id	integer	Идентификатор оценки	РК	1
StudentId	integer	Идентификатор студента		1
ScheduleId	integer	Идентификатор расписания		1
Value	integer	Значение оценки		5

Таблица Subject необходима для содержания информации о предметах. Поля данной таблицы указаны в таблице 4.

Таблица 4 – Атрибуты таблицы “Subject”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
Id	integer	Идентификатор предмета	РК	1
Name	varchar	Название предмета		Распределенные вычисления
Description	varchar	Описание предмета		Изучение принципов вычислений
TeacherId	integer	Код преподавателя, закрепленного за предметом		1
GradeLevel	integer	Уровень обучающегося студента, год или курс		1
CraetedAt	varchar	Дата создания или добавления		01.01.2025

Таблица Schedule необходима для содержания информации о расписании. Поля данной таблицы указаны в таблице 5.

Таблица 5 – Атрибуты таблицы “Schedule”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
Id	integer	Идентификатор расписания	РК	1
TeacherId	integer	Идентификатор преподавателя		1
SubjectId	integer	Идентификатор предмета		1
StudyGroupId	integer	Идентификатор учебной группы		1
EventDate	varchar	Дата проведения		01.01.2025
LessonNumber	integer	Номер урока или пары		1

Таблица EducationalMaterial необходима для содержания информации об учебных материалах. Поля данной таблицы указаны в таблице 6.

Таблица 6 – Атрибуты таблицы “EducationalMaterial ”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
Id	integer	Идентификатор учебного материала	РК	1
Name	varchar	Название учебного материала		Лекции
Description	varchar	Описание учебного материала		Лекционные материалы по предмету 'АиКС'
LinkToSource	varchar	Ссылки		https://ru.wikipedia.org

Таблица Presence необходима для содержания информации о присутствующих на занятиях. Поля данной таблицы указаны в таблице 7.

Таблица 7 – Атрибуты таблицы “ Presence ”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
Id	integer	Идентификатор учебного материала	РК	1
StudentId	integer	Идентификатор студента		1
ScheduleId	integer	Идентификатор расписания		1

Таблица Student_StudyGroup необходима для содержания информации о связях студента с учебной группой. Поля данной таблицы указаны в таблице 8.

Таблица 8 – Атрибуты таблицы “ Student_StudyGroup ”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
StudentId	integer	Идентификатор студента	РК	1
StudyGroupId	integer	Идентификатор студенческой группы		1

Таблица FAQ необходима для содержания информации о часто задаваемых вопросах. Поля данной таблицы указаны в таблице 9.

Таблица 9 – Атрибуты таблицы “ FAQ ”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
Id	integer	Идентификатор часто задаваемых вопросов	РК	1
Name	varchar	Название вопроса		Критерии оценивания
Description	varchar	Описание или ответ на вопрос		Выполнение полного объема, своевременная сдача - отлично

Таблица StudyGroup необходима для содержания информации об учебных группах. Поля данной таблицы указаны в таблице 10.

Таблица 10 – Атрибуты таблицы “ StudyGroup ”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
Id	integer	Идентификатор студенческой группы	РК	1
TeacherId	integer	Идентификатор преподавателя закрепленного за группой		1
Name	varchar	Название группы		21ВВП2
StartDate	varchar	Дата создания группы		01.01.2021
EndDate	varchar	Дата окончания группы		01.01.2025

Таблица MailingList необходима для содержания информации о списке рассылки. Поля данной таблицы указаны в таблице 11.

Таблица 11 – Атрибуты таблицы “ MailingList ”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
Id	integer	Идентификатор студента	РК	1
Text	varchar	Текст сообщения		Пример сообщения
StudyGroupId	varchar	Идентификатор учебной группы		1
DispatchDate	varchar	Дата рассылки		01.01.2025
DispatchTime	varchar	Время рассылки		00:00

Таблица Homework необходима для содержания информации о домашних заданиях. Поля данной таблицы указаны в таблице 12.

Таблица 12 – Атрибуты таблицы “Homework”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
Id	integer	Идентификатор домашнего задания	РК	1
GroupId	integer	Идентификатор группы		1
Name	varchar	Название домашнего задания		Домашняя работа №1
StartDate	varchar	Дата начала		01.01.2025
EndDate	varchar	Дата окончания		01.02.2025
LinkToSource	varchar	Ссылка на ресурсы		https://drive.google.com

Таблица AnswerToHomework необходима для содержания информации об ответах на домашние задания. Поля данной таблицы указаны в таблице 13.

Таблица 13 – Атрибуты таблицы “AnswerToHomework”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
Id	integer	Идентификатор домашнего задания	РК	1
StudentId	integer	Идентификатор студента		1
HomeworkId	integer	Идентификатор домашнего задания		1
Grade	integer	Оценка		5
DateAdded	varchar	Дата добавления		01.01.2025
LinkToSource	varchar	Ссылка на загруженные материалы		https://drive.google.com

Схема таблиц БД приведена на рисунке 13.

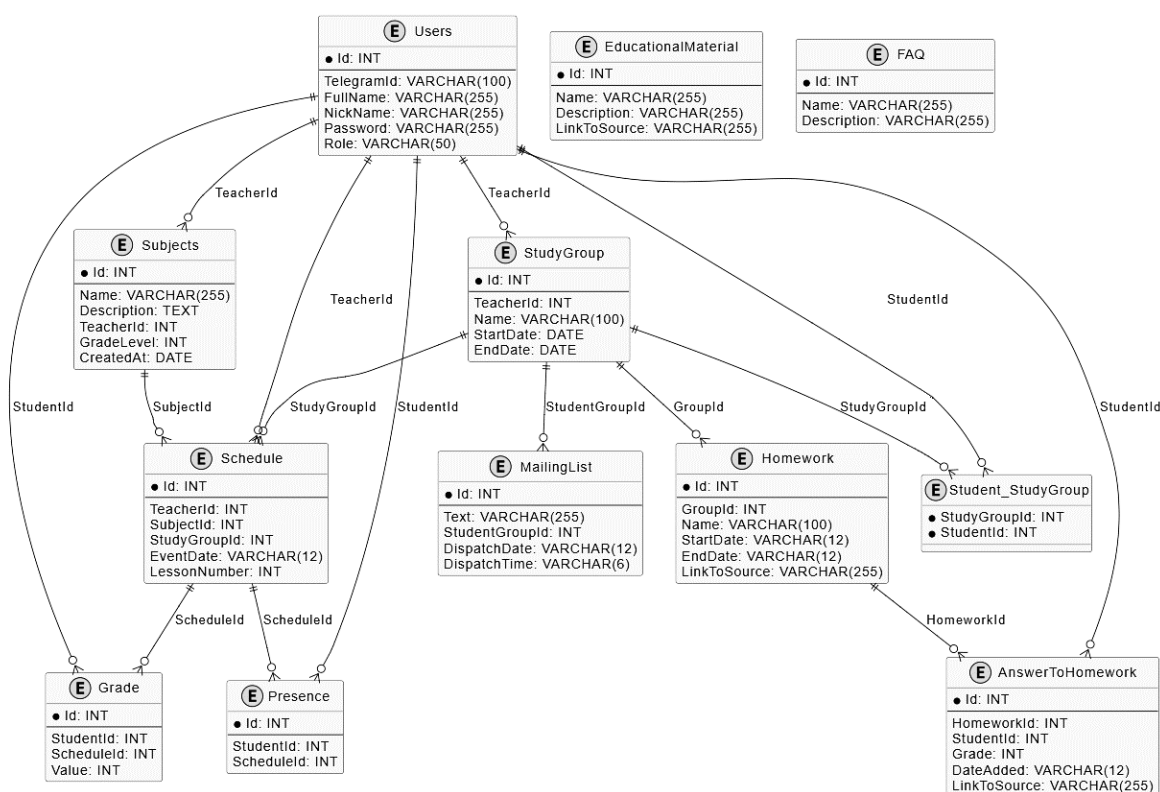


Рисунок 13 – Физическая модель БД

3.3 Описание программы

Диаграмма развертывания представлена на рисунке 14.

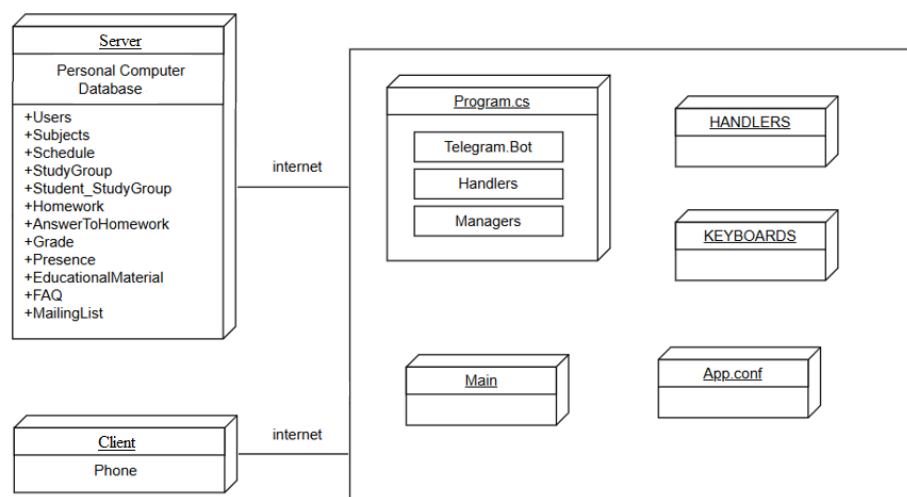


Рисунок 14 – UML диаграмма развертывания

В файле App.config содержится информация о токене для подключения к чат-боту и строки подключения к базе данных.

```
<configuration>
  <connectionStrings>
    <add name="DbTypeLocalSQL"
      providerName="Microsoft.Data.SqlClient"
      connectionString=
"Server=localhost;Database=TeachingAssistant_DB;Trusted_Connecti
on=True;Integrated Security=True;Encrypt=False;"
    />
    <add name="TelegramBotToken"
      providerName="Telegram.Bot.TelegramBotClient"

connectionString="7316188451:AAEfTmyg2viamgC3Xi3fK3c_lQKp8RVhNZw
"
    />
  </connectionStrings>
</configuration>
```

В главной функции, которая является точкой входа текущего приложения, инициализируются все компоненты и запускается обработка входных сообщений пока не будет вызвана команда stop в консоли.

```

        static async Task Main(string[] args)
        {
            string token =
ConfigurationManager.ConnectionStrings["TelegramBotToken"].Conne
ctionString;
            _botClient = new(token);

            _userManager = new();
            _commandHandler = new(_userManager);

            _updateHandelr = new(_commandHandler);
            _errorHandelr = new();
        }

        private static void Exiting()
        {
            Console.WriteLine("Command: stop called");
            Console.WriteLine("Startup exit ...");
            Console.WriteLine("Exited");
        }
    }
}

```

Обработка всех ошибок, возникающих на стороне API Telegram.Bot обрабатываются в классе ErrorHandler.

```

        public async Task HandleErrorAsync(ITelegramBotClient
client, Exception exception, HandleErrorSource source,
Cancellation token)
        {
            try
            {
                var ErrorMessage = exception switch
                {
                    ApiRequestException apiRequestException
                        => $"Telegram API
Error:\n[{apiRequestException.ErrorCode}]\n{apiRequestException.
Message}",
                    _ => exception.ToString()
                };

                Logger.Instance.Log(ErrorMessage,
LogLevel.Error);
            }
        }

```

Обработка введенный сообщений в виде нажатий на кнопки или текстовые сообщений и команды, происходит в классе UpdateHandler.

```

        public async Task HandleUpdateAsync(ITelegramBotClient
botClient, Update update, Cancellation token)

```

```

        {
            if (update.Message is not null)
            {
                long userId = update.Message.From.Id;

                await Task.Run(() =>
ProcessUserMessageRequest(userId, botClient, update));
            }
            else if (update.CallbackQuery is not null)
            {
                long userId = update.CallbackQuery.From.Id;
                await Task.Run(() =>
ProcessUserCallbackRequest(userId, botClient, update));
            }
        }

```

Каждое введенное значение соответствует зарегистрированной команде в определенных классах, в которых сконцентрированы команды для конкретного пользователя. Выбор, какой именно набор команд запустить происходит в классе CommandHandlerManager.

```

public async Task ExecuteCommandForMessageAsync(long
userId, ITelegramBotClient botClient, Update update)
{
    var message = update.Message;
    if (message == null)
    {
        var t_chat = message.Chat;
        await
_commonCommands.UnrecognizedMessage(botClient, t_chat);
        return;
    }

    if (CatcherUserMessageInput.Instance.IsCatch)
    {
        if (await TryCatchUserMessageInput(userId,
botClient, message))
            return;
    }

    var userSession =
_sessionManager.GetOrCreateSession(userId);

    if (update.Message.Text == CommandNames.FAQ ||
update.Message.Text == CommandNames.EDUCATION_MATERIAL)
    {
        await
_commonCommands.ExecuteMessageCommandAsync(userId, botClient,
update);
        return;
    }
}

```

```
}
```

Общий набор команд описан в классе CommonCommands.

```
public async Task
CreateMenuCommandsAsync (ITelegramBotClient botClient)
{
    var bot_commands = new[]
    {
        new BotCommand { Command = "menu", Description =
"Показать меню" },
        new BotCommand { Command = "faq", Description =
"Часто задаваемые вопросы" },
        new BotCommand { Command = "educational_material",
Description = "Учебные материалы" },
        new BotCommand { Command = "help", Description =
"Помощь" },
        new BotCommand { Command = "exit", Description =
"Выйти из учетной записи чат-бота" }
    };

    await botClient.SetMyCommands (bot_commands);
}
```

Все команды, связанные с авторизацией описаны в классе LoginMainMenuCommands.

```
public class LoginMainMenuCommands : CommonUserCommands
{
    private async Task ViewLoginMenuMessageCommandAsync (long
userId, ITelegramBotClient botClient, Update update)
    {
        var t_chat = update.Message.Chat;

        var inlineKeyboard = new InlineKeyboardMarkup(
            new List<InlineKeyboardButton[]>()
            {
                new
                InlineKeyboardButton[] {
                    InlineKeyboardButton.WithCallbackData ("Вход", ButtonNames.LOGIN),
                },
                new
                InlineKeyboardButton[]
                {

```

```

        InlineKeyboardButton.WithCallbackData("Регистрация",
        ButtonNames.REGISTRATION),
    });
    });

    await botClient.SendMessage(
        t_chat.Id,
        "Добро пожаловать в чат-бот \"Ассистент учебного
процесса\"!\n" +
        "Доступные действия:\n",
        replyMarkup: inlineKeyboard
    );
}

private async Task LoginCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
{
    var t_chat = update.CallbackQuery.Message.Chat;

    await botClient.SendMessage(
        t_chat.Id,
        $"Для входа отправьте ваш пароль от учетной
записи!\n"
    );

    CatcherUserMessageInput.Instance.HandlerAdd(userId,
    OnLoginButtonCatchUserInput);
}

private async Task RegistrationCallbackCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
{
    var t_chat = update.CallbackQuery.Message.Chat;
    var t_user = update.CallbackQuery.From;

    await botClient.SendMessage(
        t_chat.Id,
        $"Для регистрации сообщите системному
администратору ваши следующие данные:\n" +
        $"TelegramId: {t_user.Id}\n" +
        $"FullName: ваше ФИО\n" +
        $"Telegram NickName: {t_user.Username}\n\n" +
        $"После регистрации системный администратор сообщит
вам данные для входа ..."
    );
}

userSession.Role = currentUser.Role;
userSession.CurrentState =
CurrentStateNames.MAIN_MENU;

```

```

        string role = userSession.Role == RoleNames.STUDENT ?
"студент" : "преподаватель";
        role = userSession.Role == RoleNames.ADMIN ?
"администратор" : role;

        await botClient.SendMessage(
            t_chat.Id,
            $"Вход выполнен! " +
            $"Вы вошли как {role}.\n" +
            $"Для просмотра меню действий вызовите команду
{CommandNames.MENU}"
        );
    }
}

```

Все команды связанные с режимом работы администратора описаны в классе AdminMainMenuCommands.

```

namespace Teaching_Assistant.Commands.User
{
    public class AdminMainMenuCommands : CommonUserCommands
    {
        public AdminMainMenuCommands(UserSessionManager
sessionManager) : base(sessionManager)
        {
            _commands = new Dictionary<string, Func<long,
ITelegramBotClient, Update, Task>>
            {
                { CommandNames.MENU,
ViewAdminMenuMessageCommandAsync },
                { CommandNames.HELP, ViewHelpMessageCommandAsync
},
                { CommandNames.EXIT,
ExitAccountMessageCommandAsync },
                { ButtonNames.BACK,
ExitAccountCallbackCommandAsync },
                { ButtonNames.EXIT,
ExitAccountCallbackCommandAsync },
                { ButtonNames.STUDENT_LOGIN,
StudentLoginCallbackCommandAsync },
                { ButtonNames.TEACHER_LOGIN,
TeacherLoginCallbackCommandAsync },
                { ButtonNames.ACCOUNT_INFO, AccountInfoCallbackCommandAsync
},
                { ButtonNames.SHOW_PASSWORD,
ShowPasswordCallbackCommandAsync }
            };
        }
    }
}

```

```

private async Task ViewAdminMenuMessageCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
{
    var t_chat = update.Message.Chat;

    var inlineKeyboard = new InlineKeyboardMarkup(
        new List<InlineKeyboardButton[]>()
        {
            new
InlineKeyboardButton[]
            {
                InlineKeyboardButton.WithCallbackData("Выйти", ButtonNames.BACK),

                InlineKeyboardButton.WithCallbackData("Личные данные",
ButtonNames.ACCOUNT_INFO),

            },
            new
InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Войти как студент",
ButtonNames.STUDENT_LOGIN),

            },
            new
InlineKeyboardButton[]
            {
                InlineKeyboardButton.WithCallbackData("Войти как преподаватель",
ButtonNames.TEACHER_LOGIN),

            },
        });

    await botClient.SendMessage(
        t_chat.Id,
        "Доступные действия:\n",
        replyMarkup: inlineKeyboard
    );
}

private async Task StudentLoginCallbackCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
{
    var t_chat = update.CallbackQuery.Message.Chat;

    UserSession session =
_sessionManager.GetOrCreateSession(userId);
    session.Role = RoleNames.STUDENT;
    session.CurrentState = CurrentStateNames.MAIN_MENU;

    await botClient.SendMessage(
        t_chat.Id,

```

```

        $"Вы вошли как студент!\n" +
        $"Для просмотра меню действий вызовите команду
{CommandNames.MENU}"
        );
    }

    private async Task TeacherLoginCallbackCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
    {
        var t_chat = update.CallbackQuery.Message.Chat;

        UserSession session =
_sessionManager.GetOrCreateSession(userId);
        session.Role = RoleNames.TEACHER;
        session.CurrentState = CurrentStateNames.MAIN_MENU;

        await botClient.SendMessage(
            t_chat.Id,
            $"Вы вошли как преподаватель!\n" +
            $"Для просмотра меню действий вызовите команду
{CommandNames.MENU}"
        );
    }
}

```

Все команды, связанные со студентами, описаны в классе StudentMainMenuCommands.

```

public class StudentMainMenuCommands : CommonUserCommands
{
    public StudentMainMenuCommands(UserSessionManager
sessionManager) : base(sessionManager)
    {
        private async Task VeiwMenuMessageCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
        {
            var inlineKeyboard = new InlineKeyboardMarkup(
                new List<InlineKeyboardButton[]>()
                {
                    new
                    InlineKeyboardButton[] {
                        InlineKeyboardButton.WithCallbackData("Личные данные",
ButtonNames.ACCOUNT_INFO),
                        InlineKeyboardButton.WithCallbackData("Выйти", ButtonNames.EXIT)
                    },
                }
            );
        }
    }
}

```

```

new
InlineKeyboardButton[]
{
InlineKeyboardButton.WithCallbackData("Просмотр расписания на
сегодня", ButtonNames.STUDENT_SHOW_DAY_SCHEDULE),
},
new
InlineKeyboardButton[] {
InlineKeyboardButton.WithCallbackData("Добавить домашнее
задание", ButtonNames.STUDENT_SHOW_MENU_FOR_ADDING_HOWEWORK),
},
new
InlineKeyboardButton[] {
InlineKeyboardButton.WithCallbackData("Просмотр моих оценок по
дате", ButtonNames.STUDENT_SHOW_MY_GRADES_BY_DATE),
},
new
InlineKeyboardButton[] {
InlineKeyboardButton.WithCallbackData("Просмотр моих оценок по
предмету", ButtonNames.STUDENT_SHOW_MY_GRADES_BY_SUBJECT),
}
});

```

Код всех команд для преподавателей описан в классе
TeacherMainMenuCommands.

```

public class TeacherMainMenuCommands : CommonUserCommands
{
private async Task VeiwMenuMessageCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
{
var t_chat = update.Message.Chat;

var inlineKeyboard = new InlineKeyboardMarkup(
new List<InlineKeyboardButton[]>()
{
new
InlineKeyboardButton[] {
InlineKeyboardButton.WithCallbackData("Личные данные",
ButtonNames.ACCOUNT_INFO),
InlineKeyboardButton.WithCallbackData("Выйти", ButtonNames.EXIT)
},
new
InlineKeyboardButton[]
{

```

```

InlineKeyboardButton.WithCallbackData("Просмотр расписания на
сегодня", ButtonNames.TEACHER_SHOW_DAY_SCHEDULE),
    },
    new
InlineKeyboardButton[] {

InlineKeyboardButton.WithCallbackData("Просмотр домашних
заданий", ButtonNames.TEACHER_SHOW_HOWEWORK),
    },
    new
InlineKeyboardButton[] {

InlineKeyboardButton.WithCallbackData("Добавить домашнее
задание", ButtonNames.TEACHER_ADD_HOWEWORK),
    },
    new
InlineKeyboardButton[] {

InlineKeyboardButton.WithCallbackData("Просмотр ответов на
домашние задания", ButtonNames.TEACHER_SHOW_ANSWERS_TO_HOWEWORK),
    },
    new
InlineKeyboardButton[] {

InlineKeyboardButton.WithCallbackData("Добавить сообщение для
рассылки", ButtonNames.TEACHER_ADD_MAILING),
    }
});

```

Для управления сессиями пользователей, была реализована модель UserSession.

```

public class UserSession
{
    public long TelegramId { get; }
    public string Role { get; set; } = RoleNames.GUEST;
    public string CurrentState { get; set; } =
CurrentStateNames.DEFAULT
}

```

Управление сессиями пользователей осуществляется через класс UserSessionManager.

```

public UserSession GetOrCreateSession(long userId)
{
    return _sessions.GetOrAdd(userId, id => new
UserSession(id));
}

```

Для удобства работы с базой данных были реализованы классы моделей данных по определенному шаблону, в котором описываются поля данных, свойства для преобразования в необходимую строку запроса к базе данных, и набор методов. Например, модель пользователей DbUser.

```
namespace MyORM.Models
{
    public class DbUser : IBaseModel
    {
        private int _id;
        public int Id { get => _id; set => _id = value; }
        public long TelegramId { get; set; }
        public string FullName { get; set; }
        public string NickName { get; set; }
        public string Password { get; set; }
        public string Role { get; set; }

        public DbUser() { }
        public DbUser(int id, long telegramId, string fullname,
string nickname, string login, string password, string role)
    }
}
```

Для управления моделями данных реализованы контроллеры, которые осуществляют все необходимые действия относительно своей модели данных, например, DbUserController.

```
public class DbUserController : DbBaseController<DbUser>
{
    public DbUserController(string provider_name, string
connection_string) :
        base(provider_name, connection_string)
    {
        _tableName = "Users";
    }

    public bool Query_TrySelectByTelegramId(long t_id, out
DbUser found_user)
    {
        string query = $"select * from {_tableName} where
TelegramId = @param0";
        found_user = Query_SelectByQuery(query,
t_id.ToString());
        return found_user != null;
    }

    public async Task<DbUser>
Query_SelectByTelegramIdAsync(long t_id)
```

```

        {
            string query = $"select * from {_tableName} where
TelegramId = @param0";
            found_user = Query_SelectByQuery(query,
t_id.ToString());
            return found_user;
        }

        public bool Query_TryLogin(long telegramId, string pass,
out DbUser login_user)
        {
            string query = $"select * from {_tableName} where
TelegramId = @param0 and Password = @param1";
            login_user = Query_SelectByQuery(query,
telegramId.ToString(), pass)
            return login_user != null;
        }
        public bool Query_Registration(ref DbUser new_user)
        {
            bool isDone = false;
            Query_TrySelectByTelegramId(new_user.TelegramId,
out new_user);
            return new_user != null && new_user.Id != 0;
        }
    }

```

4 Тестирование чат-бота

Чат-бот реализует следующие действия:

- авторизация пользователя по паролю;
- регистрация нового пользователя;
- просмотр расписания;
- просмотр домашних заданий;
- добавление на домашнее задание;
- просмотр учебных материалов;
- просмотр часто задаваемых вопросов;
- просмотр учебных материалов;
- просмотр личной информации;
- отметки присутствующих студентов;
- отметки оценки студентов;
- отметки оценки домашних работ студентов;
- добавление сообщений для рассылки в назначенную дату и время.

При первом запуске чат-бота пользователю необходимо вызвать команду /start. После этого будет доступно меню для авторизации, как показано на рисунке 15.

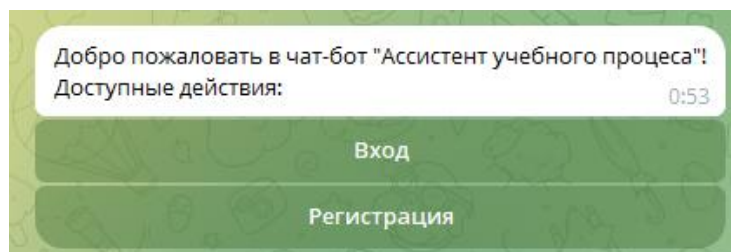


Рисунок 15 – Авторизация пользователя

Так как система планируется для закрытого использования, то здесь необходима дополнительная мера защиты в виде ввода пароля для входа. В случае успешного входа, появится сообщение, как показано на рисунке 16. Для

регистрации необходимо указать определенные данные, как показано на рисунке 17.

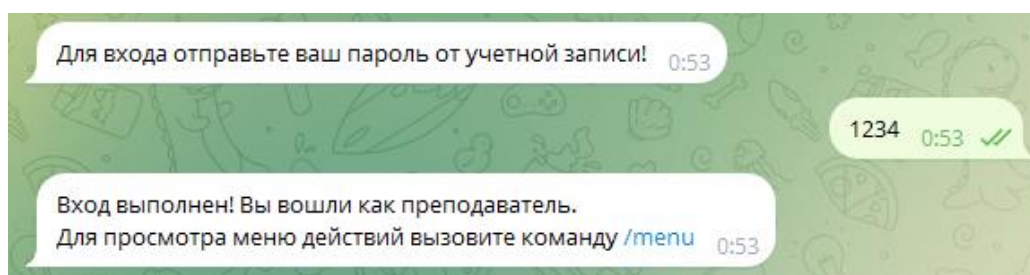


Рисунок 16 – Вход в систему

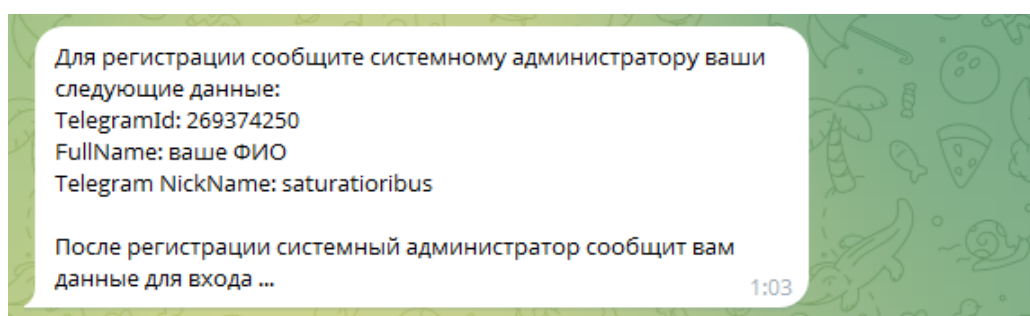


Рисунок 17 – Регистрация

Для удобства использования реализованы различные меню для разных ролей пользователей. В каждом режиме использования, относительно роли пользователя, доступно меню команд, как показано на рисунке 18.

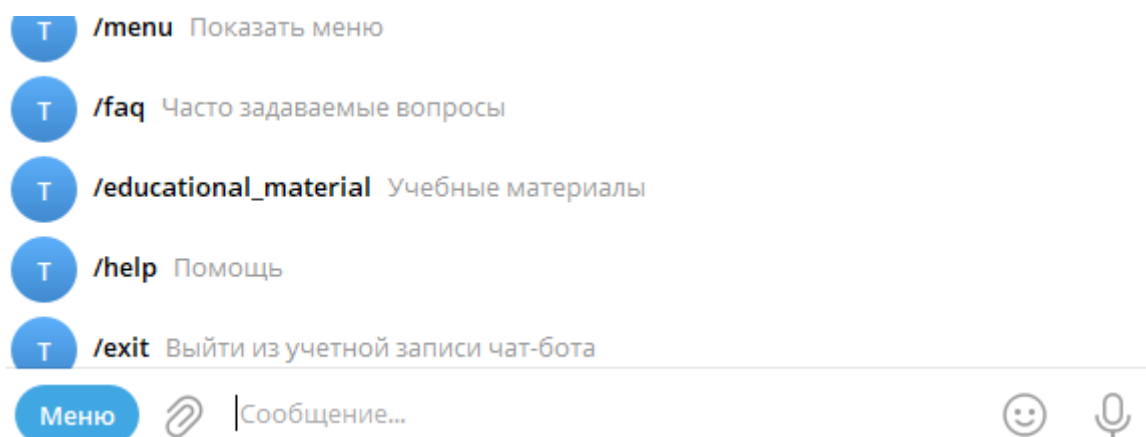


Рисунок 18 – Меню команд

На рисунке 19 показано меню для системного администратора. Основной функционал, которого в чат-боте, тестирование разработанных функций, управление данными происходит через интерфейс базы данных, консольные команды сервера, так как данный функционал был бы не так удобен для использования на стороне чат-бота.

На рисунке 20 показана меню для преподавателя. На рисунке 21 показано меню для студента.

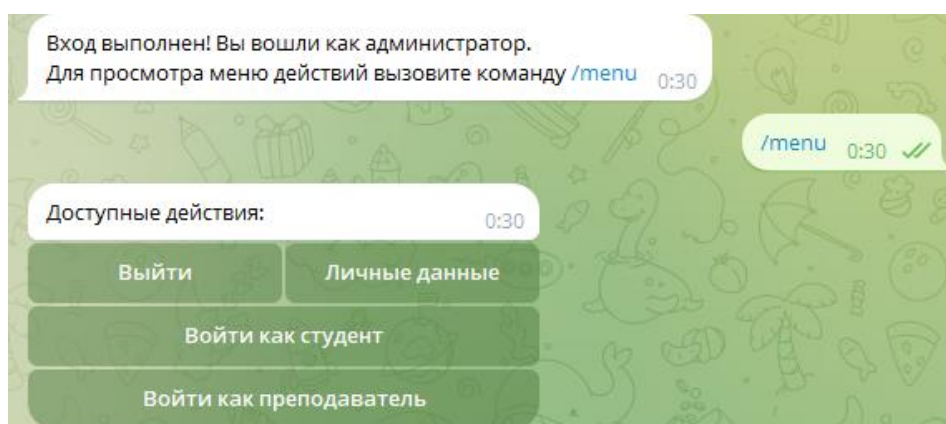


Рисунок 19 – Меню администратора

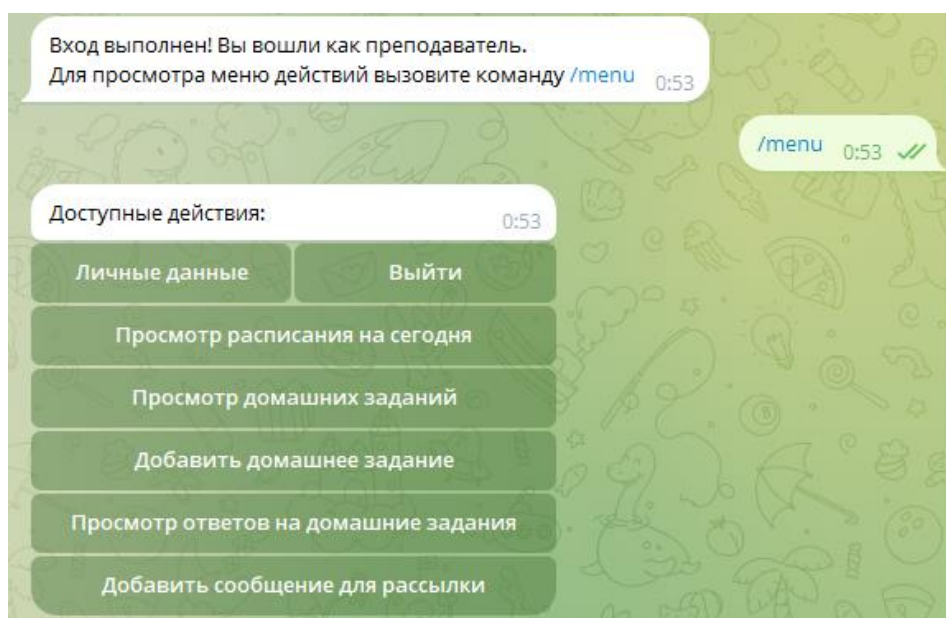


Рисунок 20 – Меню преподавателя

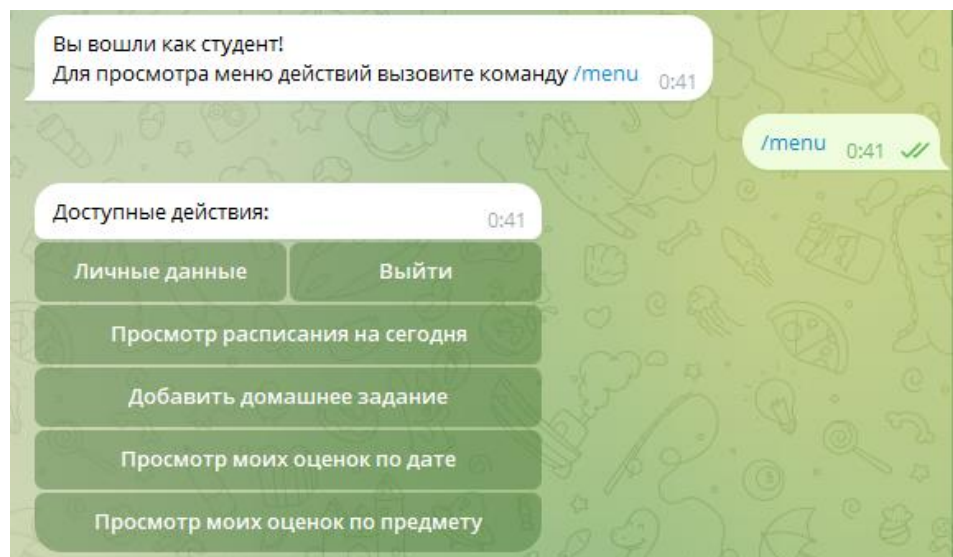


Рисунок 21 – Меню студента

Каждый пользователь может посмотреть личные данные зарегистрированные в системе, при этом пароль не показывается, как показано на рисунке 22. Пароль можно посмотреть дополнительно, при этом через определенное время сообщение с паролем будет удалено, как показано на рисунке 23.

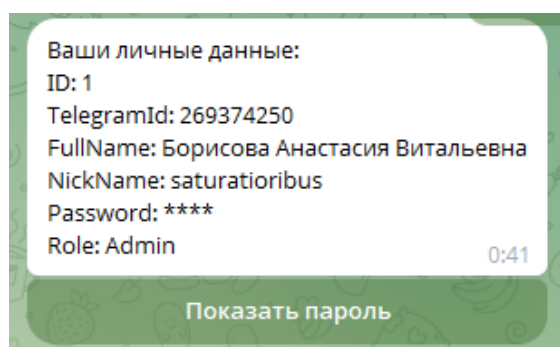


Рисунок 22 – Личные данные

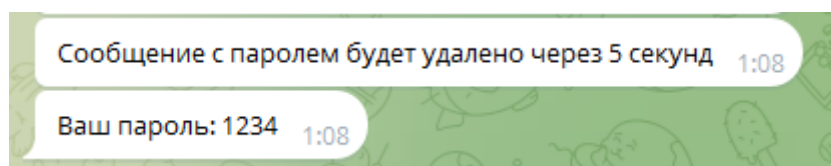


Рисунок 23 – Показ пароля

Каждый преподаватель может посмотреть расписание на сегодня, чтобы выбрать необходимое занятие (рисунок 24), на котором сможет отметить присутствующих и выставить оценки за занятие (рисунок 25).

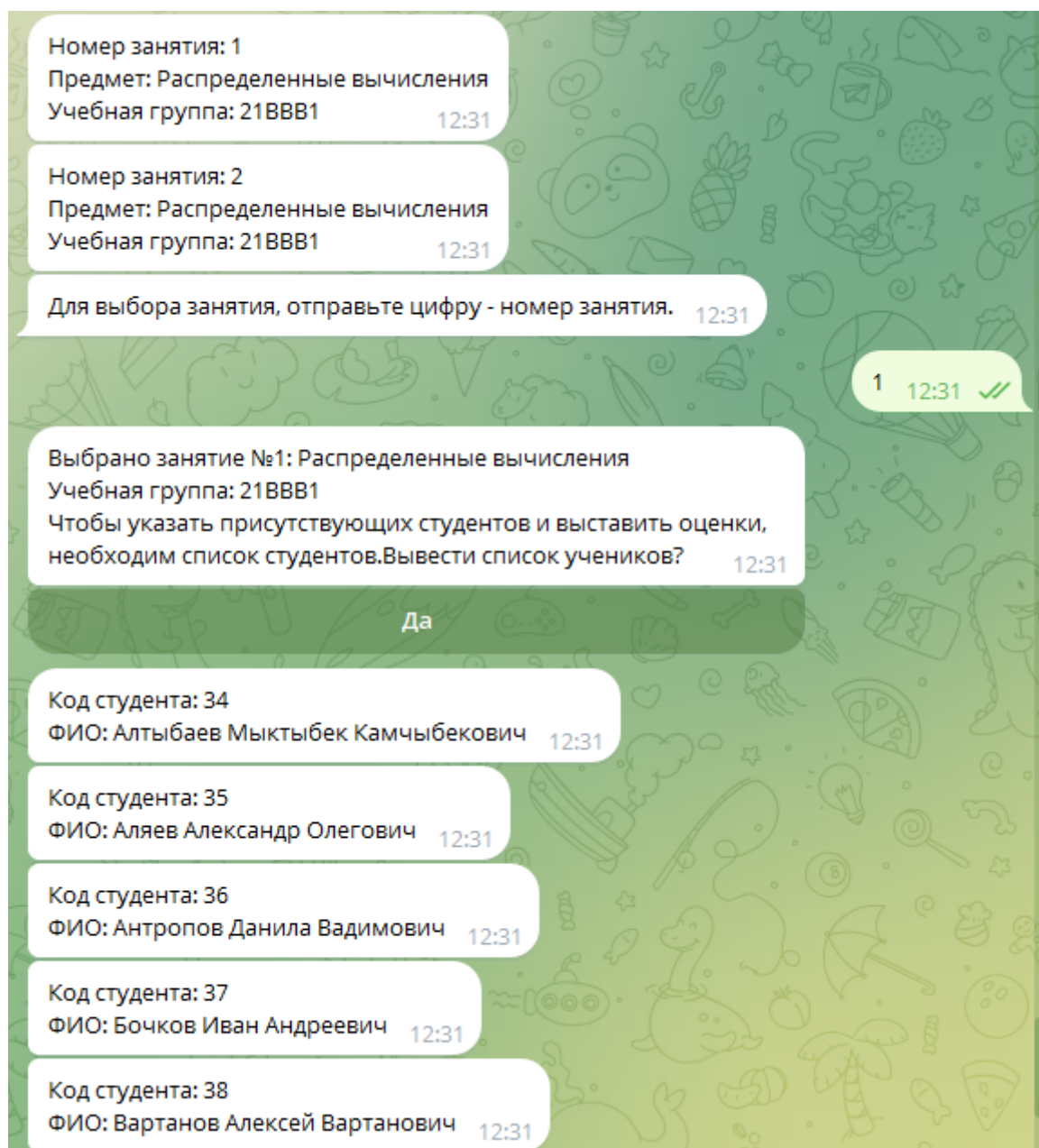


Рисунок 24 – Выбор занятия

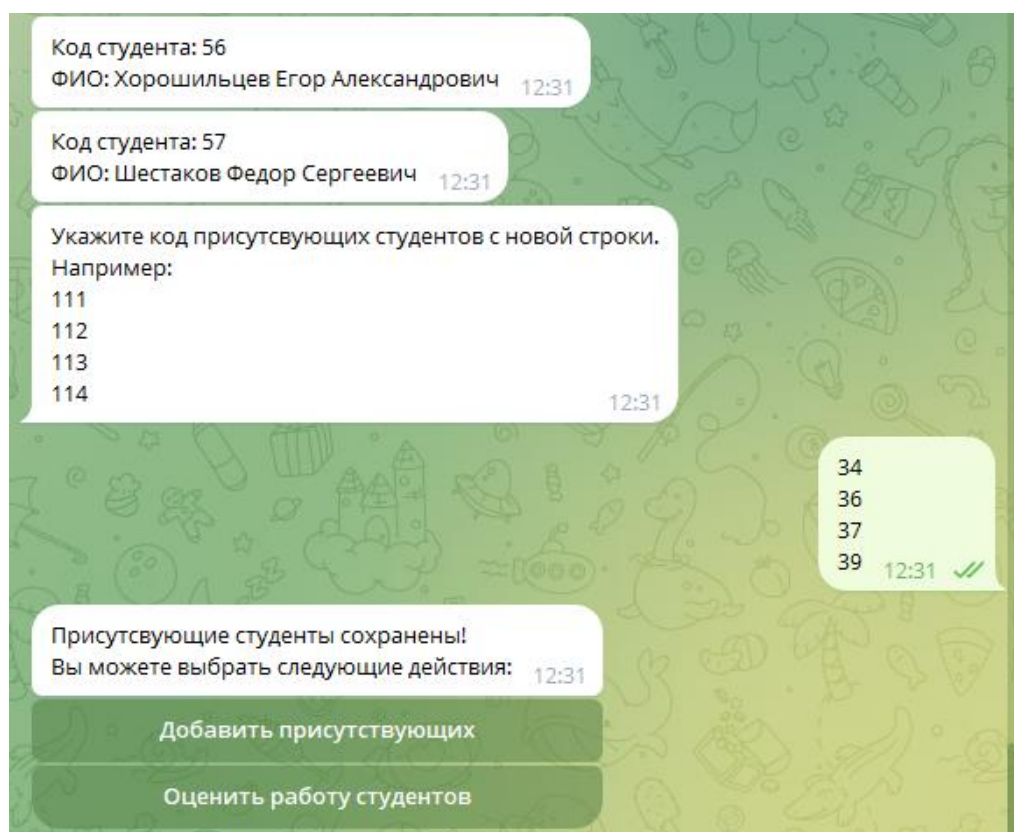


Рисунок 25 – Отметка присутствующих студентов

На рисунке 26 показан пример списка домашних заданий. На рисунке 27 показан пример добавления нового домашнего задания. На рисунке 28 показан пример оценивания домашнего задания.

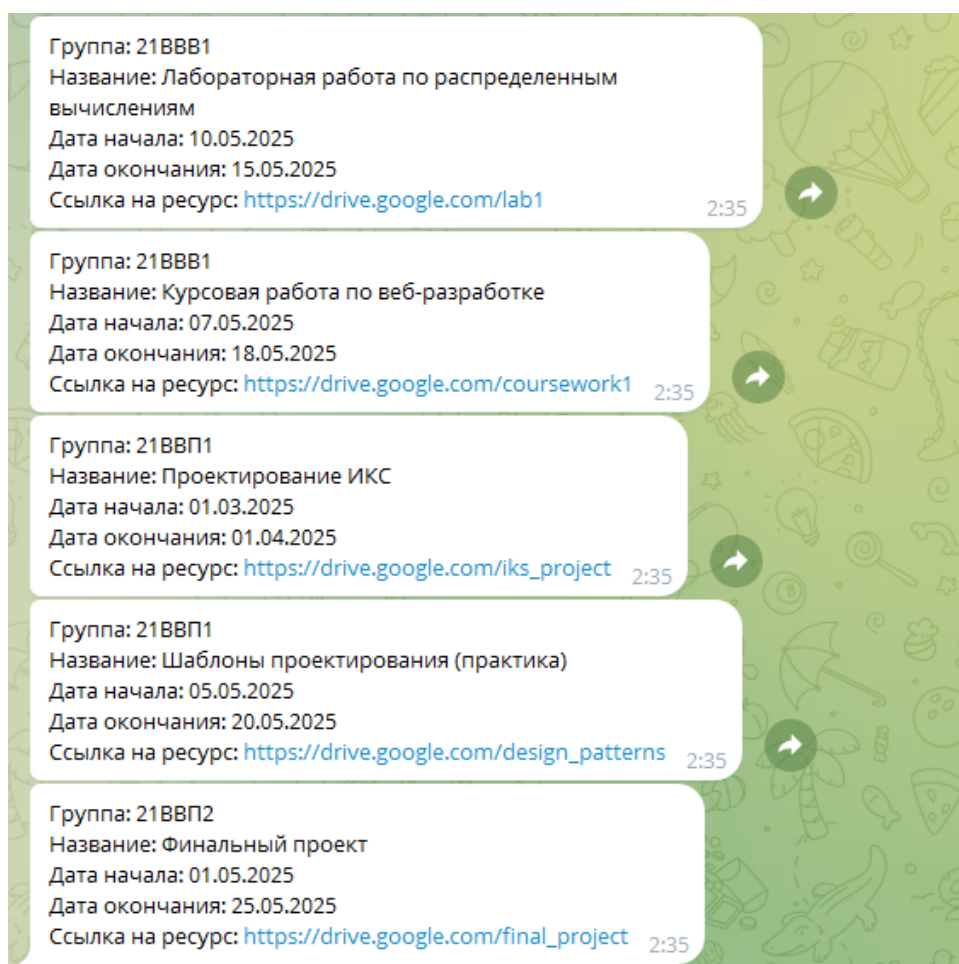


Рисунок 26 – Список домашних заданий

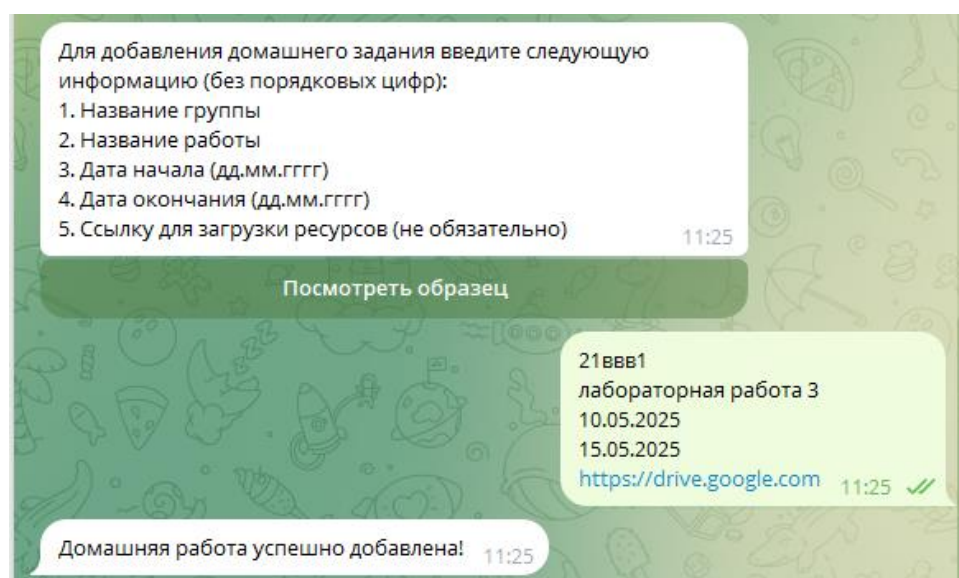


Рисунок 27 – Добавление домашней работы

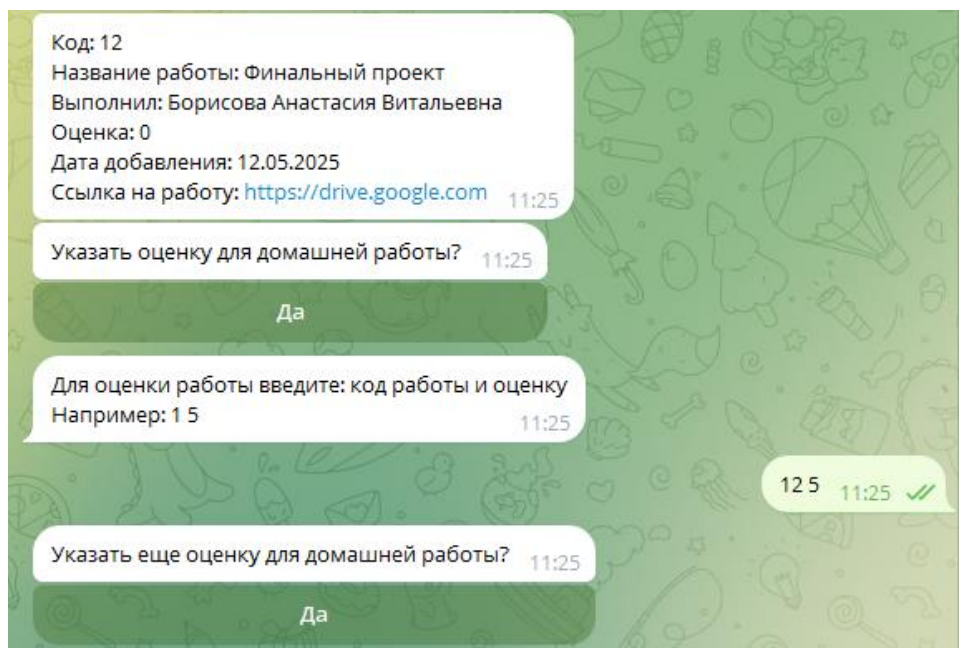


Рисунок 28 – Выставление оценки домашней работы

В случае, если был некорректный ввод во время ожидания ввода от сервера, то возникнет сообщение, в котором сообщится в чем конкретно ошибка или выведется сообщение, об ошибке на сервере, как показано на рисунке 29.

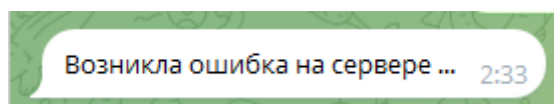


Рисунок 29 – Сообщение об ошибке

Для добавления сообщения на рассылку, необходимо указать дату и время рассылки, а также текст сообщения. Пример показан на рисунке 30.

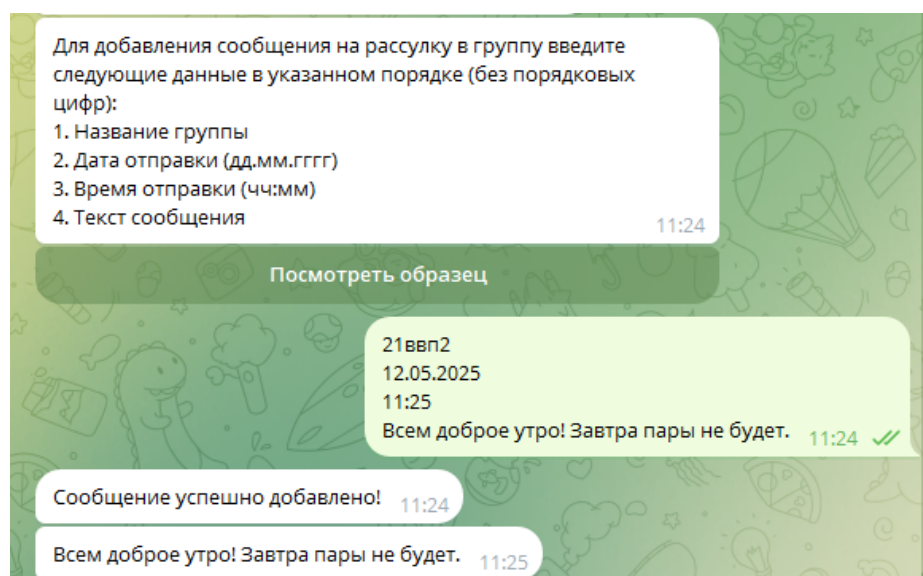


Рисунок 30 – Пример добавления сообщения на рассылку

При вызове команды /faq будет выведены сообщения с часто задаваемыми вопросами, которые хранятся в базе данных, как показано на рисунке 31.

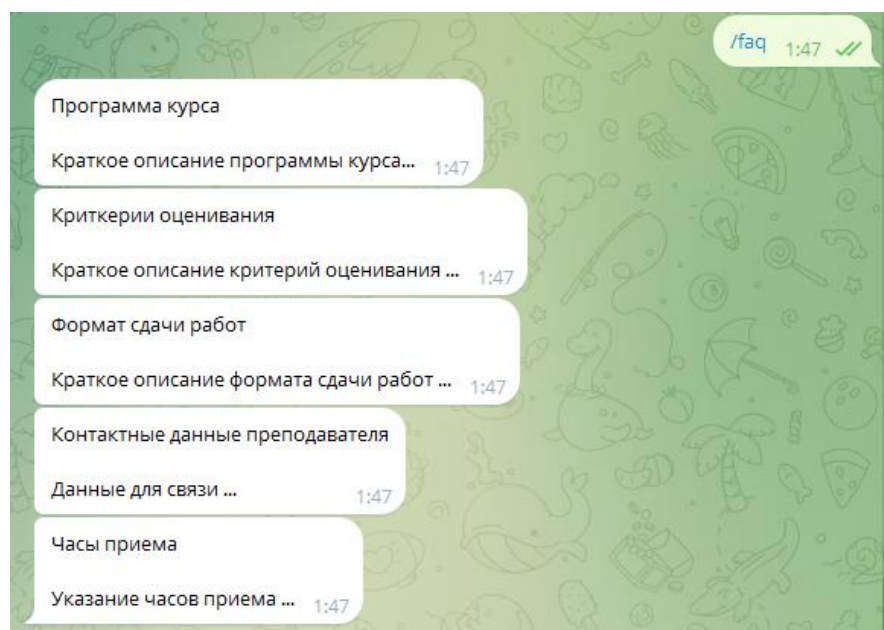


Рисунок 31 – Часто задаваемые вопросы

При вызове команды /educational_material, будут показаны все учебные материалы, которые находятся в базе данных, как показано на рисунке 32.

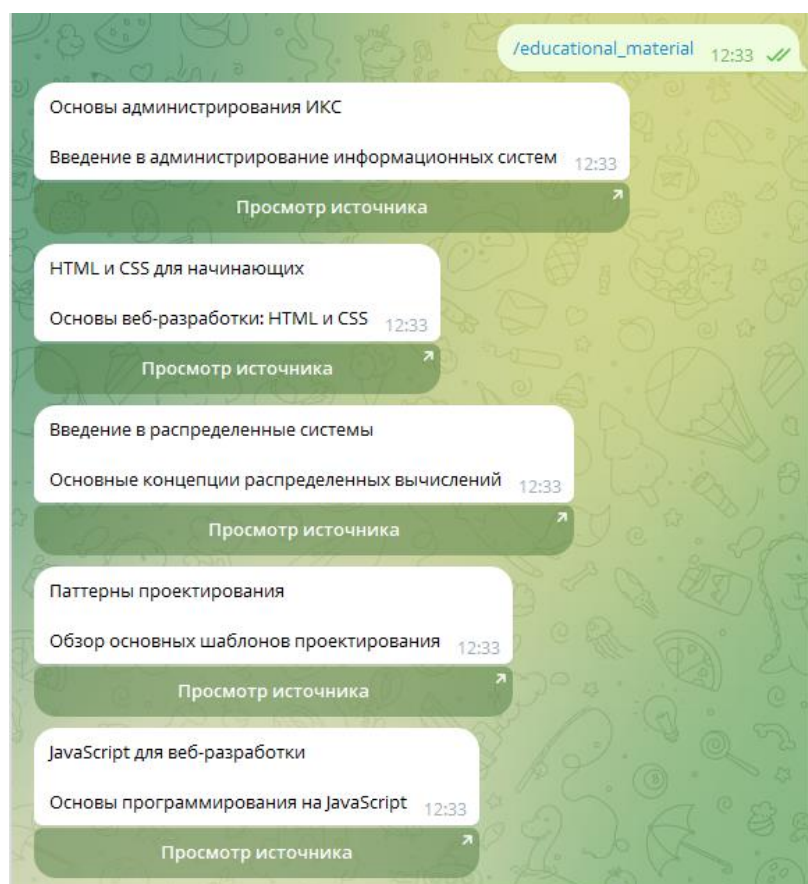


Рисунок 32 – Учебные материалы

Каждый студент может посмотреть расписание на текущий день, как показано на рисунке 33.

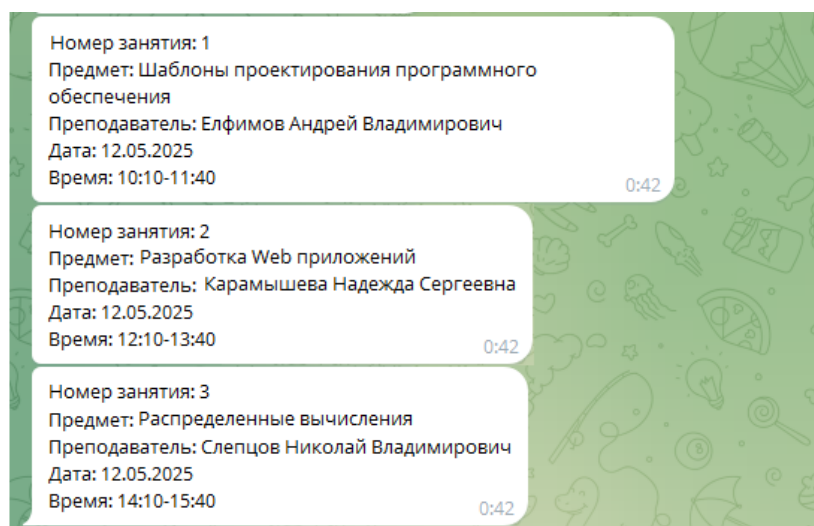


Рисунок 33 – Расписание на сегодня

Каждый студент может добавить домашнее задание, предварительно будут выведены все домашние задания для возможности указать код домашнего задания и указать ответ на него, как показано на рисунке 34.



Рисунок 34 – Добавление домашнего задания

Каждый студент может посмотреть свои оценки. Доступен просмотр оценок по дате, как показано на рисунке 35 и по названию предмета, как показано на рисунке 36.

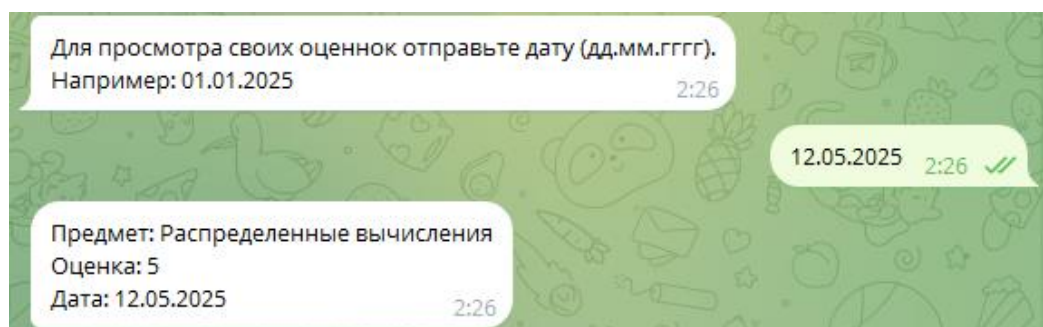


Рисунок 35 – Просмотр оценок по дате

Для просмотра своих оценок необходимо название предмета.
Доступные действия: 2:25

Показать список предметов

Ввести название предмета

Код: 1
Название: Администрирование информационно-коммуникационных систем
Преподаватель: Калиниченко Евгений Иванович 2:25

Код: 2
Название: Разработка Web приложений
Преподаватель: Елфимов Андрей Владимирович 2:25

Код: 3
Название: Распределенные вычисления
Преподаватель: Слепцов Николай Владимирович 2:25

Код: 4
Название: Шаблоны проектирования программного обеспечения
Преподаватель: Зинкин Сергей Александрович 2:25

Доступные действия: 2:25

Выбрать предмет для просмотра оценки

Для просмотра оценок введите название или код предмета.
Например: Математика
Или: 111 2:25

3 2:25 ✓✓

Предмет: Распределенные вычисления
Оценка: 5
Дата: 12.05.2025 2:25

Рисунок 36 – Просмотр оценок по названию предмету

Заключение

В рамках данной дипломной работы был разработан Telegram-бот-ассистент, предназначенный для автоматизации учебного процесса. Бот реализует широкий спектр функций, направленных на повышение удобства взаимодействия студентов и преподавателей с учебными материалами, расписанием занятий и уведомлениями о важных событиях.

В ходе работы был проведен анализ технических требований к системе, включая функциональные и нефункциональные аспекты. Изучены различные классификации чат-ботов и варианты их архитектуры. В результате был выбран многоуровневый подход к разработке, который включает клиентскую часть (бот в Telegram), серверную часть (обработка запросов, хранение данных, логика взаимодействий) и базу данных для хранения информации.

При разработке были использованы современные технологии, включая C# и .NET 9 для серверной части, SQL для хранения данных и Telegram.Bot API для интеграции с мессенджером. Это позволило создать гибкую, масштабируемую и удобную в использовании систему.

Основные результаты работы: Реализована регистрация и авторизация пользователей с разграничением ролей (студент, преподаватель, администратор). Разработан удобный интерфейс взаимодействия через Telegram, обеспечивающий доступ к расписанию занятий, учебным материалам и рассылки сообщений.

Обеспечена поддержка многопользовательского режима, при котором каждый пользователь взаимодействует с ботом независимо от других.

Разработанный бот может быть доработан и расширен в будущем, например, путем интеграции с нейросетями для интеллектуального ответа на вопросы студентов, подключением голосовых команд или внедрением системы персонализированного обучения.

Таким образом, данная работа продемонстрировала эффективность использования Telegram-ботов в образовательной сфере и показала перспективность подобных решений для автоматизации учебных процессов.

Список использованных источников

1. Стивенс Р. / Совершенный код. Мастер-класс / ISBN: 978-5-8459-1694
2. Telegram APIs / <https://core.telegram.org/> / дата доступа: 03.04.2025
3. WhatsApp Help Center / <https://faq.whatsapp.com/> / (дата обращения: 01.05.2025)
4. Messenger Platform / <https://developers.facebook.com/docs/messenger-platform/> / (дата обращения: 03.05.2025)
5. Viber Developers / <https://developers.viber.com/> / (дата обращения: 01.05.2025)
6. Slack API / <https://api.slack.com/> / (дата обращения: 03.05.2025)
7. Discord Developer Portal / <https://discord.com/developers/docs/intro/> / (дата обращения: 06.05.2025)
8. Лутц М. / Изучаем Python / ISBN: 978-5-8459-1692-7
9. Дукетт Дж. / JavaScript и jQuery. Интерактивная веб-разработка / ISBN: 978-5-8459-1856-3
10. From BotFather to 'Hello World' / <https://core.telegram.org/bots/tutorial/> / (дата обращения: 01.05.2025)
11. A guide to Telegram.Bot library / <https://telegrambots.github.io/book/> / (дата обращения: 01.05.2025)
12. Васильев А. / Программирование на C# для начинающих / ISBN: 978-5-9775-0943-5
13. Шилдт Г. / Java. Полное руководство / ISBN: 978-5-8459-1693-4
14. Донован А., Керниган Б. / Язык программирования Go / ISBN: 978-5-8459-1857-0
15. Котеров А., Шафрановский И. / PHP 7. Искусство разработки / ISBN: 978-5-8459-1859-4
16. PostgreSQL Documentation / <https://www.postgresql.org/docs/> / (дата обращения: 08.05.2025)

17. MySQL Reference Manual / <https://dev.mysql.com/doc/refman/8.0/en/> /
(дата обращения: 01.05.2025)

18. Microsoft SQL Server Documentation / <https://docs.microsoft.com/ru-ru/sql/sql-server/> / (дата обращения: 08.05.2025)

					ПГУ 1.090301.26.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		69

Приложение А

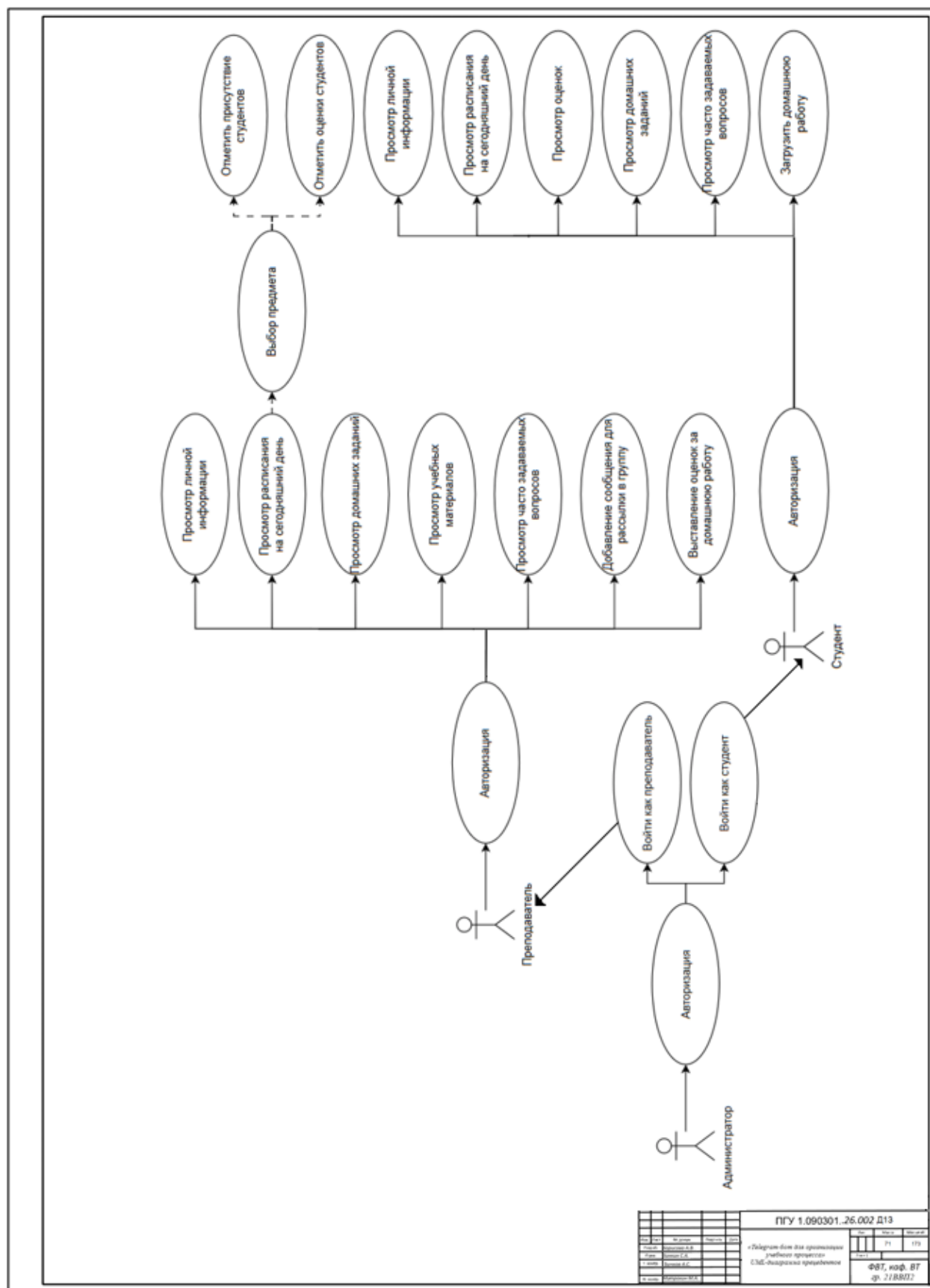
(Обязательное)

Telegram-бот для организации учебного процесса

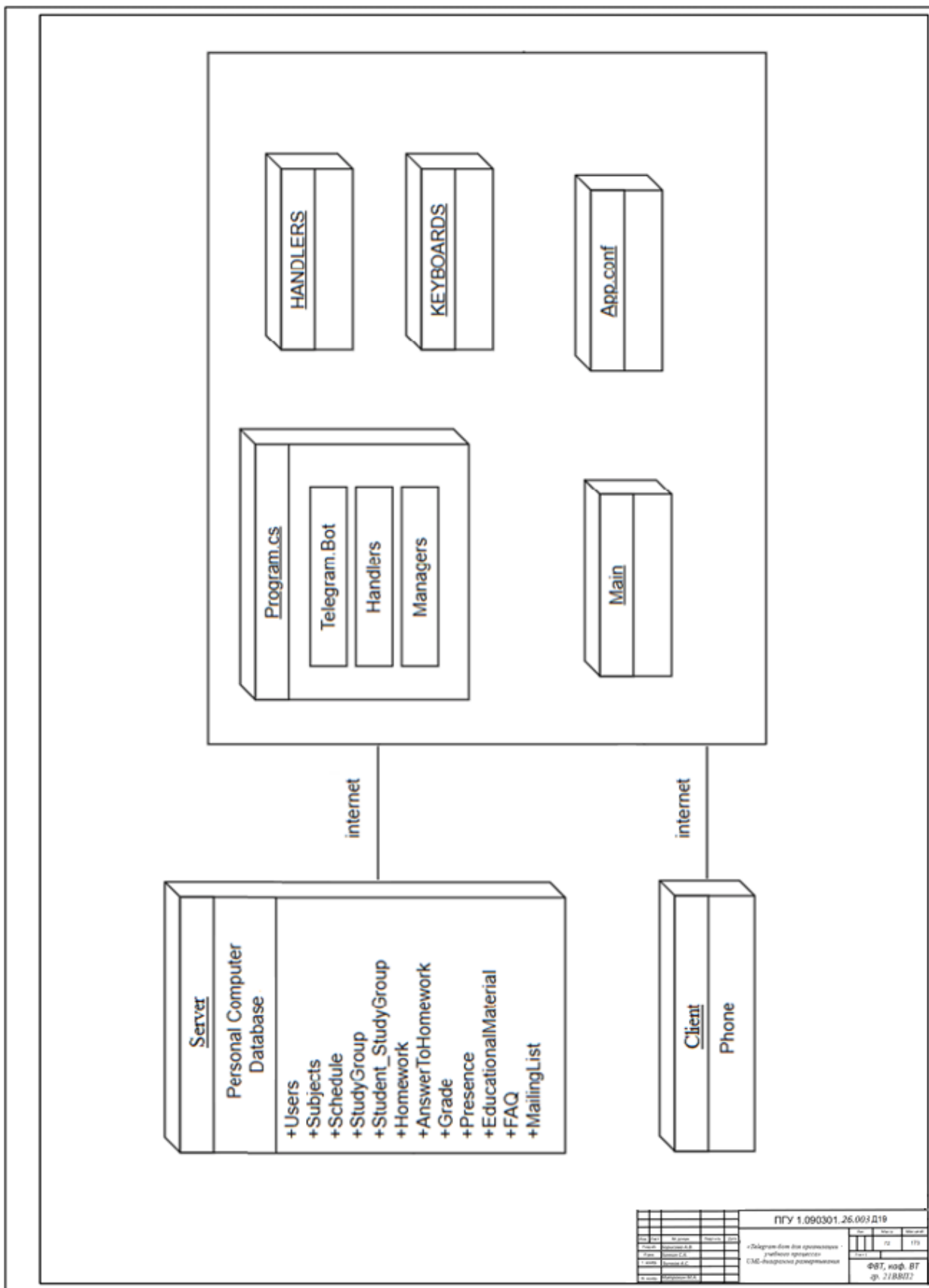
UML-диаграммы

					ПГУ 1.090301.26.001 ПЗ	Лист
						70
Изм.	Лист	№ докум	Подпись	Дата		

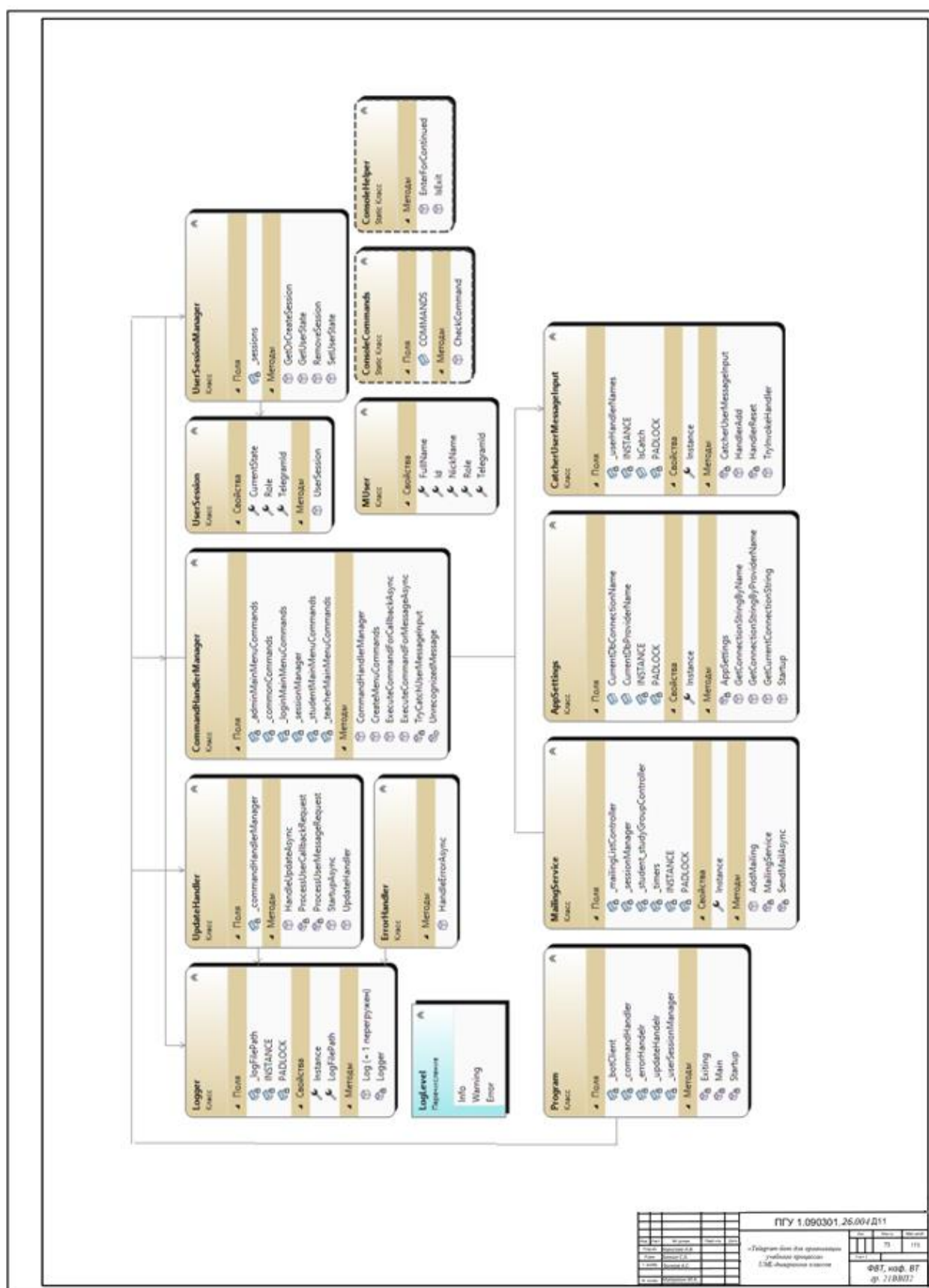
Приложение А.1 - Диаграмма прецедентов



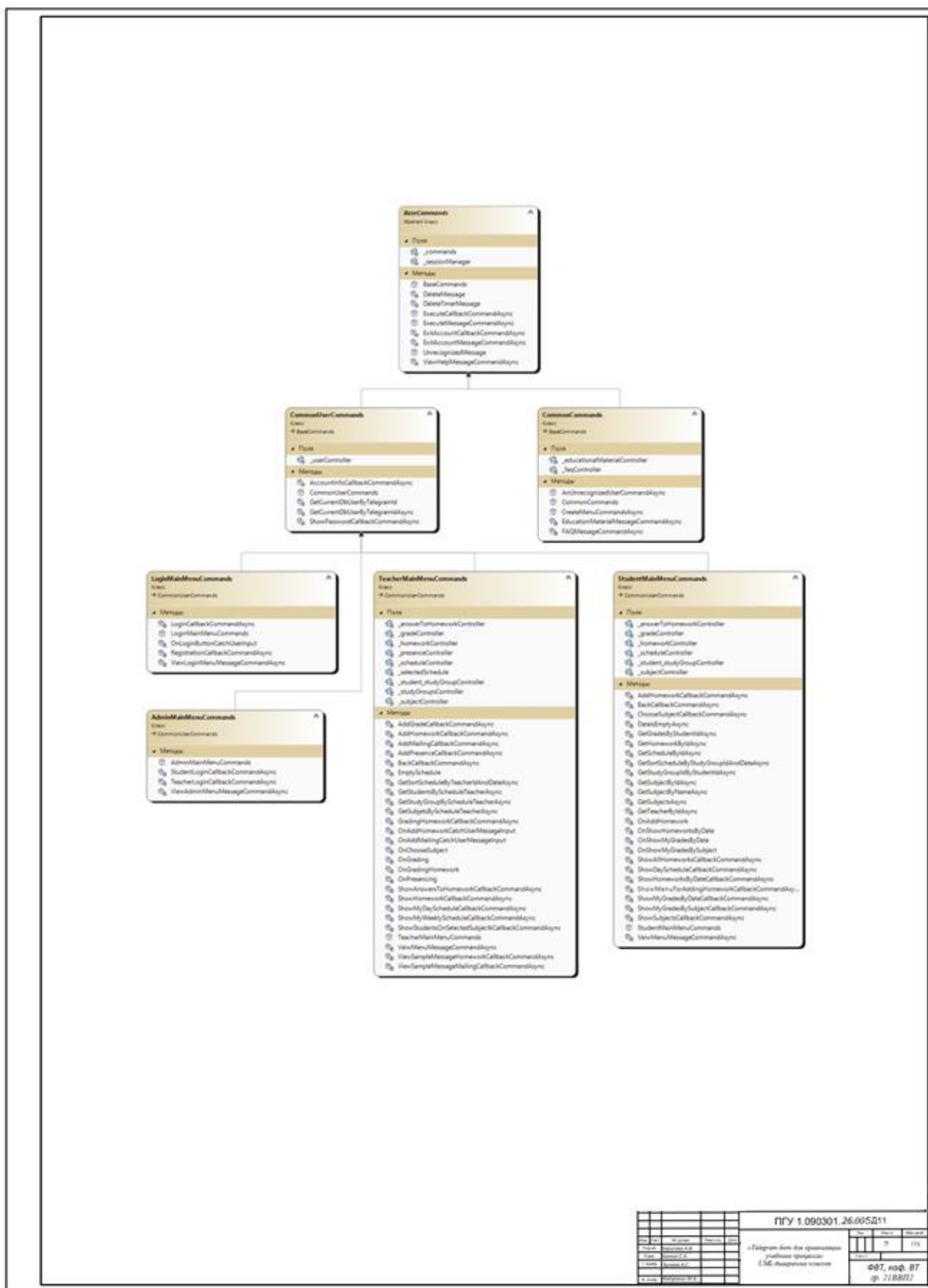
Приложение А.2 - Диаграмма развертывания



Приложение А.3 - Диаграмма классов



Приложение А.4 - Диаграмма классов



Приложение Б

(Обязательное)

Telegram-бот для организации учебного процесса

Листинг программы

					ПГУ 1.090301.26.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		75

Приложение Б.1

Файл Program.cs

```
using System.Configuration;
using Telegram.Bot;
using Telegram.Bot.Polling;
using Telegram.Bot.Types.Enums;
using TelegramChatBot.Handlers;
using TelegramChatBot.Managers;
using Utils;

namespace TelegramChatBot.Main
{
    internal class Program
    {
        private static UpdateHandler _updateHandler;
        private static ErrorHandler _errorHandler;
        private static TelegramBotClient _botClient;

        private static UserSessionManager _userSessionManager;
        private static CommandHandlerManager _commandHandler;

        static async Task Main(string[] args)
        {
            Startup();

            var me = await _botClient.GetMe();
            var receiverOptions = new ReceiverOptions
            {
                AllowedUpdates = new[] { UpdateType.Message,
                UpdateType.CallbackQuery },
                DropPendingUpdates = true
            };
            var cts = new CancellationTokenSource();

            try
            {
                _botClient.StartReceiving(_updateHandler.HandleUpdateAsync,
                _errorHandler.HandleErrorAsync, receiverOptions, cts.Token);
                await _updateHandler.StartupAsync(_botClient);

                Logger.Instance.Log($"Бот {me.Username} запущен
                ...");
            }
            catch (Exception ex)
            {
                Logger.Instance.Log(ex.Message, LogLevel.Error);
            }
        }
    }
}
```

```

        while (!ConsoleHelper.IsExit()) ;
        Exiting();
    }

    private static void Startup()
    {
        string token =
ConfigurationManager.ConnectionStrings["TelegramBotToken"].Conne
ctionString;
        _botClient = new(token);

        _userSessionManager = new();
        _commandHandler = new(_userSessionManager);

        _updateHandelr = new(_commandHandler);
        _errorHandelr = new();
    }

    private static void Exiting()
    {
        Console.WriteLine("Command: stop called");
        Console.WriteLine("Startup exit ...");

        // do for needed for exit

        Console.WriteLine("Exited");
    }
}

```

Приложение Б.2

Файл UpdateHandler.cs

```

using Telegram.Bot.Types;
using Telegram.Bot;
using TelegramChatBot.Managers;
using Utils;

namespace TelegramChatBot.Handlers
{
    public class UpdateHandler
    {
        private CommandHandlerManager? _commandHandlerManager =
null;

        public UpdateHandler(CommandHandlerManager
commandHandler)
        {
            _commandHandlerManager = commandHandler;
        }
    }
}

```

```

        private async Task ProcessUserMessageRequest(long userId,
ITelegramBotClient botClient, Update update)
        {
            var t_chat = update.Message.Chat;
            var t_user = update.Message.From;

            Logger.Instance.Log($"{t_user.FirstName}
({t_user.Id}) написал сообщение: {update.Message.Text}");

            try
            {
                await
_commandHandlerManager.ExecuteCommandForMessageAsync(userId,
botClient, update);
            }
            catch (Exception ex)
            {

Logger.Instance.Log($"UpdateHandler.ProcessUserMessageRequest (Ex
ception): {ex.Message}", LogLevel.Error);

                await botClient.SendMessage(
                    t_chat.Id,
                    "Возникла ошибка на сервере ...")
                ;
            }
        }

        private async Task ProcessUserCallbackRequest(long
userId, ITelegramBotClient botClient, Update update)
        {
            var callbackQuery = update.CallbackQuery;
            await
botClient.AnswerCallbackQuery(callbackQuery.Id);

            var t_user = callbackQuery.From;
            var t_chat = update.CallbackQuery.Message.Chat;
            var callbackData = callbackQuery.Data;

            Logger.Instance.Log($"{t_user.Username} ({t_user.Id})
нажал кнопку: {callbackData}");

            try
            {
                await
_commandHandlerManager.ExecuteCommandForCallbackAsync(userId,
botClient, update);
            }
            catch (Exception ex)
            {

```

```

Logger.Instance.Log($"UpdateHandler.ProcessUserCallbackRequest (E
xception): {ex.Message}", LogLevel.Error);

        await botClient.SendMessage(
            t_chat.Id,
            "Возникла ошибка на сервере ..."
        );
    }
}

public async Task StartupAsync(ITelegramBotClient
botClient)
{
    await
_commandHandlerManager.CreateMenuCommands(botClient);
}

public async Task HandleUpdateAsync(ITelegramBotClient
botClient, Update update, CancellationToken token)
{
    if (update.Message is not null)
    {
        long userId = update.Message.From.Id;
        // var t_chat = update.Message.Chat;

        await Task.Run(() =>
ProcessUserMessageRequest(userId, botClient, update));
    }
    else if (update.CallbackQuery is not null)
    {
        long userId = update.CallbackQuery.From.Id;
        // var t_chat =
update.CallbackQuery.Message.Chat;

        await Task.Run(() =>
ProcessUserCallbackRequest(userId, botClient, update));
    }
    else
    {
        var t_chat = update.Message.Chat;
        await
_commandHandlerManager.UnrecognizedMessage(botClient, t_chat);
    }
}
}
}

```

Приложение Б.3

Файл ErrorHandler.cs

```

using Telegram.Bot.Exceptions;
using Telegram.Bot.Polling;
using Telegram.Bot;
using Utils;

namespace TelegramChatBot.Handlers
{
    public class ErrorHandler
    {
        public async Task HandleErrorAsync(ITelegramBotClient
client,      Exception exception,      HandleErrorSource source,
CancellationToken token)
        {
            try
            {
                var ErrorMessage = exception switch
                {
                    ApiRequestException apiRequestException
                        => $"Telegram API
Error:\n[{apiRequestException.ErrorCode}]\n{apiRequestException.
Message}",
                    _ => exception.ToString()
                };

                Logger.Instance.Log(ErrorMessage,
LogLevel.Error);
            }
            catch (Exception ex)
            {
                throw new Exception($"ErrorHandler (Exception) :
{ex.Message}");
            }
        }
    }
}

```

Приложение Б.4

Файл CommandHandlerManager.cs

```

using Teaching_Assistant.Commands.User;
using Telegram.Bot;
using Telegram.Bot.Types;
using TelegramChatBot.Commands;
using TelegramChatBot.Global;

namespace TelegramChatBot.Managers
{
    public class CommandHandlerManager
    {
        private readonly UserSessionManager _sessionManager;
    }
}

```

```

        private readonly CommonCommands _commonCommands;
        private readonly LoginMainMenuCommands
_loginMainMenuCommands;
        private readonly AdminMainMenuCommands
_adminMainMenuCommands;
        private readonly StudentMainMenuCommands
_studentMainMenuCommands;
        private readonly TeacherMainMenuCommands
_teacherMainMenuCommands;

        public CommandHandlerManager (UserSessionManager
sessionManager)
        {
            _sessionManager = sessionManager;

            _commonCommands = new(sessionManager);
            _loginMainMenuCommands = new(sessionManager);
            _adminMainMenuCommands = new(sessionManager);
            _studentMainMenuCommands = new(sessionManager);
            _teacherMainMenuCommands = new(sessionManager);
        }

        private async Task<bool> TryCatchUserMessageInput (long
userId, ITelegramBotClient botClient, Message message)
        {
            return await
CatcherUserMessageInput.Instance.TryInvokeHandler(userId,
botClient, message);
        }

        public async Task ExecuteCommandForMessageAsync (long
userId, ITelegramBotClient botClient, Update update)
        {
            var message = update.Message;
            if (message == null)
            {
                var t_chat = message.Chat;
                await
_commonCommands.UnrecognizedMessage(botClient, t_chat);
                return;
            }

            if (CatcherUserMessageInput.Instance.IsCatch)
            {
                if (await TryCatchUserMessageInput (userId,
botClient, message))
                    return;
            }

            var userSession =
_sessionManager.GetOrCreateSession(userId);

```

```

        if (update.Message.Text == CommandNames.FAQ ||
update.Message.Text == CommandNames.EDUCATION_MATERIAL)
        {
            await
_commonCommands.ExecuteMessageCommandAsync(userId, botClient,
update);
            return;
        }

        if (userSession.Role == RoleNames.GUEST &&
userSession.CurrentState == CurrentStateNames.START)
        {
            await
_loginMainMenuCommands.ExecuteMessageCommandAsync(userId,
botClient, update);
        }
        else if (userSession.Role == RoleNames.ADMIN &&
userSession.CurrentState == CurrentStateNames.MAIN_MENU)
        {
            await
_adminMainMenuCommands.ExecuteMessageCommandAsync(userId,
botClient, update);
        }
        else if (userSession.Role == RoleNames.TEACHER &&
userSession.CurrentState == CurrentStateNames.MAIN_MENU)
        {
            await
_teacherMainMenuCommands.ExecuteMessageCommandAsync(userId,
botClient, update);
        }
        else if (userSession.Role == RoleNames.STUDENT &&
userSession.CurrentState == CurrentStateNames.MAIN_MENU)
        {
            await
_studentMainMenuCommands.ExecuteMessageCommandAsync(userId,
botClient, update);
        }
        else
        {
            await
_commonCommands.AnUnrecognizedUserCommandAsync(userId, botClient,
update);
            return;
        }
    }

    public async Task ExecuteCommandForCallbackAsync(long
userId, ITelegramBotClient botClient, Update update)
    {
        var callback = update.CallbackQuery;
        if (callback == null)

```

```

        return;

        string data = callback.Data;
        if (!data.StartsWith(ButtonNames.START_NAME_BUTTON))
        {
            var t_chat = callback.Message.Chat;
            await
            _commonCommands.UnrecognizedMessage(botClient, t_chat);
            return;
        }

        var userSession =
        _sessionManager.GetOrCreateSession(userId);

        if (userSession.Role == RoleNames.GUEST &&
            userSession.CurrentState == CurrentStateNames.START)
        {
            await
            _loginMainMenuCommands.ExecuteCallbackCommandAsync(userId,
            botClient, update);
        }
        else if (userSession.Role == RoleNames.ADMIN &&
            userSession.CurrentState == CurrentStateNames.MAIN_MENU)
        {
            await
            _adminMainMenuCommands.ExecuteCallbackCommandAsync(userId,
            botClient, update);
        }
        else if (userSession.Role == RoleNames.TEACHER &&
            userSession.CurrentState == CurrentStateNames.MAIN_MENU)
        {
            await
            _teacherMainMenuCommands.ExecuteCallbackCommandAsync(userId,
            botClient, update);
        }
        else if (userSession.Role == RoleNames.STUDENT &&
            userSession.CurrentState == CurrentStateNames.MAIN_MENU)
        {
            await
            _studentMainMenuCommands.ExecuteCallbackCommandAsync(userId,
            botClient, update);
        }
        else
        {
            await
            _commonCommands.AnUnrecognizedUserCommandAsync(userId, botClient,
            update);
        }
    }

    public async Task CreateMenuCommands(ITelegramBotClient
    botClient)

```

```

        {
            await
            _commonCommands.CreateMenuCommandsAsync(botClient);
        }

        internal                                async                                Task
        UnrecognizedMessage(ITelegramBotClient botClient, Chat t_chat)
        {
            await _commonCommands.UnrecognizedMessage(botClient,
            t_chat);
        }
    }
}

```

Приложение Б.5

Файл UserSessionManager.cs

```

using System.Collections.Concurrent;
using TelegramChatBot.Models;

namespace TelegramChatBot.Managers
{
    /// <summary>
    /// Класс для хранения состояний пользователей
    /// </summary>
    public class UserSessionManager
    {
        private readonly ConcurrentDictionary<long, UserSession>
        _sessions = new();

        public UserSession GetOrCreateSession(long userId)
        {
            return _sessions.GetOrAdd(userId, id => new
            UserSession(id));
        }

        public void RemoveSession(long userId)
        {
            _sessions.TryRemove(userId, out _);
        }

        public string GetUserState(long userId)
        {
            return _sessions.TryGetValue(userId, out var state) ?
            state.CurrentState : "default";
        }

        public void SetUserState(long userId, string state)
        {
            _sessions[userId].CurrentState = state;
        }
    }
}

```

```
}
}
```

Приложение Б.6

Файл BaseCommands.cs

```
using Telegram.Bot.Types;
using Telegram.Bot;
using TelegramChatBot.Managers;
using TelegramChatBot.Global;

namespace TelegramChatBot.Commands
{
    public abstract class BaseCommands
    {
        protected readonly UserSessionManager _sessionManager;
        protected Dictionary<string, Func<long,
ITelegramBotClient, Update, Task>> _commands = null;

        public BaseCommands(UserSessionManager sessionManager)
        {
            _sessionManager = sessionManager;
        }

        protected virtual async Task
ViewHelpMessageCommandAsync(long userId, ITelegramBotClient
botClient, Update update)
        {
            var t_chat = update.Message.Chat;
            await botClient.SendMessage(
                t_chat.Id,
                "Данный чат-бот является тестовым!\n\n" +
                "Описание функционала:\n" +
                "... \n\n" +
                "Описание исполнителя:\n" +
                "... \n\n"
            );
        }

        protected virtual async Task
ExitAccountMessageCommandAsync(long userId, ITelegramBotClient
botClient, Update update)
        {
            await botClient.SendMessage(
                update.Message.Chat.Id,
                $"Вы вышли из учетной записи!\n" +
                $"Для просмотра доступных действий вызовите команду
/menu"
            );
        }
    }
}
```

```

        var userSession =
_sessionManager.GetOrCreateSession(userId);
        userSession.Role = RoleNames.GUEST;
        userSession.CurrentState =
CurrentStateNames.MAIN_MENU;
    }

    protected virtual async Task
ExitAccountCallbackCommandAsync(long userId, ITelegramBotClient
botClient, Update update)
    {
        await botClient.SendMessage(
            update.CallbackQuery.Message.Chat.Id,
            $"Вы вышли из учетной записи!\n" +
            $"Для просмотра доступных действий вызовите команду
/menu"
        );

        var userSession =
_sessionManager.GetOrCreateSession(userId);
        userSession.Role = RoleNames.GUEST;
        userSession.CurrentState =
CurrentStateNames.MAIN_MENU;
    }

    protected async Task DeleteTimerMessage(int
waitMilliseconds, ITelegramBotClient botClient, long chatId, int
messageId)
    {
        await Task.Delay(waitMilliseconds);
        await botClient.DeleteMessage(chatId, messageId);
    }

    protected async Task DeleteMessage(ITelegramBotClient
botClient, long chatId, int messageId)
    {
        await botClient.DeleteMessage(chatId, messageId);
    }

    public virtual async Task ExecuteMessageCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
    {
        var message = update.Message;

        if (_commands.TryGetValue(message.Text, out var
command))
        {
            await command(userId, botClient, update);
        }
        else
        {
            var t_chat = update.Message.Chat;

```

```

        await UnrecognizedMessage(botClient, t_chat);
    }
}

public virtual async Task
ExecuteCallbackCommandAsync(long userId, ITelegramBotClient
botClient, Update update)
{
    var callback = update.CallbackQuery;

    if (_commands.TryGetValue(callback.Data, out var
command))
    {
        await command(userId, botClient, update);
    }
    else
    {
        var t_chat = update.CallbackQuery.Message.Chat;
        await UnrecognizedMessage(botClient, t_chat);
    }
}

public async Task UnrecognizedMessage(ITelegramBotClient
botClient, Chat chat)
{
    await botClient.SendMessage(
        chat.Id,
        "Нераспознанный ввод!\n" +
        "Попробуйте другую команду или текст сообщения
...");
}
}

```

Приложение Б.7

Файл CommonCommands.cs

```

using Telegram.Bot.Types;
using Telegram.Bot;
using TelegramChatBot.Managers;
using TelegramChatBot.Global;
using Teaching_Assistant.MyORM.Controllers;
using MyORM.Global;
using Telegram.Bot.Types.ReplyMarkups;

namespace TelegramChatBot.Commands
{
    public class CommonCommands : BaseCommands
    {

```

```

        private DbEducationalMaterialController
        _educationalMaterialController;
        private DbFAQController _faqController;

        public CommonCommands (UserSessionManager sessionManager)
        : base(sessionManager)
        {
            _commands = new Dictionary<string, Func<long,
            ITelegramBotClient, Update, Task>>
            {
                { CommandNames.FAQ, FAQMessageCommandAsync },
                { CommandNames.EDUCATION_MATERIAL,
            EducationMaterialMessageCommandAsync }
            };

            _educationalMaterialController = new(
                AppSettings.Instance.CurrentDbProviderName,
                AppSettings.Instance.GetCurrentConnectionString()
            );

            _faqController = new(
                AppSettings.Instance.CurrentDbProviderName,
                AppSettings.Instance.GetCurrentConnectionString()
            );
        }

        private async Task
        EducationMaterialMessageCommandAsync (long userId,
        ITelegramBotClient botClient, Update update)
        {
            var t_chat = update.Message.Chat;

            await
            _educationalMaterialController.Query_SelectAllAsync();

            foreach (var mat in
            _educationalMaterialController.Data) {
                var inlineKeyboard = new InlineKeyboardMarkup(
                    new List<InlineKeyboardButton[]>()
                    {
                        new
                        InlineKeyboardButton[]
                        {
                            InlineKeyboardButton.WithUrl ("Просмотр источника",
                            mat.LinkToSource),
                        },
                    })
                );

            await botClient.SendMessage(
                t_chat.Id,

```

```

        $"{mat.Name}\n\n" +
        $"{mat.Description}\n",
        replyMarkup: inlineKeyboard
    );
}

_educationalMaterialController.Data.Clear();
}

private async Task FAQMessageCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
{
    var t_chat = update.Message.Chat;

    await _faqController.Query_SelectAllAsync();

    foreach (var faq in _faqController.Data)
    {
        await botClient.SendMessage(
            t_chat.Id,
            $"{faq.Name}\n\n" +
            $"{faq.Description}\n"
        );
    }

    _faqController.Data.Clear();
}

public async Task
CreateMenuCommandsAsync(ITelegramBotClient botClient)
{
    var bot_commands = new[]
    {
        new BotCommand { Command = "menu", Description =
"Показать меню" },
        new BotCommand { Command = "faq", Description =
"Часто задаваемые вопросы" },
        new BotCommand { Command = "educational_material",
Description = "Учебные материалы" },
        new BotCommand { Command = "help", Description =
"Помощь" },
        new BotCommand { Command = "exit", Description =
"Выйти из учетной записи чат-бота" }
    };

    await botClient.SetMyCommands(bot_commands);
}

public async Task AnUnrecognizedUserCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
{
    var t_chat = update.Message.Chat;

```

```

        await botClient.SendMessage (
            t_chat.Id,
            "Ошибка: ваша роля и текущий статус не распознан
... \n"
        );
    }
}
}

```

Приложение Б.8

Файл AdminMainMenuCommands.cs

```

using Telegram.Bot;
using Telegram.Bot.Types;
using Telegram.Bot.Types.ReplyMarkups;
using TelegramChatBot.Global;
using TelegramChatBot.Managers;
using TelegramChatBot.Models;

namespace Teaching_Assistant.Commands.User
{
    public class AdminMainMenuCommands : CommonUserCommands
    {
        public AdminMainMenuCommands (UserSessionManager
sessionManager) : base(sessionManager)
        {
            _commands = new Dictionary<string, Func<long,
ITelegramBotClient, Update, Task>>
            {
                { CommandNames.MENU,
ViewAdminMenuMessageCommandAsync },
                { CommandNames.HELP, ViewHelpMessageCommandAsync
},
                { CommandNames.EXIT,
ExitAccountMessageCommandAsync },
                { ButtonNames.BACK,
ExitAccountCallbackCommandAsync },
                { ButtonNames.EXIT,
ExitAccountCallbackCommandAsync },
                { ButtonNames.STUDENT_LOGIN,
StudentLoginCallbackCommandAsync },
                { ButtonNames.TEACHER_LOGIN,
TeacherLoginCallbackCommandAsync },
                { ButtonNames.ACCOUNT_INFO,
AccountInfoCallbackCommandAsync },
                { ButtonNames.SHOW_PASSWORD,
ShowPasswordCallbackCommandAsync }
            };
        }
    }
}

```

```

private async Task ViewAdminMenuMessageCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
{
    var t_chat = update.Message.Chat;

    var inlineKeyboard = new InlineKeyboardMarkup(
        new List<InlineKeyboardButton[]>()
        {
            new
InlineKeyboardButton[]
            {
                InlineKeyboardButton.WithCallbackData("Выйти", ButtonNames.BACK),
                InlineKeyboardButton.WithCallbackData("Личные данные",
ButtonNames.ACCOUNT_INFO),
            },
            new
InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Войти как студент",
ButtonNames.STUDENT_LOGIN),
            },
            new
InlineKeyboardButton[]
            {
                InlineKeyboardButton.WithCallbackData("Войти как преподаватель",
ButtonNames.TEACHER_LOGIN),
            },
        });

    await botClient.SendMessage(
        t_chat.Id,
        "Доступные действия:\n",
        replyMarkup: inlineKeyboard
    );
}

private async Task StudentLoginCallbackCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
{
    var t_chat = update.CallbackQuery.Message.Chat;

    UserSession session =
_sessionManager.GetOrCreateSession(userId);
    session.Role = RoleNames.STUDENT;
    session.CurrentState = CurrentStateNames.MAIN_MENU;

    await botClient.SendMessage(
        t_chat.Id,
        $"Вы вошли как студент!\n" +

```

```

        $"Для просмотра меню действий вызовите команду
{CommandNames.MENU}"
    );
}

private async Task TeacherLoginCallbackCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
{
    var t_chat = update.CallbackQuery.Message.Chat;

    UserSession session =
_sessionManager.GetOrCreateSession(userId);
    session.Role = RoleNames.TEACHER;
    session.CurrentState = CurrentStateNames.MAIN_MENU;

    await botClient.SendMessage(
        t_chat.Id,
        $"Вы вошли как преподаватель!\n" +
        $"Для просмотра меню действий вызовите команду
{CommandNames.MENU}"
    );
}
}
}

```

Приложение Б.9

Файл CommonUserCommands.cs

```

using MyORM.Controllers;
using Telegram.Bot.Types.ReplyMarkups;
using Telegram.Bot.Types;
using Telegram.Bot;
using TelegramChatBot.Commands;
using TelegramChatBot.Global;
using TelegramChatBot.Managers;
using MyORM.Models;
using MyORM.Global;
using Utils;

namespace Teaching_Assistant.Commands.User
{
    public class CommonUserCommands : BaseCommands
    {
        protected DbUserController _userController;

        public CommonUserCommands(UserSessionManager
sessionManager) : base(sessionManager)
        {
            _userController = new(
                AppSettings.Instance.CurrentDbProviderName,
                AppSettings.Instance.GetCurrentConnectionString()
            );
        }
    }
}

```

```

        );
    }

    protected async Task<DbUser>
GetCurrentDbUserByTelegramIdAsync(long telegramId)
    {
        DbUser result = await
_userController.Query_SelectByTelegramIdAsync(telegramId);
        return result;
    }

    protected DbUser GetCurrentDbUserByTelegramId(long
telegramId)
    {
        _userController.Query_TrySelectByTelegramId(telegramId, out
DbUser result);
        return result;
    }

    protected async Task
ShowPasswordCallbackCommandAsync(long userId, ITelegramBotClient
botClient, Update update)
    {
        var t_caht = update.CallbackQuery.Message.Chat;
        var current_user = await
GetCurrentDbUserByTelegramIdAsync(userId);

        await botClient.SendMessage(
            t_caht.Id,
            $"Сообщение с паролем будет удалено через 5
секунд"
        );

        Message message_with_pass = await
botClient.SendMessage(
            t_caht.Id,
            $"Ваш пароль: {current_user.Password}"
        );

        // Without waiting
        DeleteTimerMessage(5000, botClient, t_caht.Id,
message_with_pass.Id);
    }

    protected async Task AccountInfoCallbackCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
    {
        var t_caht = update.CallbackQuery.Message.Chat;
        var t_user = update.CallbackQuery.From;
    }

```

```

        if
        (!_userController.Query_TrySelectByTelegramId(userId, out DbUser
current_user))
        {

Logger.Instance.Log($"AdminMainMenuCommands.AccountInfoCallbackC
ommandAsync: пользователь ({userId}) не найден в БД",
LogLevel.Error);
            await botClient.SendMessage(
                t_caht.Id,
                $"Ошибка поиска пользователя в БД
...";

            );
            return;
        }

        var inlineKeyboard = new InlineKeyboardMarkup(
            new List<InlineKeyboardButton[]>()
            {
                new
                InlineKeyboardButton[]
                {
                    InlineKeyboardButton.WithCallbackData("Показать пароль",
                    ButtonNames.SHOW_PASSWORD),
                },
            },
            ));

        string hidden_password = "";
        foreach (var c in current_user.Password)
            hidden_password += "*";

        await botClient.SendMessage(
            t_caht.Id,
            $"Ваши личные данные: \n" +
            $"ID: {current_user.Id}\n" +
            $"TelegramId: {current_user.TelegramId}\n" +
            $"FullName: {current_user.FullName}\n" +
            $"NickName: {current_user.NickName}\n" +
            $"Password: {hidden_password}\n" +
            $"Role: {current_user.Role}",
            replyMarkup: inlineKeyboard
        );
    }
}
}

```

Приложение Б.10

Файл CommonUserCommands.cs

```
using Telegram.Bot.Types;
```

Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ

Лист

94

```

using Telegram.Bot;
using TelegramChatBot.Managers;
using Telegram.Bot.Types.ReplyMarkups;
using TelegramChatBot.Global;
using TelegramChatBot.Models;
using MyORM.Controllers;
using MyORM.Global;
using MyORM.Models;

namespace Teaching_Assistant.Commands.User
{
    public class LoginMainMenuCommands : CommonUserCommands
    {
        public LoginMainMenuCommands (UserSessionManager
sessionManager) : base(sessionManager)
        {
            _commands = new Dictionary<string, Func<long,
ITelegramBotClient, Update, Task>>
            {
                {
                    CommandNames.START,
ViewLoginMenuMessageCommandAsync },
                {
                    CommandNames.MENU,
ViewLoginMenuMessageCommandAsync },
                { CommandNames.HELP, ViewHelpMessageCommandAsync
},
                { ButtonNames.LOGIN, LoginCallbackCommandAsync },
                {
                    ButtonNames.REGISTRATION,
RegistrationCallbackCommandAsync }
            };
        }

        private async Task ViewLoginMenuMessageCommandAsync (long
userId, ITelegramBotClient botClient, Update update)
        {
            var t_chat = update.Message.Chat;

            var inlineKeyboard = new InlineKeyboardMarkup (
                new List<InlineKeyboardButton []> ()
                {
                    new
InlineKeyboardButton [] {
                        InlineKeyboardButton.WithCallbackData ("Вход", ButtonNames.LOGIN),
                    },
                    new
InlineKeyboardButton []
                    {
                        InlineKeyboardButton.WithCallbackData ("Регистрация",
ButtonNames.REGISTRATION),
                    }
                }
            );
        }
    }
}

```

```

    });

    await botClient.SendMessage(
        t_chat.Id,
        "Добро пожаловать в чат-бот \"Ассистент учебного
процесса\"!\n" +
        "Доступные действия:\n",
        replyMarkup: inlineKeyboard
    );
}

private async Task LoginCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
{
    var t_chat = update.CallbackQuery.Message.Chat;

    await botClient.SendMessage(
        t_chat.Id,
        $"Для входа отправьте ваш пароль от учетной
записи!\n"
    );

    CatcherUserMessageInput.Instance.HandlerAdd(userId,
OnLoginButtonCatchUserInput);
}

private async Task RegistrationCallbackCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
{
    var t_chat = update.CallbackQuery.Message.Chat;
    var t_user = update.CallbackQuery.From;

    await botClient.SendMessage(
        t_chat.Id,
        $"Для регистрации сообщите системному
администратору ваши следующие данные:\n" +
        $"TelegramId: {t_user.Id}\n" +
        $"FullName: ваше ФИО\n" +
        $"Telegram NickName: {t_user.Username}\n\n" +
        $"После регистрации системный администратор сообщит
вам данные для входа ..."
    );
}

private async Task OnLoginButtonCatchUserInput(long
userId, ITelegramBotClient botClient, Message message)
{
    var t_chat = message.Chat;
    var userSession =
_sessionManager.GetOrCreateSession(userId);
    var pass = message.Text;

```

```

        if (!_userController.Query_TryLogin(userId, pass, out
DbUser currentUser))
        {
            await botClient.SendMessage(
                t_chat.Id,
                $"Неверный пароль!\n" +
                $"Проверьте корректность ввода или
зарегистрируйтесь ..."
            );

            return;
        }

        userSession.Role = currentUser.Role;
        userSession.CurrentState =
CurrentStateNames.MAIN_MENU;

        string role = userSession.Role == RoleNames.STUDENT ?
"студент" : "преподаватель";
        role = userSession.Role == RoleNames.ADMIN ?
"администратор" : role;

        await botClient.SendMessage(
            t_chat.Id,
            $"Вход выполнен! " +
            $"Вы вошли как {role}.\n" +
            $"Для просмотра меню действий вызовите команду
{CommandNames.MENU}"
        );
    }
}

```

Приложение Б.11

Файл StudentMainMenuCommands.cs

```

using Telegram.Bot.Types.ReplyMarkups;
using Telegram.Bot.Types;
using Telegram.Bot;
using TelegramChatBot.Managers;
using TelegramChatBot.Global;
using Teaching_Assistant.MyORM.Models;
using Teaching_Assistant.MyORM.Controllers;
using MyORM.Global;
using MyORM.Models;
using Teaching_Assistant.Global;

namespace Teaching_Assistant.Commands.User
{
    public class StudentMainMenuCommands : CommonUserCommands

```

```

        {
            private DbStudent_StudyGroupController
            _student_studyGroupController;
            private DbSchedulesController _scheduleController;
            private DbSubjectController _subjectController;
            private DbGradeController _gradeController;
            private DbHomeworkController _homeworkController;
            private DbAnswerToHomeworkController
            _answerToHomeworkController;

            public StudentMainMenuCommands (UserSessionManager
            sessionManager) : base(sessionManager)
            {
                _commands = new Dictionary<string, Func<long,
                ITelegramBotClient, Update, Task>>
                {
                    { CommandNames.MENU, VeiwMenuMessageCommandAsync
                },
                    { CommandNames.HELP, ViewHelpMessageCommandAsync
                },
                    {
                        CommandNames.EXIT,
                        ExitAccountMessageCommandAsync },
                    {
                        ButtonNames.EXIT,
                        ExitAccountCallbackCommandAsync },
                    {
                        ButtonNames.BACK,
                        ExitAccountCallbackCommandAsync },
                    {
                        ButtonNames.ACCOUNT_INFO,
                        AccountInfoCallbackCommandAsync },
                    {
                        ButtonNames.SHOW_PASSWORD,
                        ShowPasswordCallbackCommandAsync },
                    {
                        ButtonNames.STUDENT_BACK,
                        BackCallbackCommandAsync },
                    {
                        ButtonNames.STUDENT_SHOW_DAY_SCHEDULE,
                        ShowDayScheduleCallbackCommandAsync },
                    {
                        ButtonNames.STUDENT_SHOW_MENU_FOR_ADDING_HOWEWORK,
                        ShowMenuForAddingHomeworkCallbackCommandAsync },
                    {
                        ButtonNames.STUDENT_SHOW_MY_GRADES_BY_DATE,
                        ShowMyGradesByDateCallbackCommandAsync },
                    {
                        ButtonNames.STUDENT_SHOW_MY_GRADES_BY_SUBJECT,
                        ShowMyGradesBySubjectCallbackCommandAsync },
                    {
                        ButtonNames.STUDENT_SHOW_SUBJECTS,
                        ShowSubjectsCallbackCommandAsync },
                    {
                        ButtonNames.STUDENT_CHOOSE_SUBJECT,
                        ChooseSubjectCallbackCommandAsync },
                    {
                        ButtonNames.STUDENT_SHOW_ALL_HOMEWORKS,
                        ShowAllHomeworksCallbackCommandAsync },
                    {
                        ButtonNames.STUDENT_SHOW_HOMEWORKS_BY_DATE,
                        ShowHomeworksByDateCallbackCommandAsync },
                    {
                        ButtonNames.STUDENT_ADD_HOWEWORK,
                        AddHomeworkCallbackCommandAsync },
                };
            }
        }
    
```

```

        _student_studyGroupController = new(
            AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
        );

        _scheduleController = new(
            AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
        );

        _subjectController = new(
            AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
        );

        _gradeController = new(
            AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
        );

        _homeworkController = new(
            AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
        );

        _answerToHomeworkController = new(
            AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
        );
    }

    #region === CallbackQuery Commands ===
    private async Task VeiwMenuMessageCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
    {
        var t_chat = update.Message.Chat;

        var inlineKeyboard = new InlineKeyboardMarkup(
            new List<InlineKeyboardButton[]>()
            {
                new
                InlineKeyboardButton[] {

```

```

InlineKeyboardButton.WithCallbackData("Личные данные",
ButtonNames.ACCOUNT_INFO),

InlineKeyboardButton.WithCallbackData("Выйти", ButtonNames.EXIT)
    },
    new

InlineKeyboardButton[]

    {

InlineKeyboardButton.WithCallbackData("Просмотр расписания на
сегодня", ButtonNames.STUDENT_SHOW_DAY_SCHEDULE),
    },
    new

InlineKeyboardButton[] {

InlineKeyboardButton.WithCallbackData("Добавить домашнее
задание", ButtonNames.STUDENT_SHOW_MENU_FOR_ADDING_HOWEWORK),
    },
    new

InlineKeyboardButton[] {

InlineKeyboardButton.WithCallbackData("Просмотр моих оценок по
дате", ButtonNames.STUDENT_SHOW_MY_GRADES_BY_DATE),
    },
    new

InlineKeyboardButton[] {

InlineKeyboardButton.WithCallbackData("Просмотр моих оценок по
предмету", ButtonNames.STUDENT_SHOW_MY_GRADES_BY_SUBJECT),
    }

    });

    await botClient.SendMessage(
        t_chat.Id,
        "Доступные действия:\n",
        replyMarkup: inlineKeyboard
    );
}

private async Task BackCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
{
    await botClient.SendMessage(
        update.CallbackQuery.Message.Chat.Id,
        $"Это пример #4 работы inline кнопок. Ты нажал на
кнопку: {update.CallbackQuery.Data}\n" +
        $"Вернемся назад, для просмотра доступных действий
ВЫЗОВИ команду \"/menu\"";
    );
}

```

```

        var userSession =
        _sessionManager.GetOrCreateSession(userId);
        userSession.Role = RoleNames.GUEST;
    }

    private async Task
    ShowDayScheduleCallbackCommandAsync(long userId,
    ITelegramBotClient botClient, Update update)
    {
        var t_chat = update.CallbackQuery.Message.Chat;
        int studentId =
        GetCurrentDbUserByTelegramId(userId).Id;
        string today = DateTime.Now.ToString("dd.MM.yyyy");

        int studyGroupId = await
        GetStudyGroupIdByStudentIdAsync(studentId);
        if (studyGroupId == -1)
        {
            await botClient.SendMessage(
                t_chat.Id,
                $"Ваша учебная группа не найдена!"
            );
            return;
        }

        List<DbSchedule> schedule = await
        GetSortScheduleByStudyGroupIdAndDateAsync(studyGroupId, today);
        if (schedule == null)
        {
            await DataIsEmptyAsync(botClient, t_chat);
            return;
        }
        else if (schedule.Count == 0)
        {
            await DataIsEmptyAsync(botClient, t_chat);
            return;
        }

        int i = 1;
        foreach (var item in schedule)
        {
            await botClient.SendMessage(
                t_chat.Id,
                $"Номер занятия: {i++}\n" +
                $"Предмет:
{GetSubjectByIdAsync(item.SubjectId).Result.Name}\n" +
                $"Преподаватель:
{GetTeacherByIdAsync(item.TeacherId).Result.FullName}\n"
            );
        }
    }

```

```

private async Task
ShowMyGradesByDateCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
{
    var t_chat = update.CallbackQuery.Message.Chat;

    await botClient.SendMessage(
        t_chat.Id,
        "Для просмотра своих оценок отправьте дату
(дд.мм.гггг).\n" +
        "Например: 01.01.2025"
    );

    CatcherUserMessageInput.Instance.HandlerAdd(userId,
OnShowMyGradesByDate);
}

private async Task
ShowMyGradesBySubjectCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
{
    var t_chat = update.CallbackQuery.Message.Chat;

    var inlineKeyboard = new InlineKeyboardMarkup(
        new List<InlineKeyboardButton[]>()
        {
            new InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Показать список
предметов", ButtonNames.STUDENT_SHOW_SUBJECTS),
            },
            new InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Ввести название предмета",
ButtonNames.STUDENT_CHOOSE_SUBJECT),
            }
        }
    );

    await botClient.SendMessage(
        t_chat.Id,
        "Для просмотра своих оценок необходимо название
предмета.\n" +
        "Доступные действия:",
        replyMarkup: inlineKeyboard
    );
}

private async Task ShowSubjectsCallbackCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
{
    var t_chat = update.CallbackQuery.Message.Chat;
    List<DbSubject> subjects = await GetSubjectsAsync();

```

```

        if (subjects == null)
        {
            await DataIsEmptyAsync(botClient, t_chat);
            return;
        }
        else if (subjects.Count == 0)
        {
            await DataIsEmptyAsync(botClient, t_chat);
            return;
        }

        string fullname_teacher;
        foreach (var item in subjects)
        {
            var teacher = await
            GetTeacherByIdAsync(item.TeacherId);
            fullname_teacher = teacher != null ?
            teacher.FullName : "не найден";

            await botClient.SendMessage(
                t_chat.Id,
                $"Код: {item.Id}\n" +
                $"Название: {item.Name}\n" +
                $"Преподаватель: {fullname_teacher}"
            );
        }

        var inlineKeyboard = new InlineKeyboardMarkup(
            new List<InlineKeyboardButton[]>()
            {
                new
                InlineKeyboardButton[] {
                    InlineKeyboardButton.WithCallbackData("Выбрать предмет для
                    просмотра оценки", ButtonNames.STUDENT_CHOOSE_SUBJECT),
                }
            }
        ));

        await botClient.SendMessage(
            t_chat.Id,
            "Доступные действия:",
            replyMarkup: inlineKeyboard
        );
    }

    private async Task
    ShowHomeworksByDateCallbackCommandAsync(long userId,
    ITelegramBotClient botClient, Update update)
    {
        var t_chat = update.CallbackQuery.Message.Chat;
    }

```

```

        await botClient.SendMessage(
            t_chat.Id,
            "Укажите дату начала выполнения домашней работы в
формате (дд.мм.гггг). " +
            "Дополнительно можно указать дату окончания через
пробел.\n" +
            "Например: 01.01.2025\n" +
            "Или: 01.01.2025 01.02.2025"
        );

        CatcherUserMessageInput.Instance.HandlerAdd(userId,
OnShowHomeworksByDate);
    }

    private async Task
ShowAllHomeworksCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
    {
        var t_chat = update.CallbackQuery.Message.Chat;
        int student_id =
GetCurrentDbUserByTelegramIdAsync(userId).Result.Id;
        int study_group_id = await
GetStudyGroupIdByStudentIdAsync(student_id);

        var homeworks = await
_homeworkController.Qyery_SelectByGroupIdAsync(study_group_id);

        if(homeworks == null)
        {
            await DataIsEmptyAsync(botClient, t_chat);
            return;
        }

        if (homeworks.Count == 0)
        {
            await DataIsEmptyAsync(botClient, t_chat);
            return;
        }

        foreach (var h in homeworks)
        {
            await botClient.SendMessage(
                t_chat.Id,
                $"Код: {h.Id}\n" +
                $"Название: {h.Name}\n" +
                $"Дата начала: {h.StartDate}\n" +
                $"Дата окончания: {h.EndDate}\n" +
                $"Ссылка для загрузки материалов:
{h.LinkToSource}\n"
            );
        }
    }

```

```

        var inlineKeyboard = new InlineKeyboardMarkup(
            new List<InlineKeyboardButton[]>()
            {
                new InlineKeyboardButton[] {
                    InlineKeyboardButton.WithCallbackData("Добавить",
                        ButtonNames.STUDENT_ADD_HOWEWORK)
                },
            }
        );

        await botClient.SendMessage(
            update.CallbackQuery.Message.Chat.Id,
            "Добавить ответ на домашнюю работу?",
            replyMarkup: inlineKeyboard
        );
    }

    private async Task ShowMenuForAddingHomeworkCallbackCommandAsync(long userId,
        ITelegramBotClient botClient, Update update)
    {
        var inlineKeyboard = new InlineKeyboardMarkup(
            new List<InlineKeyboardButton[]>()
            {
                new InlineKeyboardButton[] {
                    InlineKeyboardButton.WithCallbackData("Просмотр всех домашних
                        заданий", ButtonNames.STUDENT_SHOW_ALL_HOMEWORKS)
                },
                new InlineKeyboardButton[] {
                    InlineKeyboardButton.WithCallbackData("Просмотр домашних заданий
                        по дате", ButtonNames.STUDENT_SHOW_HOMEWORKS_BY_DATE)
                }
            }
        );

        await botClient.SendMessage(
            update.CallbackQuery.Message.Chat.Id,
            "Для добавления домашнего задания необходимо
                указать код домашнего задания, для этого: ",
            replyMarkup: inlineKeyboard
        );
    }

    private async Task ChooseSubjectCallbackCommandAsync(long
        userId, ITelegramBotClient botClient, Update update)
    {
        var t_chat = update.CallbackQuery.Message.Chat;

        await botClient.SendMessage(

```

```

        t_chat.Id,
        "Для просмотра оценок введите название или код
предмета.\n" +
        "Например: " +
        "Математика\n" +
        "Или: " +
        "111"
    );

    CatcherUserMessageInput.Instance.HandlerAdd(userId,
OnShowMyGradesBySubject);
    }

    private async Task AddHomeworkCallbackCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
    {
        await botClient.SendMessage(
            update.CallbackQuery.Message.Chat.Id,
            $"Для ввода ответа на домашнее задание укажите с
новой строки без цифр: \n" +
            $"1. Код домашнего задания\n" +
            $"2. Ссылку на материалы домашней работы\n" +
            $"Например: \n" +
            $"111\n" +
            $"{Links.LINK_TO_HOMEWORK}"
        );

        CatcherUserMessageInput.Instance.HandlerAdd(userId,
OnAddHomework);
    }
#endregion

#region === CatcherUserMessageInput Callbacks ===
    private async Task OnShowMyGradesByDate(long userId,
ITelegramBotClient botClient, Message message)
    {
        var t_chat = message.Chat;
        string date = message.Text;

        int student_id =
GetCurrentDbUserByTelegramIdAsync(userId).Result.Id;
        int study_group_id = await
GetStudyGroupIdByStudentIdAsync(student_id);

        try
        {
            if (!DateTime.TryParse(date, out _))
            {
                await botClient.SendMessage(
                    t_chat.Id,
                    "Некорректный ввод даты, необходим формат
ввода (дд.мм.гггг).\n" +

```

```

        "Например: 01.01.2025"
    );
    return;
}
}
catch
{
    await botClient.SendMessage(
        t_chat.Id,
        "Некорректный ввод"
    );
}

List<DbGrade> grades = await
GetGradesByStudentIdAsync(student_id);

if(grades == null)
{
    await DataIsEmptyAsync(botClient, t_chat);
    return;
}
else if(grades.Count == 0)
{
    await DataIsEmptyAsync(botClient, t_chat);
    return;
}

foreach (var grade in grades)
{
    var schedule = await
GetScheduleByIdAsync(grade.ScheduleId);
    var subject = await
GetSubjectByIdAsync(schedule.SubjectId);

    if (schedule.EventDate == date)
    {
        await botClient.SendMessage(
            t_chat.Id,
            $"Предмет: {subject.Name}\n" +
            $"Оценка: {grade.Value}\n" +
            $"Дата: {schedule.EventDate}"
        );
    }
}

private async Task OnShowMyGradesBySubject(long userId,
ITelegramBotClient botClient, Message message)
{
    var t_chat = message.Chat;
    string text = message.Text;

```

```

        string subject_name = "";

        int student_id =
GetCurrentDbUserByTelegramIdAsync(userId).Result.Id;
        int subject_id = -1;
        int study_group_id = await
GetStudyGroupIdByStudentIdAsync(student_id);

        try
        {
            if(!int.TryParse(text, out subject_id))
            {
                subject_id =
GetSubjectByNameAsync(text).Result.Id;
            }
        }
        catch
        {
            await botClient.SendMessage(
                t_chat.Id,
                "Такой предмет не найден ..."
            );
        }

        List<DbGrade> grades = await
GetGradesByStudentIdAsync(student_id);

        if (grades == null)
        {
            await DataIsEmptyAsync(botClient, t_chat);
            return;
        }
        else if (grades.Count == 0)
        {
            await DataIsEmptyAsync(botClient, t_chat);
            return;
        }

        foreach (var grade in grades){
            var schedule = await
GetScheduleByIdAsync(grade.ScheduleId);

            if (schedule.SubjectId == subject_id)
            {
                if (string.IsNullOrEmpty(subject_name))
                {
                    subject_name =
GetSubjectByIdAsync(schedule.SubjectId).Result.Name;
                }

                await botClient.SendMessage (
                    t_chat.Id,

```

```

        $"Предмет: {subject_name}\n" +
        $"Оценка: {grade.Value}\n" +
        $"Дата: {schedule.EventDate}"
    );
    }
}

private async Task OnShowHomeworksByDate(long userId,
ITelegramBotClient botClient, Message message)
{
    string date_from = "", date_to = "";

    var t_chat = message.Chat;
    int student_id =
GetCurrentDbUserByTelegramIdAsync(userId).Result.Id;
    int study_group_id = await
GetStudyGroupIdByStudentIdAsync(student_id);

    var inlineKeyboard_error = new InlineKeyboardMarkup(
        new List<InlineKeyboardButton[]>()
        {
            new InlineKeyboardButton[] {
InlineKeyboardButton.WithCallbackData("Повторить",
ButtonNames.STUDENT_SHOW_HOMEWORKS_BY_DATE)
            },
        });

    try
    {
        date_from = message.Text.Split(' ')[0];

        if (message.Text.Split(' ').Length == 2)
            date_to = message.Text.Split(' ')[1];
        else if (message.Text.Split(' ').Length > 2)
            throw new Exception();

        DateTime checker_format =
DateTime.Parse(date_from);
        if (date_to != "")
            checker_format = DateTime.Parse(date_to);
    }
    catch
    {
        await botClient.SendMessage(
            t_chat.Id,
            "Некорректный ввод, необходимо ввести одну
или две даты формата (дд.мм.гггг).\n" +
            "Повторить ввод даты?",
            replyMarkup: inlineKeyboard_error
        );
    }
}

```

```

        return;
    }

    var homeworks = await
_homeworkController.Query_SelectByGroupIdAndDateAsync(study_group_id, date_from, date_to);

    if (homeworks == null)
    {
        await DataIsEmptyAsync(botClient, t_chat);
        return;
    }

    if (homeworks.Count == 0)
    {
        await DataIsEmptyAsync(botClient, t_chat);
        return;
    }

    foreach (var h in homeworks)
    {
        await botClient.SendMessage(
            t_chat.Id,
            $"Код: {h.Id}\n" +
            $"Название: {h.Name}\n" +
            $"Дата начала: {h.StartDate}\n" +
            $"Дата окончания: {h.EndDate}\n" +
            $"Ссылка для загрузки материалов:
{h.LinkToSource}\n"
        );
    }

    var inlineKeyboard = new InlineKeyboardMarkup(
        new List<InlineKeyboardButton[]>()
        {
            new InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Добавить",
                ButtonNames.STUDENT_ADD_HOWEWORK)
            },
        }
    );

    await botClient.SendMessage(
        message.Chat.Id,
        "Добавить ответ на домашнюю работу?",
        replyMarkup: inlineKeyboard
    );
}

private async Task OnAddHomework(long userId,
ITelegramBotClient botClient, Message message)

```

```

{
    var t_chat = message.Chat;
    int homework_id;
    int student_id =
GetCurrentDbUserByTelegramIdAsync(userId).Result.Id;
    string link_to_answer = "";
    string today = DateTime.Now.ToString("dd.MM.yyyy");

    var inlineKeyboard = new InlineKeyboardMarkup(
        new List<InlineKeyboardButton[]>()
        {
            new InlineKeyboardButton[] {

InlineKeyboardButton.WithCallbackData("Указать домашнюю работу",
ButtonNames.STUDENT_ADD_HOWEWORK)
            },
        });

    try
    {
        homework_id =
int.Parse(message.Text.Split('\n')[0]);
        link_to_answer = message.Text.Split('\n')[1];
    }
    catch
    {
        await botClient.SendMessage(
            t_chat.Id,
            "Некорректный ввод. Попробовать снова?",
            replyMarkup: inlineKeyboard
        );

        return;
    }

    if (GetHomeworkByIdAsync(homework_id) == null)
    {
        await botClient.SendMessage(
            t_chat.Id,
            $"Код домашней работы - {homework_id} не
найден. Попробовать снова?",
            replyMarkup: inlineKeyboard
        );

        return;
    }

    if (await
_answerToHomeworkController.Query_InsertAsync(new
DbAnswerToHomework(0, homework_id, student_id, 0, today,
link_to_answer)) == false)
    {

```

```

        await botClient.SendMessage(
            t_chat.Id,
            $"Ошибка добавления домашней работы.
Попробовать снова?",
            replyMarkup: inlineKeyboard
        );

        return;
    }

    _answerToHomeworkController.Data.Clear();

    await botClient.SendMessage(
        t_chat.Id,
        "Ответ добавлен. Указать еще домашнюю работу?",
        replyMarkup: inlineKeyboard
    );
}
#endregion

#region === Methods ===
private async Task DataIsEmptyAsync(ITelegramBotClient
botClient, Chat t_chat)
{
    await botClient.SendMessage(
        t_chat.Id,
        $"Данные не найдены!"
    );
}

private async Task<int>
GetStudyGroupIdByStudentIdAsync(int studentId)
{
    DbStudent_StudyGroup result = await
_student_studyGroupController.Query_SelectByStudentIdAsync(stude
ntId);

    int id = -1;
    if (result != null)
        id = result.Id;

    return id;
}

private async Task<List<DbSchedule>>
GetSortScheduleByStudyGroupIdAndDateAsync(int studyGroupId,
string date)
{
    List<DbSchedule> result = await
_scheduleController.Query_TrySelectByStudyGroupIdAndDateAsync(st
udyGroupId, date);
    if (result == null)

```

```

        return null;

        result = result.OrderBy(x =>
x.LessonNumber).ToList();

        return result;
    }

    private async Task<DbSubject> GetSubjectByIdAsync(int
subjectId)
    {
        DbSubject result = await
_subjectController.Query_SelectByIdAsync(subjectId);
        return result;
    }

    private async Task<DbSubject>
GetSubjectByNameAsync(string name)
    {
        DbSubject result = await
_subjectController.Query_SelectByNameAsync(name);
        return result;
    }

    private async Task<DbUser> GetTeacherByIdAsync(int
teacherId)
    {
        DbUser result = await
_userController.Query_SelectByIdAsync(teacherId);
        return result;
    }

    private async Task<List<DbGrade>>
GetGradesByStudentIdAsync(int stidentId)
    {
        List<DbGrade> result = await
_gradeController.Query_SelectsByStudentId(stidentId);
        return result;
    }

    private async Task<DbSchedule> GetScheduleByIdAsync(int
id)
    {
        var result = await
_scheduleController.Query_SelectByIdAsync(id);
        return result;
    }

    private async Task<List<DbSubject>> GetSubjectsAsync()
    {
        string query = $"select * from
{_subjectController.TableName}";

```

```

        List<DbSubject> result =
        _subjectController.Query_SelectsByQuery(query);

        return result;
    }

    private async Task<DbHomework> GetHomeworkByIdAsync(int
id)
    {
        DbHomework result = await
        _homeworkController.Query_SelectByIdAsync(id);
        return result;
    }
    #endregion
}
}

```

Приложение Б.12

Файл Program.cs

```

using Telegram.Bot.Types.ReplyMarkups;
using Telegram.Bot.Types;
using Telegram.Bot;
using TelegramChatBot.Managers;
using TelegramChatBot.Global;
using Teaching_Assistant.MyORM.Controllers;
using MyORM.Global;
using Teaching_Assistant.MyORM.Models;
using Teaching_Assistant.Service;
using Teaching_Assistant.Global;
using MyORM.Controllers;
using System.Collections.Concurrent;
using Microsoft.AspNetCore.Mvc.Diagnostics;
using MyORM.Models;

namespace Teaching_Assistant.Commands.User
{
    public class TeacherMainMenuCommands : CommonUserCommands
    {
        private DbSubjectController _subjectController;
        private DbSchedulesController _scheduleController;
        private DbStudyGroupsController _studyGroupsController;
        private DbHomeworkController _homeworkController;
        private DbAnswerToHomeworkController
        _answerToHomeworkController;
        private DbStudent_StudyGroupController
        _student_studyGroupController;
        private DbPresenceController _presenceController;
        private DbGradeController _gradeController;
    }
}

```

```

        private ConcurrentDictionary<long, DbSchedule>
        _selectedSchedule;

        public TeacherMainMenuCommands (UserSessionManager
        sessionManager) : base(sessionManager)
        {
            _commands = new Dictionary<string, Func<long,
            ITelegramBotClient, Update, Task>>
            {
                { CommandNames.MENU, VeiwMenuMessageCommandAsync
            },
                { CommandNames.HELP, ViewHelpMessageCommandAsync
            },
                {
                    CommandNames.EXIT,
                ExitAccountMessageCommandAsync },
                {
                    ButtonNames.EXIT,
                ExitAccountCallbackCommandAsync },
                {
                    ButtonNames.BACK,
                ExitAccountCallbackCommandAsync },
                {
                    ButtonNames.ACCOUNT_INFO,
                AccountInfoCallbackCommandAsync },
                {
                    ButtonNames.SHOW_PASSWORD,
                ShowPasswordCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_SHOW_DAY_SCHEDULE,
                ShowMyDayScheduleCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_SHOW_WEEKLY_SCHEDULE,
                ShowMyWeeklyScheduleCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_BACK,
                BackCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_SHOW_HOWEWORK,
                ShowHomeworkCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_ADD_HOWEWORK,
                AddHomeworkCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_SHOW_ANSWERS_TO_HOWEWORK,
                ShowAnswersToHomeworkCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_ADD_MAILING,
                AddMailingCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_VIEW_SAMPLE_MESSAGE_MAILING,
                ViewSampleMessageMailingCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_VIEW_SAMPLE_MESSAGE_HOMEWORK,
                ViewSampleMessageHomeworkCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_SHOW_STUDENTS_ON_SELECTED_SCHUDLE,
                ShowStudentsOnSelectedSubjectkCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_YES_GRADING,
                GradingHomeworkCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_ADD_PRESENCE,
                AddPresenceCallbackCommandAsync },
                {
                    ButtonNames.TEACHER_ADD_GRADE,
                AddGradeCallbackCommandAsync },
            }
        }
    }

```

```

};

_selectedSchedule = new();

_subjectController = new(
    AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
);

_scheduleController = new(
    AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
);

_studyGroupsController = new(
    AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
);

_homeworkController = new(
    AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
);

_answerToHomeworkController = new(
    AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
);

_student_studyGroupController = new(
    AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
);

_presenceController = new(
    AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
);

_gradeController = new(
    AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
);
}

```

```

#region === CallbackQuery Commands ===
private async Task VeiwMenuMessageCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
{
    var t_chat = update.Message.Chat;

    var inlineKeyboard = new InlineKeyboardMarkup(
        new List<InlineKeyboardButton[]>()
        {
            new
            InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Личные данные",
                ButtonNames.ACCOUNT_INFO),
                InlineKeyboardButton.WithCallbackData("Выйти", ButtonNames.EXIT)
            },
            new
            InlineKeyboardButton[]
            {
                InlineKeyboardButton.WithCallbackData("Просмотр расписания на
                сегодня", ButtonNames.TEACHER_SHOW_DAY_SCHEDULE),
            },
            new
            InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Просмотр домашних
                заданий", ButtonNames.TEACHER_SHOW_HOWEWORK),
            },
            new
            InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Добавить домашнее
                задание", ButtonNames.TEACHER_ADD_HOWEWORK),
            },
            new
            InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Просмотр ответов на
                домашние задания", ButtonNames.TEACHER_SHOW_ANSWERS_TO_HOWEWORK),
            },
            new
            InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Добавить сообщение для
                рассылки", ButtonNames.TEACHER_ADD_MAILING),
            }
        }
    ));

```

```

        await botClient.SendMessage(
            t_chat.Id,
            "Доступные действия:\n",
            replyMarkup: inlineKeyboard
        );
    }

    private async Task AddPresenceCallbackCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
    {
        var t_chat = update.CallbackQuery.Message.Chat;
        await botClient.SendMessage(
            t_chat.Id,
            $"Укажите код присутствующих студентов с новой
строки.\n" +
            $"Например:\n" +
            $"111\n" +
            $"112\n" +
            $"113\n" +
            $"114"
        );

        CatcherUserMessageInput.Instance.HandlerAdd(userId,
OnPresencing);
    }

    private async Task AddGradeCallbackCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
    {
        var t_chat = update.CallbackQuery.Message.Chat;
        await botClient.SendMessage(
            t_chat.Id,
            $"Укажите код студента(ов) и требуемую оценку
через пробел. " +
            $"Если требуется указать несколько студентов, то
код каждого студента необходимо указать с новой строки\n" +
            $"Например:\n" +
            $"111 5\n" +
            $"Или\n" +
            $"112 5\n" +
            $"113 5\n" +
            $"114 4"
        );

        CatcherUserMessageInput.Instance.HandlerAdd(userId,
OnGrading);
    }

    private async Task
ShowStudentsOnSelectedSubjectkCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
    {

```

```

        var t_chat = update.CallbackQuery.Message.Chat;
        var studednts = GetStudentsByScheduleTeacherAsync(userId).Result;

        if(studednts == null)
        {
            await botClient.SendMessage(
                t_chat.Id,
                "Студенты не найдены!"
            );
        }

        foreach (var student in studednts) {
            await botClient.SendMessage(
                t_chat.Id,
                $"Код студента: {student.Id}\n" +
                $"ФИО: {student.FullName}\n"
            );
        }

        await botClient.SendMessage(
            t_chat.Id,
            $"Укажите код присутствующих студентов с новой
строки.\n" +
            $"Например:\n" +
            $"111\n" +
            $"112\n" +
            $"113\n" +
            $"114"
        );

        CatcherUserMessageInput.Instance.HandlerAdd(userId,
OnPresencing);
    }

    private async Task
GradingHomeworkCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
    {
        var t_chat = update.CallbackQuery.Message.Chat;

        await botClient.SendMessage(
            t_chat.Id,
            $"Для оценки работы введите: код работы и
оценку\n" +
            $"Например: 1 5"
        );

        CatcherUserMessageInput.Instance.HandlerAdd(userId,
OnGradingHomework);
    }

```

```

        private async Task
ViewSampleMessageMailingCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
{
    await botClient.SendMessage(
        update.CallbackQuery.Message.Chat.Id,
        "gr-111\n" +
        "01.01.2025\n" +
        "12:00\n" +
        "Пример сообщения, которое будет отправлено для
рассылки. " +
        "В нем можно указать важную информацию, например,
такую как: " +
        "дедлайны работ, изменения в расписании и
подобное."
    );
}

        private async Task
ViewSampleMessageHomeworkCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
{
    await botClient.SendMessage(
        update.CallbackQuery.Message.Chat.Id,
        "gr-111\n" +
        "Название домашней работы\n" +
        "01.01.2025\n" +
        "01.02.2025\n" +
        "https://здесь_можно_указать_свою_ссылку_для_загрузки_работы (не
обязательно) "
    );
}

        private async Task AddMailingCallbackCommandAsync(long
userId, ITelegramBotClient botClient, Update update)
{
    var inlineKeyboard = new InlineKeyboardMarkup(
        new List<InlineKeyboardButton[]>()
        {
            new InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Посмотреть образец",
                ButtonNames.TEACHER_VIEW_SAMPLE_MESSAGE_MAILING),
            }
        }
    );

    await botClient.SendMessage(
        update.CallbackQuery.Message.Chat.Id,

```

"Для добавления сообщения на рассылку в группу введите следующие данные в указанном порядке (без порядковых цифр):\n" +

```
"1. Название группы\n" +  
"2. Дата отправки (дд.мм.гггг)\n" +  
"3. Время отправки (чч:мм)\n" +  
"4. Текст сообщения",  
replyMarkup: inlineKeyboard  
);
```

```
CatcherUserMessageInput.Instance.HandlerAdd(userId,  
OnAddMailingCatchUserMessageInput);  
}
```

```
private async Task ShowHomeworkCallbackCommandAsync(long  
userId, ITelegramBotClient botClient, Update update)  
{
```

```
    _homeworkController.Query_SelectAll();  
    var t_chat = update.CallbackQuery.Message.Chat;
```

```
    foreach (var hw in _homeworkController.Data)  
    {
```

```
        var studyGroup =  
_studyGroupsController.Query_SelectByIdAsync(hw.GroupId).Result;  
        string group_name = studyGroup != null ?  
studyGroup.Name : "не найдено";
```

```
        await botClient.SendMessage(  
            t_chat.Id,  
            $"Группа: {group_name}\n" +  
            $"Название: {hw.Name}\n" +  
            $"Дата начала: {hw.StartDate}\n" +  
            $"Дата окончания: {hw.EndDate}\n" +  
            $"Ссылка на ресурс: {hw.LinkToSource}"  
        );  
    }
```

```
    _homeworkController.Data.Clear();  
}
```

```
private async Task AddHomeworkCallbackCommandAsync(long  
userId, ITelegramBotClient botClient, Update update)  
{
```

```
    var inlineKeyboard = new InlineKeyboardMarkup(  
        new List<InlineKeyboardButton[]>()  
    {
```

```
        new InlineKeyboardButton[] {
```

```
            InlineKeyboardButton.WithCallbackData("Посмотреть образец",  
            ButtonNames.TEACHER_VIEW_SAMPLE_MESSAGE_HOMEWORK),  
        }  
    }
```

```

    });

    await botClient.SendMessage(
        update.CallbackQuery.Message.Chat.Id,
        "Для добавления домашнего задания введите
следующую информацию (без порядковых цифр):\n" +
        "1. Название группы\n" +
        "2. Название работы\n" +
        "3. Дата начала (дд.мм.гггг)\n" +
        "4. Дата окончания (дд.мм.гггг)\n" +
        "5. Ссылку для загрузки ресурсов (не
обязательно)",
        replyMarkup: inlineKeyboard
    );

    CatcherUserMessageInput.Instance.HandlerAdd(userId,
OnAddHomeworkCatchUserMessageInput);
}

private async Task
ShowAnswersToHomeworkCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
{
    _answerToHomeworkController.Query_SelectAll();
    var t_chat = update.CallbackQuery.Message.Chat;

    foreach (var hw in _answerToHomeworkController.Data)
    {
        var inlineKeyboard = new InlineKeyboardMarkup(
            new List<InlineKeyboardButton[]>()
            {
                new
                InlineKeyboardButton[] {
                    InlineKeyboardButton.WithUrl("Перейти к работе",
hw.LinkToSource),
                }
            }
        ));

        string student =
_userController.Query_SelectByIdAsync(hw.StudentId).Result.FullName;
        string name_hw =
_homeworkController.Query_SelectByIdAsync(hw.HomeworkId).Result.
Name;

        await botClient.SendMessage(
            t_chat.Id,
            $"Код: {hw.Id}\n" +
            $"Название работы: {name_hw}\n" +
            $"Выполнил: {student}\n" +

```

```

        $"Оценка: {hw.Grade}\n" +
        $"Дата добавления: {hw.DateAdded}\n" +
        $"Ссылка на работу: {hw.LinkToSource}"
    );
}

var inlineKeyboard_grading = new
InlineKeyboardMarkup(
    new List<InlineKeyboardButton[]>()
    {
        new
        InlineKeyboardButton[] {
            InlineKeyboardButton.WithCallbackData("Да",
            ButtonNames.TEACHER_YES_GRADING)
        }
    });

await botClient.SendMessage(
    t_chat.Id,
    $"Указать оценку для домашней работы?",
    replyMarkup: inlineKeyboard_grading
);

_answerToHomeworkController.Data.Clear();
}

private async Task
ShowMyDayScheduleCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
{
    var t_chat = update.CallbackQuery.Message.Chat;
    int teacherId =
GetCurrentDbUserByTelegramId(userId).Id;
    string today = DateTime.Now.ToString("dd.MM.yyyy");

    List<DbSchedule> schedule = await
GetSortScheduleByTeacherIdAndDateAsync(teacherId, today);
    List<DbSubject> subjects = await
GetSubjectsByScheduleTeacherAsync(schedule);
    List<DbStudyGroup> studyGroups = await
GetStudyGroupByScheduleTeacherAsync(schedule);

    if (schedule == null)
    {
        await EmptySchedule(botClient, t_chat);
        return;
    }
    else if (schedule.Count == 0)
    {
        await EmptySchedule(botClient, t_chat);
        return;
    }
}

```

```

    }

    if (subjects.Count != studyGroups.Count)
    {
        throw new
Exception("TeacherMainMenuCommands.ShowMyDayScheduleCallbackComm
andAsync: " +
        "количество предметов не равно количеству
учебных групп!");
    }

    for(int i = 0; i < subjects.Count; i++)
    {
        await botClient.SendMessage(
            update.CallbackQuery.Message.Chat.Id,
            $"Номер занятия: {i + 1}\n" +
            $"Предмет: {subjects[i].Name}\n" +
            $"Учебная группа: {studyGroups[i].Name}\n"
        );
    }

    await botClient.SendMessage(
        update.CallbackQuery.Message.Chat.Id,
        $"Для выбора занятия, отправьте цифру - номер
занятия."
    );

    CatcherUserMessageInput.Instance.HandlerAdd(userId,
OnChooseSubject);
}

private async Task
ShowMyWeeklyScheduleCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
{
    await botClient.SendMessage(
        update.CallbackQuery.Message.Chat.Id,
        $"Это пример #3 работы inline кнопок. Ты нажал на
кнопку: {update.CallbackQuery.Data}\n" +
        $"Напиши два целых числа через пробел!"
    );

    //
    CatcherUserMessageInput.Instance.HandlerAdd(userId,
OnButton3CatchUserInput);
}

private async Task BackCallbackCommandAsync(long userId,
ITelegramBotClient botClient, Update update)
{
    await botClient.SendMessage(
        update.CallbackQuery.Message.Chat.Id,

```

```

        $"Это пример #4 работы inline кнопок. Ты нажал на
кнопку: {update.CallbackQuery.Data}\n" +
        $"Вернемся назад, для просмотра доступных действий
вызови команду \"/menu\"";
    );

    var userSession =
_sessionManager.GetOrCreateSession(userId);
    userSession.Role = "Guest";
}
#endregion

#region CatcherUserMessageInput Callbacks
private async Task OnAddMailingCatchUserMessageInput(long
userId, ITelegramBotClient botClient, Message message)
{
    string[] lines = message.Text.Split('\n');

    DbMailing dbMailingList = new();
    dbMailingList.StudentGroupId = await
_studyGroupsController.Query_GetIdByNameAsync(lines[0]);
    dbMailingList.DispatchDate = lines[1];
    dbMailingList.DispatchTime = lines[2];
    dbMailingList.Text = lines[3];

    MailingService.Instance.AddMailing(botClient,
message.Chat, dbMailingList);

    await botClient.SendMessage(
        message.Chat.Id,
        "Сообщение успешно добавлено!"
    );
}

private async Task
OnAddHomeworkCatchUserMessageInput(long userId,
ITelegramBotClient botClient, Message message)
{
    string[] lines = message.Text.Split('\n');

    DbHomework homework = new();
    homework.GroupId = await
_studyGroupsController.Query_GetIdByNameAsync(lines[0]);
    homework.Name = lines[1];
    homework.StartDate = lines[2];
    homework.EndDate = lines[3];
    homework.LinkToSource = lines.Length == 5 ? lines[4]
: Links.LINK_TO_HOMEWORK;

    await
_homeworkController.Query_InsertAsync(homework);
}

```

```

        await botClient.SendMessage(
            message.Chat.Id,
            "Домашняя работа успешно добавлена!"
        );
    }

    private async Task OnGradingHomework(long userId,
        ITelegramBotClient botClient, Message message)
    {
        var t_chat = message.Chat;
        var inlineKeyboard = new InlineKeyboardMarkup(
            new List<InlineKeyboardButton[]>()
            {
                new
                InlineKeyboardButton[] {
                    InlineKeyboardButton.WithCallbackData("Да",
                        ButtonNames.TEACHER_YES_GRADING),
                }
            }
        ));

        string[] lines = message.Text.Split(' ');
        int answer_id = 0, answer_grade = 0;

        try
        {
            answer_id = Convert.ToInt32(lines[0]);
            answer_grade = Convert.ToInt32(lines[1]);
        }
        catch
        {
            await botClient.SendMessage(
                t_chat.Id,
                $"Нераспознанный формат ввода!" +
                $"Указать еще оценку для домашней работы?",
                replyMarkup: inlineKeyboard
            );
        }

        if(answer_id <= 0)
            return;

        DbAnswerToHomework answer = await
            _answerToHomeworkController.Query_SelectByIdAsync(answer_id);

        if (answer == null)
        {
            await botClient.SendMessage(
                t_chat.Id,
                $"Ответ ({answer_id}) не найде в базе данных!" +
                $"Указать еще оценку для домашней работы?",
                replyMarkup: inlineKeyboard
            );
        }
    }

```

```

        );
        return;
    }

    answer.Grade = answer_grade;
    await
    _answerToHomeworkController.Query_UpdateAsync(answer);

    await botClient.SendMessage(
        t_chat.Id,
        $"Указать еще оценку для домашней работы?",
        replyMarkup: inlineKeyboard
    );
}

private async Task OnChooseSubject(long userId,
ITelegramBotClient botClient, Message message)
{
    int index = 0;
    int teacher_id =
GetCurrentDbUserByTelegramId(userId).Id;
    string today = DateTime.Now.ToString("dd.MM.yyyy");

    var schedule = await
GetSortScheduleByTeacherIdAndDateAsync(teacher_id, today);

    try
    {
        index = Convert.ToInt32(message.Text);
    }
    catch
    {
        await botClient.SendMessage(
            message.Chat.Id,
            "Некорректный ввод"
        );

        return;
    }

    string subject_name = "не найден";
    string group_name = "не найдена";
    DbSchedule selected_schedule = null;

    try
    {
        selected_schedule = schedule[index];
        subject_name =
        _subjectController.Query_SelectByIdAsync(selected_schedule.Subje
ctId).Result.Name;
    }

```

```

        group_name =
        _studyGroupsController.Query_SelectByIdAsync(selected_schedule.StudyGroupId).Result.Name;
    }
    catch
    {
        await botClient.SendMessage(
            message.Chat.Id,
            "Некорректный номер предмета в расписание"
        );

        return;
    }

    _selectedSchedule.TryRemove(userId, out var _);
    _selectedSchedule.TryAdd(userId, selected_schedule);

    var inlineKeyboard = new InlineKeyboardMarkup(
        new List<InlineKeyboardButton[]>()
        {
            new InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Да",
                    ButtonNames.TEACHER_SHOW_STUDENTS_ON_SELECTED_SCHEDULE),
            }
        }
    );

    await botClient.SendMessage(
        message.Chat.Id,
        $"Выбрано занятие №{index}: {subject_name}\n"
        +
        $"Учебная группа: {group_name}\n" +
        $"Чтобы указать присутствующих студентов и выставить оценки, необходим список студентов." +
        $"Вывести список учеников?",
        replyMarkup: inlineKeyboard
    );
}

private async Task OnPresencing(long userId,
    ITelegramBotClient botClient, Message message)
{
    var t_chat = message.Chat;
    string[] lines = message.Text.Split('\n');
    var schedule = _selectedSchedule[userId];
    int studentId;

    if (schedule == null)
    {
        await botClient.SendMessage(
            t_chat.Id,
            $"Выбранный предмет не найден!"
        );
    }
}

```

```

        );

        throw new
        Exception($"TeacherMainMenuCommands.OnPresencing: расписание
        преподавателя не найдено!");
    }

    foreach (string line in lines) {
        try
        {
            studentId = Convert.ToInt32(line);
        }
        catch
        {
            await botClient.SendMessage(
                t_chat.Id,
                $"Некорректный ввод значения:\n" +
                $"{line}"
            );

            continue;
        }

        if
        (_userController.Query_SelectByIdAsync(studentId).Result == null)
        {
            await botClient.SendMessage(
                t_chat.Id,
                $"Студент с кодом - {studentId} не
найден!"
            );

            continue;
        }

        await _presenceController.Query_InsertAsync(new
        DbPresence(0, studentId, schedule.Id));
    }

    var inlineKeyboard = new InlineKeyboardMarkup(
        new List<InlineKeyboardButton[]>()
        {
            new
            InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Добавить присутствующих",
                ButtonNames.TEACHER_ADD_PRESENCE),
            },
            new
            InlineKeyboardButton[]
            {

```

```

InlineKeyboardButton.WithCallbackData("Оценить работу студентов",
ButtonNames.TEACHER_ADD_GRADE),

        }

    });

    await botClient.SendMessage(
        t_chat.Id,
        $"Присутствующие студенты сохранены!\n" +
        $"Вы можете выбрать следующие действия: ",
        replyMarkup: inlineKeyboard
    );
}

private async Task OnGrading(long userId,
ITelegramBotClient botClient, Message message)
{
    var t_chat = message.Chat;
    string[] lines = message.Text.Split('\n');
    var schedule = _selectedSchedule[userId];
    int studentId, value;

    if (schedule == null)
    {
        await botClient.SendMessage(
            t_chat.Id,
            $"Выбранный предмет не найден!"
        );

        throw new
Exception($"TeacherMainMenuCommands.OnGrading: расписание
преподавателя не найдено!");
    }

    foreach (string line in lines)
    {
        try
        {
            studentId = Convert.ToInt32(line.Split('
')[0]);

            value = Convert.ToInt32(line.Split(' ')[1]);
        }
        catch
        {
            await botClient.SendMessage(
                t_chat.Id,
                $"Некорректный ввод значения:\n" +
                $"{line.Split(' ')[0]} или
{line.Split(' ')[1]}"

            );

            continue;
        }
    }
}

```

```

    }

    if
        (_userController.Query_SelectByIdAsync(studentId).Result == null)
        {
            await botClient.SendMessage(
                t_chat.Id,
                $"Студент с кодом - {studentId} не
найден!"
            );

            continue;
        }

        await _gradeController.Query_InsertAsync(new
        DbGrade(0, studentId, schedule.Id, value));
    }

    var inlineKeyboard = new InlineKeyboardMarkup(
        new List<InlineKeyboardButton[]>()
        {
            new
            InlineKeyboardButton[] {
                InlineKeyboardButton.WithCallbackData("Добавить присутствующих",
                ButtonNames.TEACHER_ADD_PRESENCE),
            },
            new
            InlineKeyboardButton[]
            {
                InlineKeyboardButton.WithCallbackData("Оценить работу студентов",
                ButtonNames.TEACHER_ADD_GRADE),
            }
        }
    ));

    await botClient.SendMessage(
        t_chat.Id,
        $"Оценки сохранены!\n" +
        $"Вы можете выбрать следующие действия: ",
        replyMarkup: inlineKeyboard
    );
}
#endregion

#region === Methods ===
private async Task<List<DbSchedule>>
GetSortScheduleByTeacherIdAndDateAsync(int teacher_id, string
event_date)
{

```

```

        List<DbSchedule> schedule = await
        _scheduleController.Query_TrySelectByTeacherIdAndDateAsync(teach
        er_id, event_date);
        schedule = schedule.OrderBy(x =>
        x.LessonNumber).ToList();

        return schedule;
    }

    private async Task<List<DbSubject>>
    GetSubjetsByScheduleTeacherAsync(List<DbSchedule> schedule)
    {
        List<DbSubject> subjects = new();
        foreach (var item in schedule)
        {
            subjects.Add(await
            _subjectController.Query_SelectByIdAsync(item.SubjectId));
        }

        return subjects;
    }

    private async Task<List<DbStudyGroup>>
    GetStudyGroupByScheduleTeacherAsync(List<DbSchedule> schedule)
    {
        List<DbStudyGroup> studyGroups = new();
        foreach (var item in schedule)
        {
            studyGroups.Add(await
            _studyGroupsController.Query_SelectByIdAsync(item.StudyGroupId)
            );
        }

        return studyGroups;
    }

    private async Task<List<DbUser>>
    GetStudentsByScheduleTeacherAsync(long userId) // userId =
    teacherId
    {
        List<DbUser> students = null;

        var schedule = _selectedSchedule[userId];

        if (schedule == null)
            return null;

        var study_group = await
        _studyGroupsController.Query_SelectByIdAsync(schedule.StudyGroup
        Id);

        if (study_group == null)

```

```

        return null;

        var student_studyGroup = await
_student_studyGroupController.Query_SelectByStudyGroupIdAsync(st
udy_group.Id);

        if(student_studyGroup == null)
            return null;

        students = new List<DbUser>();
        foreach(var s_s in student_studyGroup)
        {
            students.Add(await
_userController.Query_SelectByIdAsync(s_s.StudentId));
        }

        return students;
    }

    private async Task EmptySchedule(ITelegramBotClient
botClient, Chat t_chat)
    {
        await botClient.SendMessage(
            t_chat.Id,
            $"На сегодня у вас отсутствуют занятия!"
        );
    }
    #endregion
}
}

```

Приложение Б.13

Файл ButtonNames.cs

```

namespace TelegramChatBot.Global
{
    public static class ButtonNames
    {
        public static string START_NAME_BUTTON = "button_";
        public static string EXIT = START_NAME_BUTTON + "exit";
        public static string BACK = START_NAME_BUTTON + "back";
        public static string ACCOUNT_INFO = START_NAME_BUTTON +
"account_info";
        public static string SHOW_PASSWORD = START_NAME_BUTTON +
"show_password";
        public static string STUDENT_LOGIN = START_NAME_BUTTON +
"student_login";
        public static string TEACHER_LOGIN = START_NAME_BUTTON +
"teacher_login";
        public static string LOGIN = START_NAME_BUTTON + "login";
    }
}

```

Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ

Лист

133

```

        public static string REGISTRATION = START_NAME_BUTTON +
"registration";

        // === Button Names For StudentMainMenuCommands === //
        public static string STUDENT_BACK = START_NAME_BUTTON +
"student_back";
        public static string
STUDENT_SHOW_MENU_FOR_ADDING_HOWEWORK = START_NAME_BUTTON +
"show_menu_for_adding_homework_student";
        public static string STUDENT_SHOW_DAY_SCHEDULE =
START_NAME_BUTTON + "show_day_schedule_student";
        public static string STUDENT_SHOW_MY_GRADES_BY_DATE =
START_NAME_BUTTON + "show_my_grades_by_date_student";
        public static string STUDENT_SHOW_MY_GRADES_BY_SUBJECT =
START_NAME_BUTTON + "show_my_grades_by_subject_student";
        public static string STUDENT_SHOW_SUBJECTS =
START_NAME_BUTTON + "show_subjects_student";
        public static string STUDENT_CHOOSE_SUBJECT =
START_NAME_BUTTON + "choose_subject_student";
        public static string STUDENT_SHOW_ALL_HOMeworks =
START_NAME_BUTTON + "show_all_homeworks_student";
        public static string STUDENT_SHOW_HOMeworks_BY_DATE =
START_NAME_BUTTON + "show_homeworks_by_date_student";
        public static string STUDENT_ADD_HOWEWORK =
START_NAME_BUTTON + "show_add_homework_student";

        // === Button Names For TeacherMainMenuCommands === //
        public static string TEACHER_SHOW_DAY_SCHEDULE =
START_NAME_BUTTON + "show_day_schedule_teacher";
        public static string TEACHER_SHOW_WEEKLY_SCHEDULE =
START_NAME_BUTTON + "show_weekly_schedule_teacher";
        public static string TEACHER_CHOOSE_SUBJECT =
START_NAME_BUTTON + "choose_subject_teacher";
        public static string TEACHER_BACK = START_NAME_BUTTON +
"teacher_back";
        public static string TEACHER_SHOW_HOWEWORK =
START_NAME_BUTTON + "show_homework_teacher";
        public static string TEACHER_SHOW_ANSWERS_TO_HOWEWORK =
START_NAME_BUTTON + "show_answers_to_homework_teacher";
        public static string
TEACHER_SHOW_STUDENTS_ON_SELECTED_SCHUDLE = START_NAME_BUTTON +
"show_students_on_selected_schudle_teacher";
        public static string TEACHER_ADD_HOWEWORK =
START_NAME_BUTTON + "add_homework_teacher";
        public static string TEACHER_ADD_MAILING =
START_NAME_BUTTON + "add_mailing_teacher";
        public static string TEACHER_VIEW_SAMPLE_MESSAGE_MAILING
= START_NAME_BUTTON + "view_sample_message_mailing_teacher";
        public static string TEACHER_VIEW_SAMPLE_MESSAGE_HOMework
= START_NAME_BUTTON + "view_sample_message_homework_teacher";
        public static string TEACHER_GRADE_1 = START_NAME_BUTTON
+ "grade_1_teacher";

```

```

        public static string TEACHER_GRADE_2 = START_NAME_BUTTON
+ "grade_2_teacher";
        public static string TEACHER_GRADE_3 = START_NAME_BUTTON
+ "grade_3_teacher";
        public static string TEACHER_GRADE_4 = START_NAME_BUTTON
+ "grade_4_teacher";
        public static string TEACHER_GRADE_5 = START_NAME_BUTTON
+ "grade_5_teacher";
        public static string TEACHER_YES_GRADING =
START_NAME_BUTTON + "grading_yes_teacher";
        public static string TEACHER_ADD_PRESENCE =
START_NAME_BUTTON + "add_presence_teacher";
        public static string TEACHER_ADD_GRADE = START_NAME_BUTTON
+ "add_grade_teacher";
    }
}

```

Приложение Б.14

Файл CatcherUserMessageInput.cs

```

using System.Collections.Concurrent;
using Telegram.Bot;
using Telegram.Bot.Types;
using Utils;

namespace TelegramChatBot.Global
{
    public class CatcherUserMessageInput
    {
        private static CatcherUserMessageInput INSTANCE;
        private static readonly object PADLOCK = new object();
        private ConcurrentDictionary<long, Func<long,
ITelegramBotClient, Message, Task>> _userHandlerNames;

        public bool IsCatch;

        public static CatcherUserMessageInput Instance
        {
            get
            {
                lock (PADLOCK)
                {
                    if (INSTANCE == null)
                        INSTANCE = new();
                }

                return INSTANCE;
            }
        }
    }
}

```

```

        private CatcherUserMessageInput() {
            _userHandlerNames = new ConcurrentDictionary<long,
Func<long, ITelegramBotClient, Message, Task>>();
        }

        private void HandlerReset(long userId)
        {
            _userHandlerNames.TryRemove(userId, out var handler);

            if (_userHandlerNames.Count == 0)
                IsCatch = false;
        }

        public async Task<bool> TryInvokeHandler(long userId,
ITelegramBotClient botClient, Message message)
        {
            if (_userHandlerNames.TryGetValue(userId, out var
handler))
            {
                await handler.Invoke(userId, botClient, message);
                HandlerReset(userId);
            }

            return handler != null;
        }

        public void HandlerAdd(long userId, Func<long,
ITelegramBotClient, Message, Task> handler)
        {
            _userHandlerNames.TryAdd (userId, handler);
            IsCatch = true;
        }
    }
}

```

Приложение Б.15

Файл CommandNames.cs

```

namespace TelegramChatBot.Global
{
    public static class CommandNames
    {
        // === Command Names For Common === //
        public static string MENU = "/menu";
        public static string START = "/start";
        public static string HELP = "/help";
        public static string EXIT = "/exit";
        public static string FAQ = "/faq";
        public static string EDUCATION_MATERIAL =
"/educational_material";
    }
}

```

```

    }
}

```

Приложение Б.16

Файл CurrentStateNames.cs

```

namespace TelegramChatBot.Global
{
    public static class CurrentStateNames
    {
        public static string START = "MainMenu";
        public static string DEFAULT = "MainMenu";
        public static string MAIN_MENU = "MainMenu";
    }
}

```

Приложение Б.17

Файл Links.cs

```

namespace Teaching_Assistant.Global
{
    public static class Links
    {
        public static string LINK_TO_HOMEWORK =
        "https://drive.google.com";
    }
}

```

Приложение Б.18

Файл RoleNames.cs

```

namespace TelegramChatBot.Global
{
    public static class RoleNames
    {
        public static string GUEST = "Guest";
        public static string ADMIN = "Admin";
        public static string TEACHER = "Teacher";
        public static string STUDENT = "Student";
    }
}

```

Приложение Б.19

Файл UserSession.cs

```

using TelegramChatBot.Global;

namespace TelegramChatBot.Models
{
    /// <summary>

```

```

/// Класс для управления состоянием пользователя
/// </summary>
public class UserSession
{
    public long TelegramId { get; }
    public string Role { get; set; } = RoleNames.GUEST;
    // по умолчанию неавторизованный
    public string CurrentState { get; set; } =
    CurrentStateNames.DEFAULT; // меню состояния

    public UserSession(long telegramId)
    {
        TelegramId = telegramId;
    }
}

```

Приложение Б.20

Файл MailingService.cs

```

using MyORM.Global;
using System.Globalization;
using Teaching_Assistant.MyORM.Controllers;
using Teaching_Assistant.MyORM.Models;
using Utils;
using System.Collections.Concurrent;
using Telegram.Bot.Types;
using TelegramChatBot.Managers;
using Telegram.Bot;

namespace Teaching_Assistant.Service
{
    public class MailingService
    {
        private static MailingService INSTANCE;
        private static readonly object PADLOCK = new object();

        private DbMailingListController _mailingListController;
        private DbStudent_StudyGroupController
        _student_studyGroupController;

        private readonly ConcurrentDictionary<Guid, Timer>
        _timers = new();
        private readonly UserSessionManager _sessionManager;

        public static MailingService Instance
        {
            get
            {
                lock (PADLOCK)
                {

```

```

        if (INSTANCE == null)
            INSTANCE = new();
    }

    return INSTANCE;
}

private MailingService()
{
    _mailingListController = new(
        AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
    );

    _student_studyGroupController = new(
        AppSettings.Instance.CurrentDbProviderName,
AppSettings.Instance.GetCurrentConnectionString()
    );

    _sessionManager = new();
}

public void AddMailing(ITelegramBotClient botClient, Chat
chat, DbMailing dbMailing)
{
    _mailingListController.Query_InsertAsync(dbMailing).Wait();

    string dateTimeString = $"{dbMailing.DispatchDate}
{dbMailing.DispatchTime}";

    DateTime sendDateTime =
DateTime.ParseExact(dateTimeString, "dd.MM.yyyy HH:mm",
CultureInfo.InvariantCulture);
    TimeSpan delay = sendDateTime - DateTime.Now;

    if (delay <= TimeSpan.Zero)
    {
        Logger.Instance.Log($"Время для отправки
сообщения уже прошло! Сообщение не отправлено.", LogLevel.Error);
        return;
    }

    var id = Guid.NewGuid();
    var timer = new Timer(async _ =>
    {
        await SendMailAsync(botClient, chat, dbMailing);
        _timers.TryRemove(id, out var _); // удаляем
таймер после выполнения
    });
}

```

```

        }, null, delay, Timeout.InfiniteTimeSpan);

        _timers[id] = timer;
        Logger.Instance.Log($"Запланирована отправка
сообщения '{dbMailing.Text}' для {dbMailing.StudentGroupId} в
{sendDateTime}");
    }

    private async Task SendMailAsync(ITelegramBotClient
botClient, Chat chat, DbMailing dbMailing)
    {
        Logger.Instance.Log($"Отправка запланированного
сообщения '{dbMailing.Text}' для {dbMailing.StudentGroupId}");
        await botClient.SendMessage(
            chat.Id,
            dbMailing.Text
        );
    }
}

```

Приложение Б.21

Файл ConsoleCommands.cs

```

namespace Utils
{
    public static class ConsoleCommands
    {
        public static Dictionary<int, string> COMMANDS = new
Dictionary<int, string>()
        {
            [-1] = "stop",
        };

        public static bool CheckCommand(int key, string input)
        {
            return COMMANDS[key] == input;
        }
    }
}

```

Приложение Б.22

Файл ConsoleHelper.cs

```

namespace Utils
{
    public static class ConsoleHelper
    {
        public static void EnterForContinued()
        {

```

```

        Console.WriteLine("Нажмите ВВОД для продолжения
        ...");
        Console.ReadLine();
    }

    public static bool IsExit() {
        string command = ConsoleCommands.COMMANDS[-1];

        Console.WriteLine($"Для выхода введите: {command}");
        string input = Console.ReadLine();

        return ConsoleCommands.CheckCommand(-1, input);
    }
}

```

Приложение Б.23

Файл Logger.cs

```

namespace Utils
{
    /// <summary>
    /// Простой логгер потокобезопасный одиначка, для логгирования
    в текстовый файл и консольный вывод
    /// </summary>
    public class Logger
    {
        private static Logger INSTANCE;
        private static readonly object PADLOCK = new object();

        private string _logFilePath = "log.txt";

        public static Logger Instance
        {
            get
            {
                lock (PADLOCK)
                {
                    if (INSTANCE == null)
                    {
                        INSTANCE = new Logger();
                    }
                }

                return INSTANCE;
            }
        }

        public string LogFilePath { get => _logFilePath; set =>
        _logFilePath = value; }

        private Logger()
    }
}

```

```

    {
    }

    public void Log(string message, string logFilePath,
LogLevel level = LogLevel.Info)
    {
        _logFilePath = logFilePath;
        Log(message, level);
    }

    public void Log(string message, LogLevel level =
LogLevel.Info)
    {
        string logMessage = $"{DateTime.Now:yyyy-MM-dd
HH:mm:ss} [{level}]: {message}";

        Console.WriteLine(logMessage);

        try
        {
            File.AppendAllText(_logFilePath, logMessage +
Environment.NewLine);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Ошибка записи в лог-файл:
{ex.Message}");
        }
    }

    public enum LogLevel
    {
        Info,
        Warning,
        Error
    }
}

```

Приложение Б.23

Файл AppSettings.cs

```

using System.Configuration;
using System.Data.Common;

namespace MyORM.Global
{
    /// <summary>
    /// Одиночка. Настройки приложения.
    /// </summary>

```

```

public class AppSettings
{
    private static AppSettings INSTANCE;
    private static readonly object PADLOCK = new object();

    public string CurrentDbProviderName =
"Microsoft.Data.SqlClient";
    public string CurrentDbConnectionName = "DbTypeLocalSQL";

    private AppSettings()
    {
        Startup();
    }

    public static AppSettings Instance
    {
        get
        {
            lock (PADLOCK)
            {
                if (INSTANCE == null)
                {
                    INSTANCE = new AppSettings();
                }

                return INSTANCE;
            }
        }

        /// <summary>
        ///     Получить строку подключения из
        ConfigurationManager.ConnectionStrings по имени провайдера
        /// </summary>
        /// <param name="providerName"></param>
        /// <returns></returns>
        public string GetConnectionStringByProviderName(string
providerName)
        {
            string found_value = null;

            ConnectionStringSettingsCollection settings =
ConfigurationManager.ConnectionStrings;

            if (settings != null)
            {
                foreach (ConnectionStringSettings cs in settings)
                {
                    if (cs.ProviderName == providerName)
                    {
                        found_value = cs.ConnectionString;
                        break;
                    }
                }
            }
        }
    }
}

```

```

        }
    }

    return found_value;
}

/// <summary>
///     Получить строку подключения из
ConfigurationManager.ConnectionStrings по названию
/// </summary>
/// <param name="name"></param>
/// <returns></returns>
public string GetConnectionStringByName(string name)
{
    string found_value = null;

    ConnectionStringSettingsCollection settings =
ConfigurationManager.ConnectionStrings;

    if (settings != null)
    {
        foreach (ConnectionStringSettings cs in settings)
        {
            if (cs.Name == name)
            {
                found_value = cs.ConnectionString;
                break;
            }
        }
    }

    return found_value;
}

/// <summary>
///     Возвращает строку подключения по имени поля
CurrentDbConnectionName из AppConfig
/// </summary>
/// <returns></returns>
public string GetCurrentConnectionString()
{
    return
GetConnectionStringByName(CurrentDbConnectionName);
}

/// <summary>
///     Вызывается в конструкторе. При необходимости можно
вызвать из вне.
///     Установка начальных настроек приложения:
///     - регистрация фабрик провайдеров;
///     - добавить при необходимости ...

```

```

    /// </summary>
    public void Startup()
    {
        //
        DbProviderFactories.RegisterFactory("System.Data.OleDb",
        System.Data.OleDb.OleDbFactory.Instance);

        DbProviderFactories.RegisterFactory("Microsoft.Data.SqlClient",
        Microsoft.Data.SqlClient.SqlClientFactory.Instance);
    }
}

```

Приложение Б.24

Файл DbAnswerToHomework.cs

```

using MyORM.Models;
using System.Data.Common;
using System.Text.RegularExpressions;

namespace Teaching_Assistant.MyORM.Models
{
    public class DbAnswerToHomework : IBaseModel
    {
        private int _id;
        public int Id { get => _id; set => _id = value; }
        public int HomeworkId { get; set; }
        public int StudentId { get; set; }
        public int Grade { get; set; }
        public string DateAdded { get; set; }
        public string LinkToSource { get; set; }

        public string ToParamsInsertQueryWithID => $"([Id],
        [HomeworkId], [StudentId], [Grade], [DateAdded], [LinkToSource])
        VALUES " +
            $"(@id, @param0, @param1, @param2, @param3, @param3)";

        public string ToParamsInsertQueryWithoutID =>
        $"([HomeworkId], [StudentId], [Grade], [DateAdded],
        [LinkToSource]) VALUES " +
            $"(@param0, @param1, @param2, @param3, @param4)";

        public string ToParamsUpdateQuery => $"[HomeworkId] =
        @param0, [StudentId] = @param1, " +
            $"[Grade] = @param2, [LinkToSource] = @param3";

        public string[] ToArrayStr => new string[] {
        HomeworkId.ToString(), StudentId.ToString(), Grade.ToString(),
        DateAdded, LinkToSource };
    }
}

```

```

        public DbAnswerToHomework() { }
        public DbAnswerToHomework(int id, int HomeworkId, int
StudentId, int Grade, string DateAdded, string LinkToSource)
        {
            _id = id;
            this.HomeworkId = HomeworkId;
            this.StudentId = StudentId;
            this.Grade = Grade;
            this.DateAdded = DateAdded;
            this.LinkToSource = LinkToSource;
        }
        public DbAnswerToHomework(ref DbDataReader reader)
        {
            SetData(ref reader);
        }

        public void SetData(ref DbDataReader reader)
        {
            _id = Convert.ToInt32(reader["Id"]);

            if (reader["HomeworkId"] != DBNull.Value)
                HomeworkId =
Convert.ToInt32(reader["HomeworkId"]);

            if (reader["StudentId"] != DBNull.Value)
                StudentId = Convert.ToInt32(reader["StudentId"]);

            if (reader["Grade"] != DBNull.Value)
                Grade = Convert.ToInt32(reader["Grade"]);

            if (reader["DateAdded"] != DBNull.Value)
                DateAdded =
Convert.ToString(reader["DateAdded"]);

            if (reader["LinkToSource"] != DBNull.Value)
                LinkToSource =
Convert.ToString(reader["LinkToSource"]);
        }
    }
}

```

Приложение Б.25

Файл DbEducationalMaterial.cs

```

using MyORM.Models;
using System.Data.Common;

namespace Teaching_Assistant.MyORM.Models
{
    public class DbEducationalMaterial : IBaseModel
    {

```

```

private int _id;
public int Id { get => _id; set => _id = value; }
public string Name { get; set; }
public string Description { get; set; }
public string LinkToSource { get; set; }

public string ToParamsInsertQueryWithID => $"([Id],
[Name], [Description], [LinkToSource]) VALUES " +
    $"(@id, @param0, @param1, @param2)";

public string ToParamsInsertQueryWithoutID => $"([Name],
[Description], [LinkToSource]) VALUES " +
    $"(@param0, @param1, @param2)";

public string ToParamsUpdateQuery => $"[Name] = @param0,
[Description] = @param1, [LinkToSource] = @param2";

public string[] ToArrayStr => new string[] { Name,
Description, LinkToSource };

public DbEducationalMaterial() { }
public DbEducationalMaterial(int id, string Name, string
Description, string LinkToSource)
{
    _id = id;
    this.Name = Name;
    this.Description = Description;
    this.LinkToSource = LinkToSource;
}
public DbEducationalMaterial(ref DbDataReader reader)
{
    SetData(ref reader);
}

public void SetData(ref DbDataReader reader)
{
    _id = Convert.ToInt32(reader["Id"]);

    if (reader["Name"] != DBNull.Value)
        Name = Convert.ToString(reader["Name"]);

    if (reader["Description"] != DBNull.Value)
        Description = Convert.ToString(reader["Description"]);

    if (reader["LinkToSource"] != DBNull.Value)
        LinkToSource = Convert.ToString(reader["LinkToSource"]);
}
}

```

Приложение Б.26

Файл DbFAQ.cs

```
using MyORM.Models;
using System.Data.Common;

namespace Teaching_Assistant.MyORM.Models
{
    public class DbFAQ : IBaseModel
    {
        private int _id;
        public int Id { get => _id; set => _id = value; }
        public string Name { get; set; }
        public string Description { get; set; }

        public string ToParamsInsertQueryWithID => $"([Id],
[Name], [Description]) " +
            $"VALUES " +
            $"(@id, @param0, @param1)";

        public string ToParamsInsertQueryWithoutID => $"([Name],
[Description]) " +
            $"VALUES " +
            $"(@param0, @param1)";

        public string ToParamsUpdateQuery => $"[Name] = @param0,
[Description] = @param1";

        public string[] ToArrayStr => new string[] { Name,
Description };

        public DbFAQ() { }
        public DbFAQ(int id, string Name, string Description)
        {
            _id = id;
            this.Name = Name;
            this.Description = Description;
        }
        public DbFAQ(ref DbDataReader reader)
        {
            SetData(ref reader);
        }

        public void SetData(ref DbDataReader reader)
        {
            _id = Convert.ToInt32(reader["Id"]);

            if (reader["Name"] != DBNull.Value)
                Name = Convert.ToString(reader["Name"]);
        }
    }
}
```

Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ

Лист

148

```

        if (reader["Description"] != DBNull.Value)
            Description
        Convert.ToString(reader["Description"]);
    }
}

```

Приложение Б.27

Файл DbGrade.cs

```

using MyORM.Models;
using System.Data.Common;

namespace Teaching_Assistant.MyORM.Models
{
    public class DbGrade : IBaseModel
    {
        private int _id;
        public int Id { get => _id; set => _id = value; }
        public int StudentId { get; set; }
        public int ScheduleId { get; set; }
        public int Value { get; set; }

        public string ToParamsInsertQueryWithID => $"([Id],
[StudentId], [ScheduleId], [Value]) VALUES " +
        $"(@id, @param0, @param1, @param2)";

        public string ToParamsInsertQueryWithoutID =>
        $"([StudentId], [ScheduleId], [Value]) VALUES " +
        $"(@param0, @param1, @param2)";

        public string ToParamsUpdateQuery => $"[StudentId] =
@param0, [ScheduleId] = @param1, [Value] = @param2";

        public string[] ToArrayStr => new string[] {
StudentId.ToString(), ScheduleId.ToString(), Value.ToString() };

        public DbGrade() { }
        public DbGrade(int id, int StudentId, int ScheduleId, int
Value)
        {
            _id = id;
            this.StudentId = StudentId;
            this.ScheduleId = ScheduleId;
            this.Value = Value;
        }
        public DbGrade(ref DbDataReader reader)
        {
            SetData(ref reader);
        }
    }
}

```

```

public void SetData(ref DbDataReader reader)
{
    _id = Convert.ToInt32(reader["Id"]);

    if (reader["StudentId"] != DBNull.Value)
        StudentId = Convert.ToInt32(reader["StudentId"]);

    if (reader["ScheduleId"] != DBNull.Value)
        ScheduleId =
Convert.ToInt32(reader["ScheduleId"]);

    if (reader["Value"] != DBNull.Value)
        Value = Convert.ToInt32(reader["Value"]);
}
}
}

```

Приложение Б.28

Файл DbHomework.cs

```

using MyORM.Models;
using System.Data.Common;

namespace Teaching_Assistant.MyORM.Models
{
    public class DbHomework : IBaseModel
    {
        private int _id;
        public int Id { get => _id; set => _id = value; }
        public int GroupId { get; set; }
        public string Name { get; set; }
        public string StartDate { get; set; }
        public string EndDate { get; set; }
        public string LinkToSource { get; set; }

        public string ToParamsInsertQueryWithID => $"([Id],
[GroupId], [Name], [StartDate], [EndDate], [LinkToSource]) VALUES
" +
        $"(@id, @param0, @param1, @param2, @param3, @param4)";

        public string ToParamsInsertQueryWithoutID =>
$"([GroupId], [Name], [StartDate], [EndDate], [LinkToSource])
VALUES " +
        $"(@param0, @param1, @param2, @param3, @param4)";

        public string ToParamsUpdateQuery => $"[GroupId] =
@param0, [Name] = @param1, " +
        $"[StartDate] = @param2, [EndDate] = @param3,
[LinkToSource] = @param4";
    }
}

```

```

        public string[] ToArrayStr => new string[] {
            GroupId.ToString(), Name, StartDate, EndDate, LinkToSource };

        public DbHomework() { }
        public DbHomework(int id, int GroupId, string Name, string
StartDate, string EndDate, string LinkToSource)
        {
            _id = id;
            this.GroupId = GroupId;
            this.Name = Name;
            this.StartDate = StartDate;
            this.EndDate = EndDate;
            this.LinkToSource = LinkToSource;
        }
        public DbHomework(ref DbDataReader reader)
        {
            SetData(ref reader);
        }

        public void SetData(ref DbDataReader reader)
        {
            _id = Convert.ToInt32(reader["Id"]);

            if (reader["GroupId"] != DBNull.Value)
                GroupId = Convert.ToInt32(reader["GroupId"]);

            if (reader["Name"] != DBNull.Value)
                Name = Convert.ToString(reader["Name"]);

            if (reader["StartDate"] != DBNull.Value)
                StartDate =
Convert.ToString(reader["StartDate"]);

            if (reader["EndDate"] != DBNull.Value)
                EndDate = Convert.ToString(reader["EndDate"]);

            if (reader["LinkToSource"] != DBNull.Value)
                LinkToSource =
Convert.ToString(reader["LinkToSource"]);
        }
    }
}

```

Приложение Б.29

Файл DbMailing.cs

```

using MyORM.Models;
using System.Data.Common;

namespace Teaching_Assistant.MyORM.Models
{

```

```

public class DbMailing : IBaseModel
{
    private int _id;
    public int Id { get => _id; set => _id = value; }
    public string Text { get; set; }
    public int StudentGroupId { get; set; }
    public string DispatchDate { get; set; }
    public string DispatchTime { get; set; }

    public string ToParamsInsertQueryWithID => $"([Id],
[Text], [StudentGroupId], [DispatchDate], [DispatchTime]) " +
        $"VALUES " +
        $"(@id, @param0, @param1, @param2, @param3)";

    public string ToParamsInsertQueryWithoutID => $"([Text],
[StudentGroupId], [DispatchDate], [DispatchTime]) " +
        $"VALUES " +
        $"(@param0, @param1, @param2, @param3)";

    public string ToParamsUpdateQuery => $"[Text] = @param0,
[StudentGroupId] = @param1, [DispatchDate] = @param2,
[DispatchTime] = @param3";

    public string[] ToArrayStr => new string[] { Text,
StudentGroupId.ToString(), DispatchDate, DispatchTime };

    public DbMailing() { }
    public DbMailing(int id, string Text, int StudentGroupId,
string DispatchDate, string DispatchTime)
    {
        _id = id;
        this.Text = Text;
        this.StudentGroupId = StudentGroupId;
        this.DispatchDate = DispatchDate;
        this.DispatchTime = DispatchTime;
    }
    public DbMailing(ref DbDataReader reader)
    {
        SetData(ref reader);
    }

    public void SetData(ref DbDataReader reader)
    {
        _id = Convert.ToInt32(reader["Id"]);

        if (reader["Text"] != DBNull.Value)
            Text = Convert.ToString(reader["Text"]);

        if (reader["StudentGroupId"] != DBNull.Value)
            StudentGroupId =
Convert.ToInt32(reader["StudentGroupId"]);

```

```

        if (reader["DispatchDate"] != DBNull.Value)
            DispatchDate
Convert.ToString(reader["DispatchDate"]);

        if (reader["DispatchTime"] != DBNull.Value)
            DispatchTime
Convert.ToString(reader["DispatchTime"]);
    }
}

```

Приложение Б.30

Файл DbPresence.cs

```

using MyORM.Models;
using System.Data.Common;

namespace Teaching_Assistant.MyORM.Models
{
    public class DbPresence : IBaseModel
    {
        private int _id;
        public int Id { get => _id; set => _id = value; }
        public int StudentId { get; set; }
        public int ScheduleId { get; set; }

        public string ToParamsInsertQueryWithID => $"([Id],
[StudentId], [ScheduleId]) VALUES " +
        $"(@id, @param0, @param1)";

        public string ToParamsInsertQueryWithoutID =>
        $"([StudentId], [ScheduleId]) VALUES " +
        $"(@param0, @param1)";

        public string ToParamsUpdateQuery => $"[StudentId] =
@param0, [ScheduleId] = @param1";

        public string[] ToArrayStr => new string[] {
StudentId.ToString(), ScheduleId.ToString() };

        public DbPresence() { }
        public DbPresence(int id, int StudentId, int ScheduleId)
        {
            _id = id;
            this.StudentId = StudentId;
            this.ScheduleId = ScheduleId;
        }
        public DbPresence(ref DbDataReader reader)
        {
            SetData(ref reader);
        }
    }
}

```

```

public void SetData(ref DbDataReader reader)
{
    _id = Convert.ToInt32(reader["Id"]);

    if (reader["StudentId"] != DBNull.Value)
        StudentId = Convert.ToInt32(reader["StudentId"]);

    if (reader["ScheduleId"] != DBNull.Value)
        ScheduleId
        =
        Convert.ToInt32(reader["ScheduleId"]);
}
}

```

Приложение Б.31

Файл DbSchedule.cs

```

using MyORM.Models;
using System.Data.Common;

namespace Teaching_Assistant.MyORM.Models
{
    public class DbSchedule : IBaseModel
    {
        private int _id;
        public int Id { get => _id; set => _id = value; }
        public int TeacherId { get; set; }
        public int SubjectId { get; set; }
        public int StudyGroupId { get; set; }
        public int LessonNumber { get; set; }
        public string EventDate { get; set; }

        public string ToParamsInsertQueryWithID => $"([Id],
[TeacherId], [SubjectId], [StudyGroupId], [LessonNumber],
[EventDate]) VALUES " +
        $"(@id, @param0, @param1, @param2, @param3, @param4)";

        public string ToParamsInsertQueryWithoutID =>
        $"([TeacherId], [SubjectId], [StudyGroupId], [LessonNumber],
[EventDate]) VALUES " +
        $"@param0, @param1, @param2, @param3, @param4)";

        public string ToParamsUpdateQuery => $"[TeacherId] =
@param0, [SubjectId] = @param1, [StudyGroupId] = @param2, " +
        $"[LessonNumber] = @param3, [EventDate] = @param4";

        public string[] ToArrayStr => new string[] {
TeacherId.ToString(), SubjectId.ToString(),
StudyGroupId.ToString(), LessonNumber.ToString(), EventDate };
}

```

```

        public DbSchedule() { }
        public DbSchedule(int id, int TeacherId, int SubjectId,
int StudyGroupId, int LessonNumber, string EventDate)
        {
            _id = id;
            this.TeacherId = TeacherId;
            this.SubjectId = SubjectId;
            this.StudyGroupId = StudyGroupId;
            this.LessonNumber = LessonNumber;
            this.EventDate = EventDate;
        }
        public DbSchedule(ref DbDataReader reader)
        {
            SetData(ref reader);
        }

        public void SetData(ref DbDataReader reader)
        {
            _id = Convert.ToInt32(reader["Id"]);

            if (reader["TeacherId"] != DBNull.Value)
                TeacherId = Convert.ToInt32(reader["TeacherId"]);

            if (reader["SubjectId"] != DBNull.Value)
                SubjectId = Convert.ToInt32(reader["SubjectId"]);

            if (reader["StudyGroupId"] != DBNull.Value)
                StudyGroupId =
Convert.ToInt32(reader["StudyGroupId"]);

            if (reader["LessonNumber"] != DBNull.Value)
                LessonNumber =
Convert.ToInt32(reader["LessonNumber"]);

            if (reader["EventDate"] != DBNull.Value)
                EventDate =
Convert.ToString(reader["EventDate"]);
        }
    }
}

```

Приложение Б.32

Файл DbStudent_StudyGroup.cs

```

using MyORM.Models;
using System.Data.Common;

namespace Teaching_Assistant.MyORM.Models
{
    public class DbStudent_StudyGroup : IBaseModel
    {

```

```

        private int _studyGroupId;
        /// <summary>
        /// StudyGroupId
        /// </summary>
        public int Id { get => _studyGroupId; set => _studyGroupId
= value; }
        public int StudentId { get; set; }

        public string ToParamsInsertQueryWithID =>
$"([StudyGroupId], [StudentId]) VALUES " +
        $"(@id, @param0)";

        public string ToParamsInsertQueryWithoutID =>
$"([StudyGroupId], [StudentId]) VALUES " +
        $"(@param0, @param1)";

        public string ToParamsUpdateQuery => $"[StudyGroupId] =
@param0, [StudentId] = @param1";

        public string[] ToArrayStr => new string[] {
_studyGroupId.ToString(), StudentId.ToString() };

        public DbStudent_StudyGroup() { }
        public DbStudent_StudyGroup(int StudyGroupId, int
StudentId)
        {
            this.Id = StudyGroupId;
            this.StudentId = StudentId;
        }
        public DbStudent_StudyGroup(ref DbDataReader reader)
        {
            SetData(ref reader);
        }

        public void SetData(ref DbDataReader reader)
        {
            this.Id = Convert.ToInt32(reader["StudyGroupId"]);
            this.StudentId
            =
            Convert.ToInt32(reader["StudentId"]);
        }
    }
}

```

Приложение Б.33

Файл DbStudyGroup.cs

```

using MyORM.Models;
using System.Data.Common;

namespace Teaching_Assistant.MyORM.Models
{

```

```

public class DbStudyGroup : IBaseModel
{
    private int _id;
    public int Id { get => _id; set => _id = value; }
    public int TeacherId { get; set; }
    public string Name { get; set; }
    public string StartDate { get; set; }
    public string EndDate { get; set; }

    public string ToParamsInsertQueryWithID => $"([Id],
[TeacherId], [Name], [StartDate], [EndDate]) " +
        $"VALUES " +
        $"(@id, @param0, @param1, @param2, @param3)";

    public string ToParamsInsertQueryWithoutID =>
$"([TeacherId], [Name], [StartDate], [EndDate]) " +
        $"VALUES " +
        $"( @param0, @param1, @param2, @param3)";

    public string ToParamsUpdateQuery => $"[TeacherId] =
@param0, [Name] = @param1, [StartDate] = @param2, [EndDate] =
@param3";

    public string[] ToArrayStr => new string[] {
TeacherId.ToString(), Name, StartDate, EndDate };

    public DbStudyGroup() { }
    public DbStudyGroup(int id, int TeacherId, string Name,
string StartDate, string EndDate)
    {
        _id = id;
        this.TeacherId = TeacherId;
        this.Name = Name;
        this.StartDate = StartDate;
        this.EndDate = EndDate;
    }
    public DbStudyGroup(ref DbDataReader reader)
    {
        SetData(ref reader);
    }

    public void SetData(ref DbDataReader reader)
    {
        _id = Convert.ToInt32(reader["Id"]);

        if (reader["TeacherId"] != DBNull.Value)
            TeacherId = Convert.ToInt32(reader["TeacherId"]);

        if (reader["Name"] != DBNull.Value)
            Name = Convert.ToString(reader["Name"]);

        if (reader["StartDate"] != DBNull.Value)

```

```

        StartDate
Convert.ToString(reader["StartDate"]);

        if (reader["EndDate"] != DBNull.Value)
            EndDate = Convert.ToString(reader["EndDate"]);
    }
}

```

Приложение Б.34

Файл DbSubject.cs

```

using MyORM.Models;
using System.Data.Common;

namespace Teaching_Assistant.MyORM.Models
{
    public class DbSubject : IBaseModel
    {
        private int _id;
        public int Id { get => _id; set => _id = value; }
        public string Name { get; set; }
        public string Description { get; set; }
        public int TeacherId { get; set; }
        public int GradeLevel { get; set; }
        public string CreatedAt { get; set; }

        public string ToParamsInsertQueryWithID => $"([Id],
[Name], [Description], [TeacherId], [GradeLevel], [CreatedAt])
VALUES " +
            $"(@id, @param0, @param1, @param2, @param3, @param4)";

        public string ToParamsInsertQueryWithoutID => $"([Name],
[Description], [TeacherId], [GradeLevel], [CreatedAt]) VALUES " +
            $"(@id, @param0, @param1, @param2, @param3, @param4)";

        public string ToParamsUpdateQuery => $"[Name] = @param0,
[Description] = @param1, " +
            $"[TeacherId] = @param2, [GradeLevel] = @param3,
[CreatedAt] = @param4";

        public string[] ToArrayStr => new string[] { Name,
Description, TeacherId.ToString(), GradeLevel.ToString(),
CreatedAt };

        public DbSubject() { }
        public DbSubject(int id, string Name, string Description,
int TeacherId, int GradeLevel, string CreatedAt)
        {
            _id = id;
            this.Name = Name;

```

```

        this.Description = Description;
        this.TeacherId = TeacherId;
        this.GradeLevel = GradeLevel;
        this.CreatedAt = CreatedAt;
    }
    public DbSubject(ref DbDataReader reader)
    {
        SetData(ref reader);
    }

    public void SetData(ref DbDataReader reader)
    {
        _id = Convert.ToInt32(reader["Id"]);

        if (reader["Name"] != DBNull.Value)
            Name = Convert.ToString(reader["Name"]);

        if (reader["Description"] != DBNull.Value)
            Description =
            Convert.ToString(reader["Description"]);

        if (reader["TeacherId"] != DBNull.Value)
            TeacherId = Convert.ToInt32(reader["TeacherId"]);

        if (reader["GradeLevel"] != DBNull.Value)
            GradeLevel =
            Convert.ToInt32(reader["GradeLevel"]);

        if (reader["CreatedAt"] != DBNull.Value)
            CreatedAt =
            Convert.ToString(reader["CreatedAt"]);
    }
}

```

Приложение Б.35

Файл DbUser.cs

```

using System.Data.Common;

namespace MyORM.Models
{
    public class DbUser : IBaseModel
    {
        private int _id;
        public int Id { get => _id; set => _id = value; }
        public long TelegramId { get; set; }
        public string FullName { get; set; }
        public string NickName { get; set; }
        public string Password { get; set; }
        public string Role { get; set; }
    }
}

```

```

        public string ToParamsInsertQueryWithID => $"([Id],
[TelegramId], [FullName], [NickName], [Password], [Role]) VALUES
" +
        $"(@id, @param0, @param1, @param2, @param3, @param4)";
        public string ToParamsInsertQueryWithoutID =>
$"([TelegramId], [FullName], [NickName], [Password], [Role])
VALUES " +
        $"(@param0, @param1, @param2, @param3, @param4)";

        public string ToParamsUpdateQuery => $"[TelegramId] =
@param0, [FullName] = @param1, " +
        $"[NickName] = @param2, [Password] = @param3, [Role]
= @param4";

        public string[] ToArrayStr => new string[] {
TelegramId.ToString(), FullName, NickName, Password, Role };

        public DbUser() { }
        public DbUser(int id, long telegramId, string fullname,
string nickname, string login, string password, string role)
        {
            _id = id;
            TelegramId = telegramId;
            FullName = fullname;
            NickName = nickname;
            Password = password;
            Role = role;
        }
        public DbUser(ref DbDataReader reader)
        {
            SetData(ref reader);
        }

        public void SetData(ref DbDataReader reader)
        {
            _id = Convert.ToInt32(reader["Id"]);

            if (reader["TelegramId"] != DBNull.Value)
                TelegramId =
Convert.ToInt32(reader["TelegramId"]);

            if (reader["FullName"] != DBNull.Value)
                FullName = reader["FullName"].ToString();

            if (reader["NickName"] != DBNull.Value)
                NickName = reader["NickName"].ToString();

            if (reader["Password"] != DBNull.Value)
                Password = reader["Password"].ToString();

            if (reader["Role"] != DBNull.Value)

```

```

        Role = reader["Role"].ToString();
    }
}

```

Приложение Б.36

Файл DbAnswerToHomeworkController.cs

```

using MyORM.Controllers;
using Teaching_Assistant.MyORM.Models;

namespace Teaching_Assistant.MyORM.Controllers
{
    public class DbAnswerToHomeworkController :
        DbBaseController<DbAnswerToHomework>
    {
        public DbAnswerToHomeworkController(string providerName,
            string connectionString, string tableName = "") :
            base(providerName, connectionString, tableName)
        {
            _tableName = "AnswerToHomework";
        }
    }
}

```

Приложение Б.37

Файл DbEducationalMaterialController.cs

```

using MyORM.Controllers;
using Teaching_Assistant.MyORM.Models;

namespace Teaching_Assistant.MyORM.Controllers
{
    public class DbEducationalMaterialController :
        DbBaseController<DbEducationalMaterial>
    {
        public DbEducationalMaterialController(string
            providerName, string connectionString, string tableName = "") :
            base(providerName, connectionString, tableName)
        {
            _tableName = "EducationalMaterial";
        }
    }
}

```

Приложение Б.38

Файл DbFAQController.cs

```

using MyORM.Controllers;
using Teaching_Assistant.MyORM.Models;

```

```

namespace Teaching_Assistant.MyORM.Controllers
{
    public class DbFAQController : DbBaseController<DbFAQ>
    {
        public DbFAQController(string providerName, string
connectionString, string tableName = "") : base(providerName,
connectionString, tableName)
        {
            _tableName = "FAQ";
        }
    }
}

```

Приложение Б.39

Файл DbGradeController.cs

```

using Microsoft.Data.SqlClient;
using MyORM.Controllers;
using System.Collections.Generic;
using System.Data.Common;
using Teaching_Assistant.MyORM.Models;
using Telegram.Bot.Types;

namespace Teaching_Assistant.MyORM.Controllers
{
    public class DbGradeController : DbBaseController<DbGrade>
    {
        public DbGradeController(string providerName, string
connectionString, string tableName = "") : base(providerName,
connectionString, tableName)
        {
            _tableName = "Grade";
        }

        public async Task<List<DbGrade>>
Query_SelectsByStudentId(int studentId)
        {
            List<DbGrade> found = null;

            try
            {
                string query = $"select * from {_tableName} where
StudentId = @param0";
                List<DbParameter> parameters = new
List<DbParameter>();
                parameters.Add(new SqlParameter() { ParameterName
= "@param0", Value = studentId });

                found = await Query_SelectsByQueryAsync(query,
parameters);
            }
            catch { }
        }
    }
}

```

```

        }
        catch (Exception ex)
        {
            throw new
Exception($"DbGradeController.Query_SelectsByStudentId(Exception
): {ex.Message}");
        }

        return found;
    }
}
}

```

Приложение Б.40

Файл DbHomeworkController.cs

```

using Microsoft.Data.SqlClient;
using MyORM.Controllers;
using System.Data.Common;
using Teaching_Assistant.MyORM.Models;
using Utils;

namespace Teaching_Assistant.MyORM.Controllers
{
    public class DbHomeworkController :
DbBaseController<DbHomework>
    {
        public DbHomeworkController(string providerName, string
connectionString, string tableName = "") : base(providerName,
connectionString, tableName)
        {
            _tableName = "Homework";
        }

        public async Task<List<DbHomework>>
Query_SelectByGroupIdAsync(int groupId)
        {
            List<DbHomework> found = null;

            try
            {
                string query = $"select * from {_tableName} where
GroupId = @groupId";

                List<DbParameter> parameters = new
List<DbParameter>();
                parameters.Add(new SqlParameter() { ParameterName
= "@groupId", Value = groupId });

                found = await Query_SelectsByQueryAsync(query,
parameters);
            }
            catch { }
        }
    }
}

```

```

        }
        catch (Exception ex)
        {
            throw new
Exception($"DbHomeworkController.Query_SelectByGroupIdAsync(Exce
ption): {ex.Message}");
        }

        return found;
    }

    public async Task<List<DbHomework>>
Query_SelectByGroupIdAndDateAsync(int groupId, string date_from,
string date_to = "")
    {
        List<DbHomework> found = null;

        try
        {
            string query = $"select * from {_tableName} where
GroupId = @groupId and StartDate = @date_from";
            query += string.IsNullOrEmpty(date_to) ? "" : "
and EndDate = @date_to";

            List<DbParameter> parameters = new
List<DbParameter>();
            parameters.Add(new SqlParameter() { ParameterName
= "@groupId", Value = groupId });
            parameters.Add(new SqlParameter() { ParameterName
= "@date_from", Value = date_from });
            if(!string.IsNullOrEmpty(date_to))
                parameters.Add(new SqlParameter() {
ParameterName = "@date_to", Value = date_to });

            found = await Query_SelectsByQueryAsync(query,
parameters);
        }
        catch (Exception ex)
        {
            throw new
Exception($"DbHomeworkController.Query_SelectByGroupIdAndDateAsy
nc(Exception): {ex.Message}");
        }

        return found;
    }
}

```

Приложение Б.41

Файл DbMailingListController.cs

					ПГУ 1.090301.26.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		164

```

using MyORM.Controllers;
using Teaching_Assistant.MyORM.Models;

namespace Teaching_Assistant.MyORM.Controllers
{
    public class DbMailingListController :
    DbBaseController<DbMailing>
    {
        public DbMailingListController(string providerName,
        string connectionString, string tableName = "") :
        base(providerName, connectionString, tableName)
        {
            _tableName = "MailingList";
        }
    }
}

```

Приложение Б.42

Файл DbPresenceController.cs

```

using MyORM.Controllers;
using Teaching_Assistant.MyORM.Models;

namespace Teaching_Assistant.MyORM.Controllers
{
    public class DbPresenceController :
    DbBaseController<DbPresence>
    {
        public DbPresenceController(string providerName, string
        connectionString, string tableName = "") : base(providerName,
        connectionString, tableName)
        {
            _tableName = "Presence";
        }
    }
}

```

Приложение Б.43

Файл DbSchedulesController.cs

```

using Microsoft.Data.SqlClient;
using MyORM.Controllers;
using System.Data.Common;
using Teaching_Assistant.MyORM.Models;

namespace Teaching_Assistant.MyORM.Controllers
{
    public class DbSchedulesController :
    DbBaseController<DbSchedule>
    {

```

```

        public DbSchedulesController(string providerName, string
connectionString, string tableName = "") : base(providerName,
connectionString, tableName)
        {
            _tableName = "Schedule";
        }

        public async Task<List<DbSchedule>>
Query_TrySelectByTeacherIdAndDateAsync(int teacher_id, string
event_date)
        {
            List<DbSchedule> found = null;

            try
            {
                string query = $"select * from {_tableName} where
TeacherId = @param0 and EventDate = @param1";

                List<DbParameter> args = new List<DbParameter>();
                args.Add(new SqlParameter() { ParameterName =
"@param0", Value = teacher_id });
                args.Add(new SqlParameter() { ParameterName =
"@param1", Value = event_date });

                found = await Query_SelectsByQueryAsync(query,
args);
            }
            catch (Exception ex)
            {
                throw new
Exception("DbSchedulesController.Query_TrySelectByTeacherIdAndDa
te(Exception): " + ex.Message);
            }

            return found;
        }

        public async Task<List<DbSchedule>>
TrySelectByTeacherIdAndDateAsync(int teacher_id, string
event_date)
        {
            List<DbSchedule> found = null;

            try
            {
                string query = $"select * from {_tableName}";
                var buff = await
Query_SelectsByQueryAsync(query);

                foreach(var s in buff)
                {

```

```

        if (s.EventDate == event_date && s.TeacherId ==
teacher_id)
            found.Add(s);
        }
    }
    catch (Exception ex)
    {
        throw new
Exception("DbSchedulesController.TrySelectByTeacherIdAndDateAsyn
c(Exception): " + ex.Message);
    }

    return found;
}

public async Task<List<DbSchedule>>
Query_TrySelectByStudyGroupIdAndDateAsync(int study_group_id,
string event_date)
{
    List<DbSchedule> found = null;

    try
    {
        string query = $"select * from {_tableName} where
StudyGroupId = @param0 and EventDate = @param1";

        List<DbParameter> args = new List<DbParameter>();
        args.Add(new SqlParameter() { ParameterName =
"@param0", Value = study_group_id });
        args.Add(new SqlParameter() { ParameterName =
"@param1", Value = event_date });

        found = await Query_SelectsByQueryAsync(query,
args);
    }
    catch (Exception ex)
    {
        throw new
Exception("DbSchedulesController.Query_TrySelectByStudyGroupIdAn
dDateAsync(Exception): " + ex.Message);
    }

    return found;
}
}
}

```

Приложение Б.44

Файл DbStudent_StudyGroupController.cs

```

using Microsoft.Data.SqlClient;
using MyORM.Controllers;
using System.Data.Common;
using Teaching_Assistant.MyORM.Models;

namespace Teaching_Assistant.MyORM.Controllers
{
    public class DbStudent_StudyGroupController :
        DbBaseController<DbStudent_StudyGroup>
    {
        public DbStudent_StudyGroupController(string
            providerName, string connectionString, string tableName = "") :
            base(providerName, connectionString, tableName)
        {
            _tableName = "Student_StudyGroup";

            /// <summary>
            /// Выборка всех студентов по коду группы
            /// </summary>
            /// <param name="studyGroupId"></param>
            /// <returns></returns>
            /// <exception cref="Exception"></exception>
            public async Task<List<DbStudent_StudyGroup>>
                Query_SelectByStudyGroupIdAsync(int studyGroupId)
            {
                List<DbStudent_StudyGroup> found = null;

                try
                {
                    string query = $"select * from {_tableName} where
                        StudyGroupId = @studyGroupId";

                    List<DbParameter> args = new List<DbParameter>();
                    args.Add(new SqlParameter() { ParameterName =
                        "@studyGroupId", Value = studyGroupId });

                    found = await Query_SelectsByQueryAsync(query,
                        args);

                }
                catch (Exception ex)
                {
                    throw new
                        Exception($"DbStudent_StudyGroupController.Query_SelectByStudyGr
                            oupId(Exception): {ex.Message}");
                }
                return found;
            }

            /// <summary>
            /// Выборка группы по коду студенту

```

```

    /// </summary>
    /// <param name="studyGroupId"></param>
    /// <returns></returns>
    /// <exception cref="Exception"></exception>
    public async Task<DbStudent_StudyGroup>
Query_SelectByStudentIdAsync(int studentId)
    {
        DbStudent_StudyGroup found = null;

        try
        {
            string query = $"select * from {_tableName} where
StudentId = @studentId";

            List<DbParameter> args = new List<DbParameter>();
            args.Add(new SqlParameter() { ParameterName =
"@studentId", Value = studentId });

            found = await Query_SelectByQueryAsync(query,
args);

        }
        catch (Exception ex)
        {
            throw new
Exception($"DbStudent_StudyGroupController.Query_SelectByStudyGr
oupId(Exception): {ex.Message}");
        }

        return found;
    }
}

```

Приложение Б.45

Файл DbStudent_DbStudyGroupsController.cs

```

using MyORM.Controllers;
using Teaching_Assistant.MyORM.Models;

namespace Teaching_Assistant.MyORM.Controllers
{
    public class DbStudyGroupsController :
DbBaseController<DbStudyGroup>
    {
        public DbStudyGroupsController(string providerName,
string connectionString, string tableName = "") :
base(providerName, connectionString, tableName)
        {
            _tableName = "StudyGroup";
        }
    }
}

```

Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ

Лист

169

```

public async Task<string> Query_GetNameByIdAsync(int id)
{
    DbStudyGroup found;

    try
    {
        found = await Query_SelectByIdAsync(id);
    }
    catch (Exception ex)
    {
        throw new
Exception($"DbStudyGroupsController.Query_GetNameById(Exception)
: {ex.Message}");
    }

    return found != null ? found.Name : "";
}

public async Task<int> Query_GetIdByNameAsync(string
name)
{
    DbStudyGroup found;

    try
    {
        string query = $"select * from {_tableName} where
Name = @param0";
        found = await Query_SelectByQueryAsync(query,
name);
    }
    catch (Exception ex)
    {
        throw new
Exception($"DbStudyGroupsController.Query_GetNameById(Exception)
: {ex.Message}");
    }

    return found != null ? found.Id : -1;
}
}
}

```

Приложение Б.46

Файл DbSubjectController.cs

```

using Microsoft.Data.SqlClient;
using MyORM.Controllers;
using System.Data.Common;
using Teaching_Assistant.MyORM.Models;

```

```

namespace Teaching_Assistant.MyORM.Controllers
{
    public class DbSubjectController :
    DbBaseController<DbSubject>
    {
        public DbSubjectController(string providerName, string
        connectionString, string tableName = "") : base(providerName,
        connectionString, tableName)
        {
            _tableName = "Subjects";
        }

        public async Task<List<DbSubject>>
        Query_SelectsByTeacherIdAsync(int teacherId)
        {
            List<DbSubject> found = new List<DbSubject>();

            try
            {
                string query = $"select * from {_tableName} where
                TeacherId = @param0";

                List<DbParameter> args = new List<DbParameter>();
                args.Add(new SqlParameter() { ParameterName =
                "@param0", Value = teacherId });

                found = await Query_SelectsByQueryAsync(query,
                args);
            }
            catch (Exception ex)
            {
                throw new
                Exception("DbSubjectController.Query_SelectByTeacherId(Exception
                ): " + ex.Message);
            }

            return found;
        }

        public async Task<DbSubject>
        Query_SelectByNameAsync(string name)
        {
            DbSubject found = null;

            try
            {
                string query = $"select * from {_tableName} where
                Name = @param0";
                found = await Query_SelectByQueryAsync(query,
                name);
            }
            catch (Exception ex)

```

```

        {
            throw new
Exception("DbSubjectController.Query_SelectByNameAsync(Exception
): " + ex.Message);
        }

        return found;
    }
}
}

```

Приложение Б.47

Файл DbUserController.cs

```

using MyORM.Models;

namespace MyORM.Controllers
{
    public class DbUserController : DbBaseController<DbUser>
    {
        public DbUserController(string provider_name, string
connection_string) :
            base(provider_name, connection_string)
        {
            _tableName = "Users";
        }

        /// <summary>
        /// Попробовать получить пользователя по TelegramId
        /// </summary>
        /// <param name="log"></param>
        /// <param name="found_user"></param>
        /// <returns></returns>
        /// <exception cref="Exception"></exception>
        public bool Query_TrySelectByTelegramId(long t_id, out
DbUser found_user)
        {
            try
            {
                string query = $"select * from {_tableName} where
TelegramId = @param0";
                found_user = Query_SelectByQuery(query,
t_id.ToString());
            }
            catch (Exception ex)
            {
                throw new
Exception($"DbUserController.Query_TrySelectByTelegramID(Excepti
on): {ex.Message}");
            }
        }
    }
}

```

```

        return found_user != null;
    }

    /// <summary>
    /// Получить пользователя по TelegramId
    /// </summary>
    /// <param name="t_id"></param>
    /// <returns></returns>
    /// <exception cref="Exception"></exception>
    public async Task<DbUser>
    Query_SelectByTelegramIdAsync(long t_id)
    {
        DbUser found_user = null;

        try
        {
            string query = $"select * from {_tableName} where
TelegramId = @param0";
            found_user = Query_SelectByQuery(query,
t_id.ToString());
        }
        catch (Exception ex)
        {
            throw new
Exception($"DbUserController.Query_TrySelectByTelegramID(Excepti
on): {ex.Message}");
        }

        return found_user;
    }

    /// <summary>
    /// Попробовать авторизироваться
    /// </summary>
    /// <param name="log"></param>
    /// <param name="pass"></param>
    /// <param name="login_user"></param>
    /// <returns></returns>
    /// <exception cref="Exception"></exception>
    public bool Query_TryLogin(long telegramId, string pass,
out DbUser login_user)
    {
        try
        {
            string query = $"select * from {_tableName} where
TelegramId = @param0 and Password = @param1";
            login_user = Query_SelectByQuery(query,
telegramId.ToString(), pass);
        }
        catch (Exception ex)
        {

```

```

        throw new
Exception($"DbUserController.Query_TryLogin(Exception):
{ex.Message}");
    }

    return login_user != null;
}

/// <summary>
/// Зарегистрировать пользователя и вернуть его данные с
/// актуальными значениями Id и подобного, которые задаются
на стороне БД
/// </summary>
/// <param name="new_user"></param>
/// <returns></returns>
/// <exception cref="Exception"></exception>
public bool Query_Registration(ref DbUser new_user)
{
    bool isDone = false;
    try
    {
        isDone = Query_Insert(new_user);
        Query_TrySelectByTelegramId(new_user.TelegramId,
out new_user);
    }
    catch (Exception ex)
    {
        throw new
Exception($"DbUserController.Query_Registration(Exception):
{ex.Message}");
    }

    return new_user != null && new_user.Id != 0;
}
}
}

```

Приложение В

(Обязательное)

Telegram-бот для организации учебного процесса

Презентация

					ПГУ 1.090301.26.001 ПЗ	Лист
						175
Изм.	Лист	№ докум	Подпись	Дата		

TELEGRAM-БОТ ДЛЯ ОРГАНИЗАЦИИ УЧЕБНОГО ПРОЦЕССА



Автор работы:
студент группы 21ВВП2 Борисова Анастасия Витальевна

Научный руководитель:
д.т.н., профессор кафедры «Вычислительная техника» Зинкин Сергей Александрович

Пенза, 2025

Цель проекта

Разработка Telegram-бота для автоматизации учебного процесса. Разработанный бот должен упрощать взаимодействие между студентами и преподавателями, автоматизировать рутинные задачи и обеспечить мгновенную рассылку уведомлений. Уникальность проекта заключается в разграничении прав доступа для различных ролей. Таких, как: администратор, преподаватель и студент.

Изм.	Лист	№ докум	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ

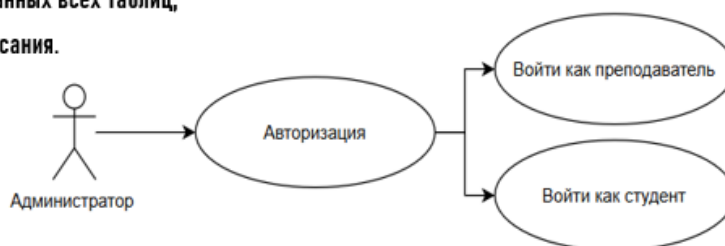
Лист

176

UML-диаграмма вариантов использования для администратора

Функциональные требования к функционалу администратора:

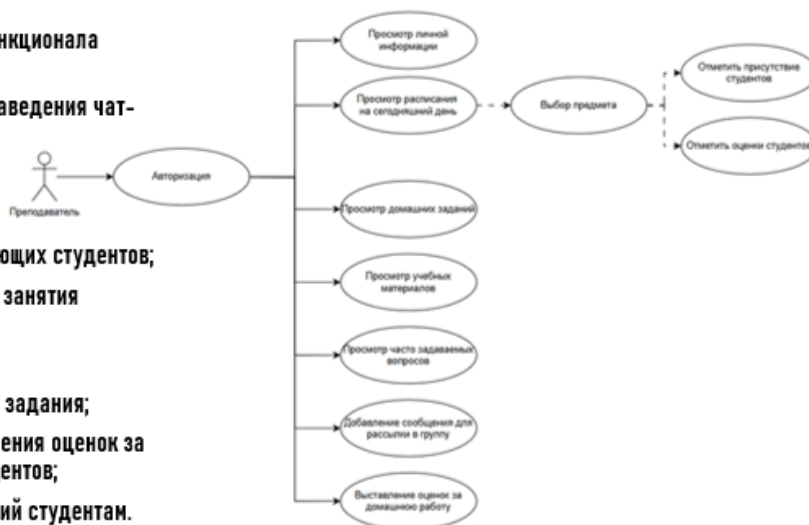
- авторизация в системе учебного заведения чат-бота;
- возможность войти под ролью студента для тестов функционала;
- возможность войти под ролью преподавателя для тестов функционала;
- просмотр всех таблиц базы данных;
- редактирование данных всех таблиц;
- составления расписания.



UML-диаграмма вариантов использования для преподавателя

Функциональные требования для функционала преподавателя:

- авторизация в системе учебного заведения чат-бота;
- просмотр личной информации;
- выбор занятия из расписания;
- возможность отметить присутствующих студентов;
- возможность выставить оценки за занятия студентам;
- просмотр домашних заданий;
- возможность добавлять домашние задания;
- возможность просмотра и выставления оценок за ответы на домашние задания студентов;
- возможность рассылки уведомлений студентам.



UML-диаграмма вариантов использования для студента

- Функциональные требования к функционалу студента:
- авторизация в системе учебного заведения чат-бота;
- просмотр личной информации;
- выбор занятия из расписания;
- просмотр полученных оценок;
- просмотр домашних заданий;
- просмотр часто задаваемых вопросов;
- возможность добавить ответ на домашнее задание.



Сравнение аналогов

Название ВУЗа	Ссылка на чат-бот	Рассылка уведомлений	Регистрация	Отметка посещаемости	Выставление оценок	Контактная информация
 ПензаГТУ	@PenzGTU58	+	-	-	-	-
 Университет Лобачев...	@lobachevsky_univ_ersity	+	-	-	-	-
 РГУ имени С.А. Есенин...	@rgu_esenina	+	-	-	-	-
 Бот абитуриентов Им...	@media_hse_bot	-	+	-	-	+
 Абитуриент РАНХиГС...	@abiturientuiu Bot	-	-	-	-	+

Стек технологий



Архитектура приложения



Спасибо за внимание!

					ПГУ 1.090301.26.001 ПЗ	Лист
						180
Изм.	Лист	№ докум	Подпись	Дата		