

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

**СТАРТАП-ПРОЕКТ ИНФОРМАЦИОННАЯ
СИСТЕМА «АВТОШКОЛА» НА ОСНОВЕ
ТЕЛЕГРАМ-БОТА**

Студент

 31.05.23
(подпись, дата)

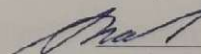
Яцков Владислав Вадимович
(ФИО полностью)

Руководитель

 31.05.23
(подпись, дата)

Карамышева Н.С.
(фамилия, инициалы)

Нормоконтролёр

 31.05.23
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 02.06.2023 г. № 14)

Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой Отличия (протокол заседания ГЭК от
06.06.2023 № 1)

Секретарь ГЭК



Е.И. Гурин

(подпись)

(фамилия, инициалы)

Пенза 2023

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

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

«26» апреля 2023 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Яцков Владислав Вадимович гр. 19ВВП2 факультета ФВТ

2. Тема работы: Стартап-проект информационная система «Автошкола» на основе телеграм-бота

Тема утверждена приказом ПГУ № 458/о от "26" апреля 2023 г.

3. Руководитель бакалаврской работы Карамышева Н.С.

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

Разработать автоматизированную информационную систему "Автошкола", использующую телеграм-бот. Составить бизнес-модель и финансовый план. Язык программирования python, среда разработки Visual Studio Code. Программа должна обладать интуитивно понятным интерфейсом в мессенджере telegram и использовать следующие технологии:

1.Telegram

2.Aiogram

3.Асинхронное
программирование

4.Сетевое программирование

5.Beautiful Soup

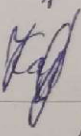
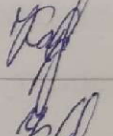
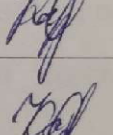
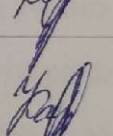
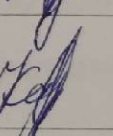
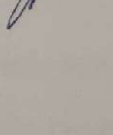
6.SMTP

5. Перечень подлежащих разработке вопросов

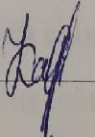
Требуется произвести анализ предметных областей: изучить существующие информационные системы, выделить их достоинства и недостатки. Выявить потребности пользователей. Выбрать язык программирования, среду разработки, СУБД, хостинг которые подойдут для создания телеграм-бота. Определить

архитектуру приложения и вспомогательные средства разработки. Изучить функционал библиотек для разработки информационной системы на основе телеграм-бота.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	26.04.2023 – 28.04.2023	
2	Анализ требований	32	29.04.2023 – 30.04.2023	
3	Создание макета графического интерфейса	30	31.04.2023 – 3.05.2023	
4	Выбор технологий, применяемых при проектировании и разработке	40	4.05.2023 – 8.05.2023	
5	Реализация графического интерфейса	66	9.05.2023 – 15.05.2023	
6	Разработка основной части программы	70	16.05.2023 – 23.05.2023	
7	Оформление пояснительной записки	54	24.05.2023 – 31.05.2023	

Дата выдачи задания "26" апреля 2023 г.

Руководитель бакалаврской работы  26. 04. 23 Карамышева Н.С.
(подпись, дата)

Задание к исполнению принял студент  - 26.04.23 Яцков В.В.
(подпись, дата)

Работу к защите допустить
Декан факультета  06.06.2023 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 166 листа, 94 рисунка, 1 формула, 7 источников, 3 приложения.

БОТ, ОБРАБОТКА СОСТОЯНИЙ, ПОДАЧА ЗАЯВКИ, УПРАВЛЕНИЕ РАСПИСАНИЕМ, ТЕСТИРОВАНИЕ, ПЛАТЕЖНАЯ СИСТЕМА, БАЗА ДАННЫХ, *AIOGRAM*, *PSYCOPG2*, *PYTHON*, *VS CODE*, *TIMEWEB*

Цель работы – разработать инструмент, позволяющий автоматизировать процесс работы автошколы.

Одним из путей достижения поставленной задачи является информационная система на основе телеграм бота, обеспечивающая выполнение следующих функций: авторизация пользователей, включая учеников, преподавателей и администрацию автошколы, просмотр и редактирование личной информации учеников и преподавателей, управление расписанием занятий, включая добавление, изменение и отмену занятий, просмотр прогресса учеников, включая пройденные этапы обучения и результаты тестирования, интеграция с системой оплаты, позволяющая ученикам оплачивать занятия, генерация чеков и выставление счетов для оплаты, обеспечивая прозрачность и документальное оформление финансовых операций.

В конечном итоге разработана информационная система на основе телеграм-бота, которая интегрируется с автошколой и обеспечивает эффективное управление и автоматизацию ее процессов. Информационная система полностью соответствующий цели данной выпускной квалификационной работы. В ходе достижения результата, были использованы следующие технологии: *python*, *aiogram*, *psycopg2*, *pdfkit*, *yagmail*, *BeautifulSoup*, *postgresSQL*, *telegram API*. Информационная система протестирована и готова к эксплуатации.

					ПГУ1.090301.39.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Стартап-проект информационная система «Автошкола» на основе телеграм-бота	Лит.	Лист	Листов
Разраб.		Яцков В.В.						
Провср.		Карамышева НС					2	166
Н. Контр		Бычков А.С.						
Утверд.		Митрохин М.А.						
						ФВТ, каф. ВТ гр. 19ВВП2		

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Стартап-проект информационная система «Автошкола» на основе телеграм-бота.....	8
1.1 Анализ предметной области	8
1.2 Анализ решения поставленной задачи	10
1.3 Анализ актуальности разработки	11
1.4 Анализ аналогов информационных систем	12
1.5 Анализ архитектуры и структура бота	16
1.6 Постановка задачи	19
1.7 Выбор инструментов разработки	20
1.7.1. Выбор мессенджера	20
1.7.2. Выбор языка программирования.....	21
1.7.3. Выбор СУБД.....	23
1.7.4. Выбор хостинга	25
1.7.5. Выбранные технологии	26
1.8 Реализация	27
1.8.1. Регистрация телеграм-бота	27
1.8.2. Работа с БД	30
1.8.3. Описание доступных функций	32
1.8.4. Перенос ИС на сервер.....	64
1.8.5. Описание программы.....	66
2 Безопасность жизнедеятельности.....	82
2.1 Правила безопасности при использовании ПК.....	82

2.2	Правила безопасности при использовании смартфона.....	83
3	Стартап проект	85
3.1	Сегменты ЦА.....	85
3.2	Ключевые процессы.....	86
3.3	Каналы взаимодействия	88
3.4	Источники дохода.....	89
3.5	Ключевые ресурсы.....	91
3.6	Риски проекта.....	Ошибка! Закладка не определена.
	Заключение	94
	Список используемых источников.....	96
	Приложение А – UML – диаграммы	997
	Приложение Б–Листинг программы	102
	ПриложениеВ– Презентация.....	143

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения
БД (база данных) - структурированная коллекция данных, организованных и хранимых в электронном виде

ГИБДД (государственная инспекция безопасности дорожного движения) - орган исполнительной власти, ответственный за контроль и надзор за соблюдением Правил дорожного движения

ИС (информационная система) - комплексное информационное программно-аппаратное решение

ПДД (правила дорожного движения) - нормативный документ, содержащий правила и требования, регулирующие поведение участников дорожного движения на дорогах

ПК (персональный компьютер) - электронное устройство, предназначенное для индивидуального использования пользователем

СУБД (система управления базами данных) - программное обеспечение, предназначенное для создания, управления и обработки баз данных

ЯП (язык программирования) - формальный язык, используемый для написания компьютерных программ, которые определяют последовательность действий, выполняемых компьютером

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

IDE (Integrated development environment) – интегрированная среда разработки

PDF (Portable Document Format) - формат электронных документов, обеспечивающий точное сохранение оригинального вида и форматирования документа, и широко используемый для создания и обмена документами.

SQL (Structured Query Language) – язык структурированных запросов

UML (Unified Modeling Language) – унифицированный язык моделирования

Введение

В современном мире, когда автомобильное движение становится все более интенсивным и сложным, важным является обеспечение высокого уровня подготовки водителей. Автошколы сталкиваются с проблемой неэффективности работы, связанной с большим количеством рутинных задач, а также необходимостью повышения качества обучения и безопасности на дорогах. Информационные технологии в этой сфере могут существенно упростить и ускорить процесс обучения.

В качестве одного из инструментов автоматизации работы автошкол можно использовать телеграм-боты, которые позволяют осуществлять множество задач, связанных с обучением. Телеграм-боты являются удобными, доступными и современными инструментами, которые широко используются в различных сферах бизнеса и образования. Кроме того, использование телеграм-бота имеет свои преимущества по сравнению с использованием приложения. Во-первых, телеграм-бот удобен в использовании, так как не требует дополнительной установки на смартфон или компьютер. Достаточно иметь аккаунт в телеграме, и пользователь сможет взаимодействовать с ботом через мессенджер. Во-вторых, использование телеграм-бота гарантирует безопасность пользователей, так как все данные передаются по защищенному протоколу HTTPS, а также соблюдается конфиденциальность личной информации.

Информационная система «Автошкола» на основе телеграм-бота, которую я рассмотрел в данной выпускной квалификационной работе, представляет собой систему, которая объединяет все необходимые функции и задачи, связанные с обучением в автошколе. Она позволяет автоматизировать рутинные задачи, такие как изменение личных данных, ведение расписания, прием и обработку платежей, облегчить работу преподавателей, а также повысить качество обучения с помощью тестирования по вопросам ПДД.

Цели выпускной квалификационной работы:

– освоение актуальных и современных средств программной разработки;

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		6

– изучить существующие проблемы и неэффективности работы автошкол, связанные с рутинными задачами, качеством обучения и безопасностью на дорогах;

– исследовать возможности применения телеграм-ботов в автошколах для автоматизации задач и улучшения процесса обучения;

– разработать информационную систему «Автошкола» на основе телеграм-бота, интегрирующую необходимые функции и задачи, связанные с обучением;

– определить требования к функциональности и интерфейсу системы, учитывая потребности автошкол и пользователей;

– реализовать прототип системы и провести тестирование его функциональности, надежности и безопасности;

– оценить эффективность внедрения информационной системы «Автошкола» на основе телеграм-бота в практической работе автошколы;

– проанализировать полученные результаты и сделать выводы о преимуществах и потенциале использования телеграм-ботов в автошколах;

– предложить рекомендации по дальнейшему развитию и совершенствованию информационной системы, а также возможным расширениям функциональности на основе обратной связи от пользователей;

– вывести обобщенные рекомендации по применению информационных технологий, включая телеграм-боты, для повышения эффективности работы автошкол и безопасности на дорогах.

1 Стартап-проект информационная система «Автошкола» на основе телеграм-бота

1.1 Анализ предметной области

Для того чтобы разработать качественную и функциональную систему, необходимо проанализировать все аспекты работы автошколы и определить ее потребности и требования. В рамках анализа была произведена оценка процессов работы автошколы и ее системы управления. Были определены потребности и требования пользователей, а также выявлены задачи, которые могут быть оптимизированы с помощью телеграм-бота. На основе проведенного анализа были сформулированы требования к новой системе, которая должна быть удобной в использовании, безопасной и эффективной.

Анализ работы автошколы является ключевым этапом в разработке информационной системы «Автошкола» на основе телеграм-бота. Для этого были проведены исследования, которые позволили понять, как работают автошколы, и выявить основные проблемы и недостатки, которые могут быть решены с помощью информационной системы на основе телеграм-бота.

Первым этапом анализа было выявление основных процессов работы автошколы. Это включало в себя описание процесса подачи заявок на обучение, прохождения теоретических и практических занятий, сдачи экзаменов, оформления документов, а также требований для сдачи экзаменов. Каждый процесс был разбит на отдельные этапы, которые были детально проанализированы. Для сбора заявок на обучение, кандидату в водители требовалось, прийти в автошколу, просмотреть цены и виды услуг, оставить свои контакты для связи. В нашем случае все делается удаленно, и пользователю всего лишь нужно сделать пару кликов в боте и ждать звонка от оператора с информацией для начала обучения. Управление календарем раньше осуществлялось только по телефону, сейчас пользователь и инструкторы имеют удобный цифровой интерфейс для управления расписанием. Также

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

усовершенствовалась система записи на экзамен, сейчас кандидат в водители в праве записаться на один экзамен практический или теоретический экзамен, только при условии, что нужные часы занятий набраны. Это не только ускоряет и облегчает процесс записи, но и устраняет большие очереди, которые были связаны с тем, что один и тот же человек записывался на несколько экзаменов подряд. Конечно, изменения затронули и хранение документов, теперь хранить паспорт и медицинскую справку можно электронно, прямо в боте. Нужно всего лишь прислать номер документа, а администратор автошколы проверит его на подлинность. Всю нужную информацию теперь можно узнавать непосредственно в телеграмм-боте, это тоже упрощает и облегчает процесс обучения.

Далее были определены потребности и требования пользователей автошколы. Они включали в себя пожелания учеников, преподавателей, администрации и других заинтересованных сторон. Такие как: удобство использования системы, безопасность, скорость, доступность и надежность.

Также были выявлены проблемы, связанные с текущей системой управления автошколы. К ним относилась сложность управления расписанием занятий, ручное учет платежей, организация тестирования по вопросам ПДД и многие другие. Все эти проблемы могут быть решены с помощью информационной системы на основе телеграм-бота.

На основе проведенного анализа были сформулированы требования к новой системе. Ими стали удобство использования системы, безопасность, доступность и надежность. Также было определено, что система должна быть автоматизированной, чтобы минимизировать ручной труд и сократить время, затрачиваемое на выполнение рутинных задач.

В результате анализа работы автошколы стало понятно, что информационная система на основе телеграм-бота может оптимизировать ряд требований и пожеланий, указанных выше. Разработка такой системы позволит

автошколе повысить качество обучения, улучшить управление учебным процессом.

1.2 Анализ решения поставленной задачи

Telegram-бот — это программа, созданная на платформе Telegram, которая позволяет взаимодействовать с пользователем через сообщения и функционал чата. Об этом говорится в [1]. Боты позволяют выполнять различные функции, такие как управление информацией, обработка заказов, проведение оплат и т.д. Сегодня телеграм-боты широко используются компаниями разных отраслей, в том числе банками, ресторанами, медицинскими учреждениями, авиакомпаниями, онлайн-магазинами и т.д. Они используют ботов для автоматизации обслуживания клиентов и упрощения процессов внутри компаний. Рассмотрим некоторые из преимуществ телеграм-бота для создания крупной информационной системы:

- доступность. Телеграм-бот доступен для всех пользователей мессенджера Telegram, что делает его очень удобным для использования;
- простота использования. Бот позволяет пользователю взаимодействовать с системой без необходимости вводить команды вручную;
- быстрота работы. Телеграм-бот обрабатывает запросы мгновенно, что позволяет пользователям быстро получать необходимую информацию;
- автоматизация процессов. Телеграм-бот позволяет автоматизировать процессы, такие как запись на занятия, управление расписанием, прием и обработка платежей, тестирование и т.д;
- безопасность. Все данные, передаваемые через телеграм-бот, защищены с помощью протокола шифрования *TLS*;
- надежность. Телеграм-боты работают на серверах *Telegram*, что обеспечивает их высокую надежность и стабильность;

– возможность многократного использования. Телеграм-бот может использоваться не только для создания Информационной системы "Автошкола", но и для других систем;

– низкие затраты. Создание телеграм-бота требует намного меньше времени и денег, чем создание приложения;

– гибкость. Телеграм-бот может быть настроен на различные задачи и требования пользователей;

– интеграция. Телеграм-бот может быть интегрирован с другими системами и приложениями;

– универсальность. Телеграм-бот может использоваться для различных целей, включая образование, бизнес, развлечения и т.д.

– широкий функционал. Телеграм-бот может предоставлять пользователю широкий спектр функций, таких как оплата, отправка файлов, рассылка сообщений и т.д;

– возможность персонализации. Телеграм-бот может быть настроен на индивидуальные потребности пользователя.

Улучшение клиентского сервиса. Телеграм-боты могут значительно улучшить клиентский сервис, предоставляя быстрый и эффективный способ связи с компанией. Клиенты могут легко получать ответы на свои вопросы, заказывать услуги и получать поддержку в любое удобное для них время.

1.3 Анализ актуальности разработки

Разработка информационной системы «Автошкола» на основе телеграм-бота является актуальной в настоящее время из-за многих факторов.

Во-первых, автошколы представляют собой важный элемент общественной инфраструктуры, которая обеспечивает безопасность дорожного движения. Однако, управление автошколой может быть неэффективным и трудоемким, особенно если процессы управления не автоматизированы.

Во-вторых, современные технологии значительно упрощают и ускоряют процессы управления бизнесом. Это сказано в [2]. Использование телеграм-бота в качестве основы для информационной системы автошколы позволит значительно упростить процессы записи на занятия, управления расписанием, приема и обработки платежей, тестирования по вопросам ПДД и других аспектов управления автошколой.

В-третьих, в настоящее время телеграм-боты все более популярны и широко используются в бизнесе. Они обеспечивают быстрый и удобный доступ к информации и услугам, а также позволяют осуществлять диалоговое взаимодействие с клиентами.

Также следует отметить, что в настоящее время все больше людей предпочитают осуществлять свои операции и действия без прямой коммуникации. В этом контексте, создание информационной системы на основе телеграм-бота для автошколы позволит предоставить пользователям дополнительный удобный инструмент для быстрого и безопасного управления своим обучением. Таким образом, предложенный проект имеет высокую значимость и актуальность для пользователей автошколы, а также для развития и совершенствования инфраструктуры образовательных учреждений в целом.

1.4 Анализ аналогов информационных систем

Обзор аналогов является важной частью разработки информационной системы, так как позволяет изучить подобные решения и определить их преимущества и недостатки. В нашем случае, я исследовал рынок информационных систем для автошкол, с целью выявления существующих аналогов на основе телеграм-ботов. Несмотря на то, что полноценной информационной системы на основе телеграм-бота не существует, можно провести сравнение с существующими мобильными приложениями и сайтами.

В качестве аналогов можно рассмотреть такое мобильное приложение, как "Автоконтроль" от автошколы «Профессионал», сайт автошколы «Твиспо»,

телеграм-боты автошколы «Лидер» и «Джек». Эти решения предоставляют возможность записаться на занятия, управлять расписанием и получать информацию о прохождении курса обучения.

Первый аналог - мобильное приложение "Автоконтроль" от автошколы «Профессионал» - предоставляет возможность записаться на занятия, управлять расписанием и получать информацию о прохождении курса обучения. Однако, насколько мне известно, это приложение не предоставляет функции по приему и обработке платежей, а также не включает тестирование по вопросам ПДД. Пример работы аналога «Автоконтроль» рассмотрен на рисунке 1.1.

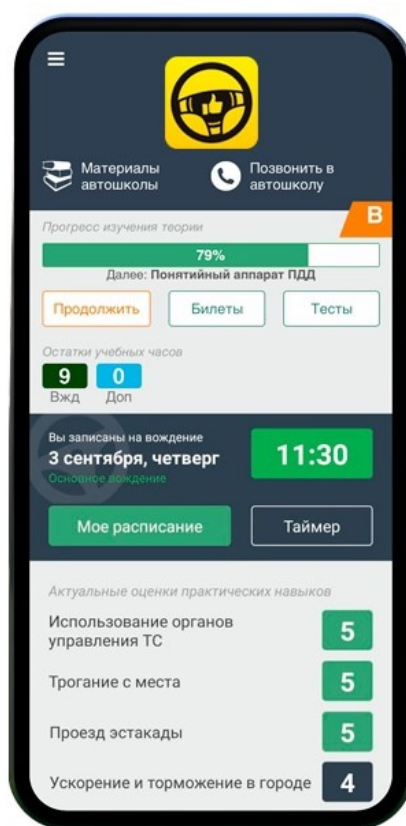


Рисунок 1.1 – Обзор аналога «Автоконтроль»

Сайт автошколы «Твиспо» также предоставляет возможность записаться на занятия, управлять расписанием и получать информацию о прохождении курса обучения. Однако, на сайте также нет функций по приему и обработке платежей, а также возможности прохождения тестирования по вопросам ПДД. Пример работы аналога «Твиспо» рассмотрен на рисунке 1.2.



Рисунок 1.2 – Обзор аналога «Твиспо»

Автошколы «Лидер» и «Джек» также предоставляют свои информационные системы в виде телеграм-ботов. Бот автошколы «Лидер» предназначен для сбора заявок на запись на обучение, а также предлагает информацию об услугах и расписании автошколы. В свою очередь, телеграм-бот от автошколы «Джек» является новостным каналом, который информирует своих подписчиков об акциях и скидках, а также изменениях в ПДД. Однако, эти боты не обладают полным функционалом, который предлагает будущая информационная система на основе телеграм-бота. Пример работы аналогов «Лидер» и «Джек» рассмотрен на рисунке 1.3.

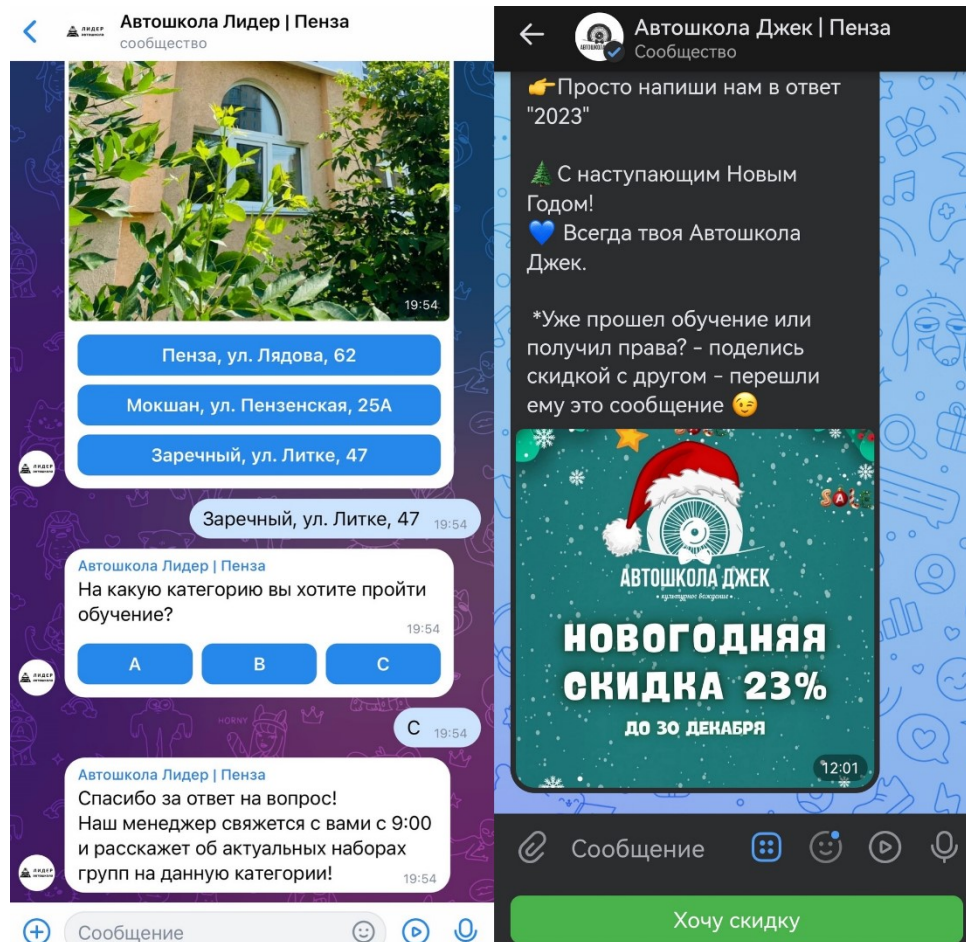


Рисунок 1.3 – Обзор аналогов «Лидер» и «Джек»

В целом, все эти аналоги предоставляют некоторые функциональные возможности для управления автошколами, но не обладают всеми необходимыми функциями. Поэтому, разработка информационной системы на основе телеграм-бота может быть более привлекательным решением для автошкол, так как такой проект обладает преимуществами в плане безопасности и надежности, а также является более удобным для пользователей.

Разработка полной информационной системы автошколы на основе телеграм-бота предоставляет ряд преимуществ перед аналогами. Во-первых, использование бота удобнее и быстрее, поскольку не требуется скачивание и установка приложения, а также не нужно заходить на сайт через браузер и вводить данные для авторизации. Бот работает непосредственно в мессенджере, что удобно и практично.

Кроме того, телеграм-боты обладают высоким уровнем надежности и безопасности, так как весь трафик шифруется и передается через защищенное соединение. Это особенно важно при работе с личной информацией, такой как данные пользователей, которые вводятся при регистрации, а также при работе с платежными картами.

Также отличием использования телеграм-бота в качестве информационной системы является то, что пользователь может получить необходимую информацию незамедлительно, без необходимости загрузки мобильного приложения или открытия веб-сайта. Боты в Telegram работают очень быстро и имеют высокую производительность, что позволяет пользователям мгновенно получать ответы на свои запросы.

Наконец, разработка и поддержка телеграм-бота может быть более выгодной финансово по сравнению с мобильными приложениями и сайтами. В отличие от них, создание телеграм-бота не требует больших затрат на разработку и поддержку разных версий для разных операционных систем. Также нет необходимости оплачивать хостинг и доменное имя, как в случае с сайтом. Телеграм-бот может быть размещен на бесплатной платформе и не требует больших затрат на обслуживание.

В итоге, использование телеграм-бота в качестве информационной системы автошколы предоставляет более быстрый, удобный и безопасный способ взаимодействия с клиентами, а также более экономичный способ создания и поддержки системы.

1.5 Анализ архитектуры и структура бота

Архитектура информационной системы на основе телеграм-бота для автошколы состоит из трех основных компонентов: клиентского приложения в виде телеграм-бота, серверного приложения и базы данных.

Клиентское приложение представлено в виде телеграм-бота, который является интерфейсом пользователя. Бот предоставляет ученикам возможность

записаться на занятия, работать с платежными системами, получать доступ к экзаменационной информации, проходить теоретическое тестирование, а также следить за своей успеваемостью. Подробную диаграмму прецедентов можно изучить на рисунке 1.4.

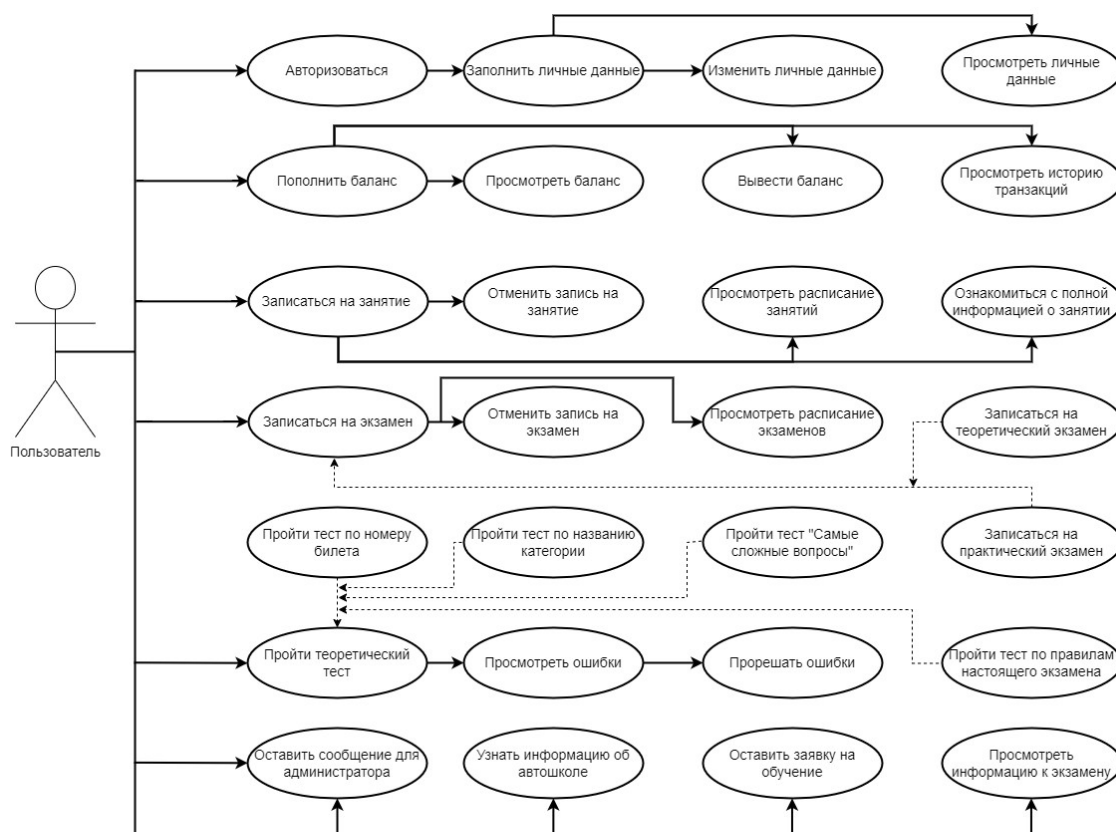


Рисунок 1.4– UML диаграмма прецедентов интерфейса пользователя

Кроме того, в состав клиентского приложения входят интерфейсы инструктора и администратора. Меню инструктора позволяет работать с расписанием занятий, отмечать прогресс учеников и устанавливать с ними связь. Администраторский интерфейс обеспечивает возможность управления данными всех пользователей системы, включая управление расписанием, доступом к материалам, информации о транзакциях, а также всей системы автошколы. Подробнее диаграммы прецедентов этих интерфейсов можно изучить на рисунках 1.5, 1.6.

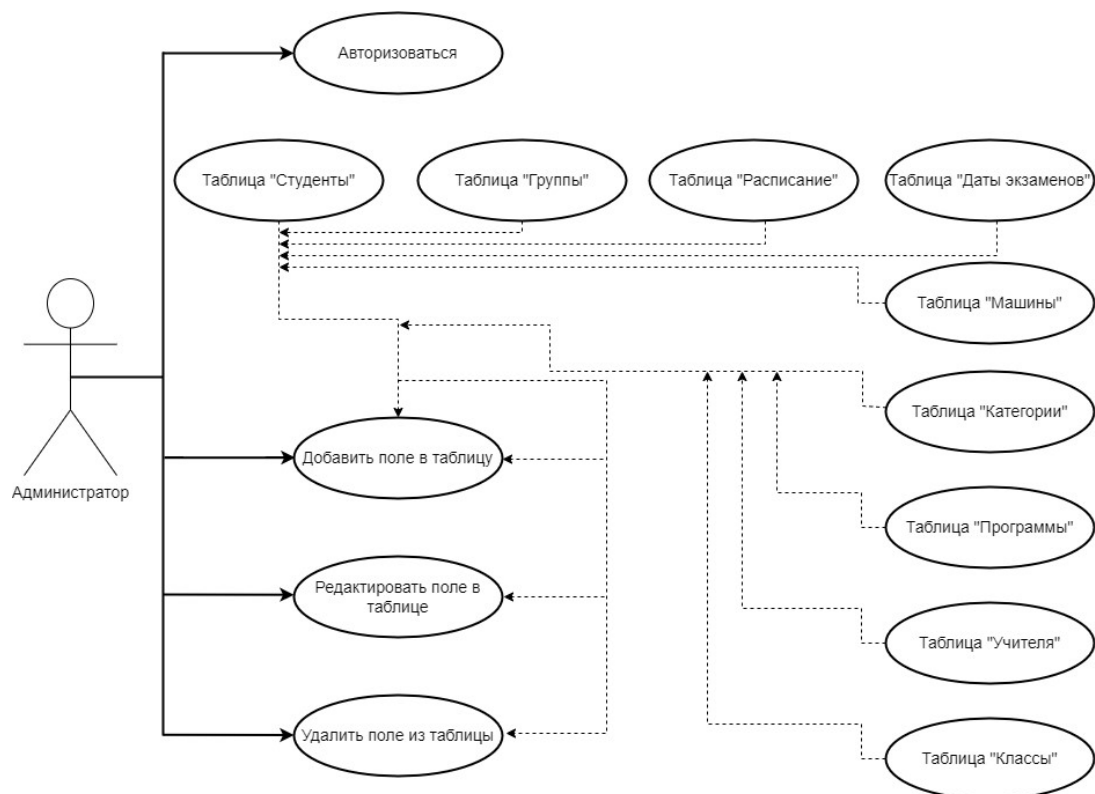


Рисунок 1.5 – UML диаграмма прецедентов интерфейса администратора

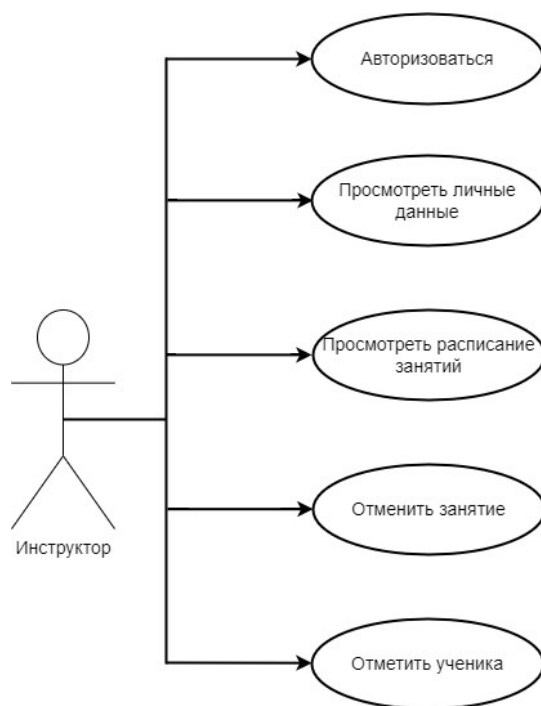


Рисунок 1.6 – UML диаграмма прецедентов интерфейса инструктора

Для четкого понимания взаимодействия компонентов бота, необходимо создать *UML*-диаграмму последовательности. На рисунке 1.7 представлена диаграмма последовательности.

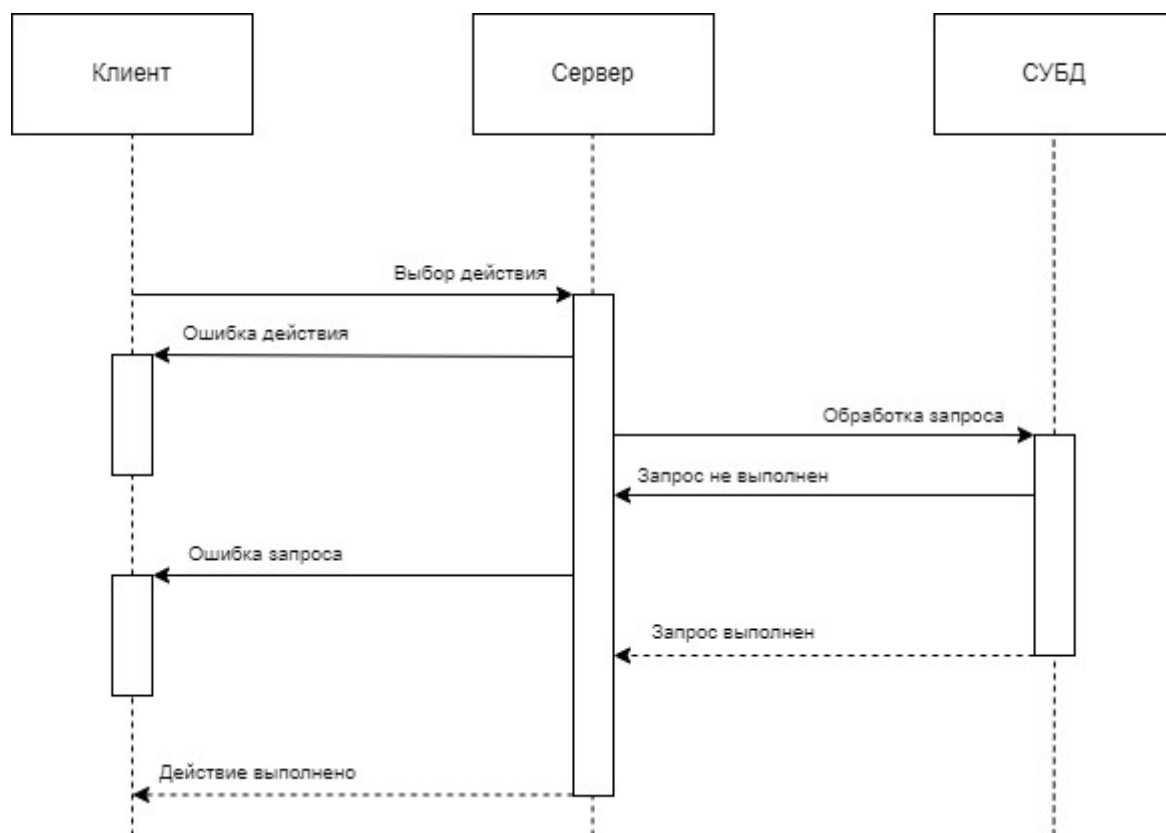


Рисунок 1.7 – *UML*-диаграмма последовательности

1.6 Постановка задачи

Необходимо разработать информационную систему на основе телеграм-бота для автошколы, позволяющую сократить время и затраты на проведение занятий, а также повысить уровень обучения учащихся. Система должна предоставлять возможность записи на занятия, отслеживание прогресса обучения, взаимодействие между учениками и инструкторами, а также генерацию чеков, что делает учет средств надежным и прозрачным.

Основная цель системы - упрощение процесса обучения и повышение его качества за счет автоматизации рутинных задач, а также усиления коммуникации между учениками и инструкторами. Информационная система должна быть

удобной и легкой в использовании, а также обладать высокой надежностью и безопасностью данных.

Кроме того, система должна быть масштабируемой, чтобы ее можно было легко расширять в будущем, добавляя новые функции и возможности. Она должна быть простой в обслуживании и экономически эффективной, с минимальными затратами на разработку и поддержку. В итоге, информационная система должна увеличить конкурентоспособность автошколы и обеспечить ее успех на рынке.

1.7 Выбор инструментов разработки

Правильный выбор инструментов разработки является одним из ключевых факторов при создании информационной системы. Это позволяет оптимизировать процесс разработки, сократить время и затраты на создание продукта, а также обеспечить качество и безопасность разработки. Неправильный выбор инструментов может привести к проблемам в процессе разработки, а также к потенциальным проблемам при использовании продукта. Поэтому важно правильно подойти к выбору инструментов разработки, учитывая конкретные требования и характеристики проекта.

В случае написания телеграм-бота для автошколы, можно использовать различные инструменты и технологии в зависимости от требований и целей проекта.

1.7.1. Выбор мессенджера

От выбора мессенджера зависит удобство использования для пользователей и функциональность системы в целом. В данном случае, выбор площадки для информационной системы был сделан в пользу телеграма.

Telegram является одним из самых популярных мессенджеров в мире, что гарантирует большое количество пользователей и удобство использования для них. Также телеграм-боты имеют ряд преимуществ перед другими мессенджерами, такие как высокая скорость и производительность, высокий

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		20

уровень безопасности и надежности, а также возможность интеграции с другими сервисами и системами.

Кроме того, *Telegram* предоставляет широкие возможности для разработчиков, такие как бесплатная и открытая платформа для создания ботов, гибкие настройки и возможность расширения функциональности [3]. Также есть возможность создавать группы для пользователей, что позволяет общаться с несколькими пользователями одновременно и предоставлять им доступ к различным функциям.

Итак, выбор телеграм-бота для разработки информационной системы автошколы был сделан в связи с его широкими возможностями, высокой скоростью и производительностью, а также удобством использования для пользователей и безопасностью.

1.7.2. Выбор языка программирования

Каждый язык программирования обладает своими уникальными особенностями, которые существенно влияют на процесс разработки и конечный результат. Поэтому критически важно тщательно подобрать язык программирования, который наиболее соответствует требованиям проекта и обеспечит успешную реализацию информационной системы.

Один из языков программирования, на котором можно разрабатывать телеграм-бота, — это *Python*. *Python* — это высокоуровневый язык программирования, который имеет простой и понятный синтаксис. Он относится к языкам с динамической типизацией, что означает, что тип переменной определяется автоматически во время выполнения программы[4].

Среди других языков программирования, которые можно использовать для разработки телеграм-ботов, можно отметить:

– *JavaScript* – это язык, который широко используется в веб-разработке и может быть использован для создания телеграм-ботов. *JavaScript* относится к языкам с динамической типизацией и имеет синтаксис, похожий на C. В отличие от *Python*, *JavaScript* не имеет встроенных возможностей для работы с матрицами

и научными вычислениями, но имеет большое количество библиотек для работы с веб-интерфейсами и взаимодействием с сервером;

– *PHP*— это язык программирования, который часто используется для создания веб-приложений и сайтов. Он также может быть использован для разработки телеграм-ботов. *PHP* относится к языкам с динамической типизацией и имеет синтаксис, похожий на C. Однако, в отличие от Python, PHP не имеет такой широкой базы научных библиотек и используется преимущественно для веб-разработки;

– *Java*— это язык программирования, который широко используется для создания приложений на *Android* и других мобильных платформах. Java относится к языкам с жесткой типизацией и имеет синтаксис, похожий на C++. В отличие от *Python*, *Java* предлагает компиляцию кода, что обеспечивает более высокую производительность. Однако, *Java* требует большого объема кода, что может затруднить процесс разработки телеграм-ботов в сравнении с *Python*[5];

Я выбрал *Python* для разработки телеграм-бота, так как этот язык программирования имеет множество преимуществ перед другими языками.

Во-первых, *Python* имеет простой и легко читаемый синтаксис, что позволяет разработчикам быстрее писать код и легче его поддерживать. Это особенно важно для команд, которые работают над одним проектом, так как простота и понятность кода упрощает коммуникацию между разработчиками и уменьшает возможность ошибок.

Во-вторых, *Python* имеет большое количество библиотек и фреймворков, которые значительно ускоряют процесс разработки и упрощают написание сложного кода. В частности, для разработки телеграм-бота мы использовали библиотеку *python-telegram-bot*, которая облегчает работу с *API Telegram*.

В-третьих, *Python* имеет высокую производительность и быстроедействие, что позволяет быстро обрабатывать большие объемы данных и выполнять сложные вычисления. Это особенно важно при разработке информационных систем, работающих с большими объемами информации, такими как базы

данных, аналитические инструменты и машинное обучение. Благодаря своей оптимизации и высокой производительности, *Python* позволяет ускорить процессы обработки данных и повысить эффективность работы системы в целом.

В-четвертых, *Python* является одним из наиболее популярных языков программирования в мире, и у него есть активное сообщество разработчиков, которые создают новые библиотеки и инструменты. Это позволяет быстро находить решения для проблем, с которыми сталкиваются разработчики, и делает *Python* одним из наиболее удобных языков для работы.

В целом, *Python* обладает всеми необходимыми функциями и инструментами для разработки чат-ботов, включая работу с *API*, обработку данных и создание пользовательского интерфейса. Кроме того, его простота и удобство делают его идеальным выбором для командных проектов, которые требуют быстрой и эффективной разработки.

Именно поэтому я выбрал *Python* для разработки информационной системы автошколы на базе телеграм-бота.

1.7.3. Выбор СУБД

Выбор СУБД является критически важным моментом при разработке программного обеспечения, поскольку это позволяет эффективно хранить, организовывать, обрабатывать и управлять большим объемом данных, которые используются в приложении.

СУБД являются программными системами, которые позволяют управлять базами данных. Они предоставляют интерфейс для создания, изменения и извлечения информации из баз данных. Существует множество различных СУБД, и каждая из них имеет свои уникальные особенности и преимущества.

Один из главных факторов при выборе СУБД является ее совместимость с конкретными требованиями приложения. Например, если приложение будет использоваться для работы с большим объемом данных, то необходимо выбрать СУБД, которая обеспечивает высокую производительность и масштабируемость.

Кроме того, важно учитывать такие факторы, как безопасность, доступность, надежность, администрирование и поддержка. Также нужно учитывать, какие языки программирования и технологии поддерживаются СУБД, чтобы быть уверенным в совместимости с приложением.

Например, существуют реляционные СУБД, которые обеспечивают надежное хранение и управление структурированными данными. Также существуют NoSQL СУБД, которые позволяют хранить и управлять неструктурированными данными.

Одной из наиболее популярных СУБД является Oracle, которая широко используется в крупных корпоративных системах и бизнес-приложениях. Oracle отличается высокой производительностью, масштабируемостью и надежностью, а также поддерживает широкий спектр функциональных возможностей, включая управление транзакциями, безопасность и бэкапы. Однако, использование Oracle может быть достаточно затратным из-за высокой стоимости лицензирования и поддержки, что делает ее не оптимальным выбором для небольших проектов и стартапов.

Еще одной популярной СУБД является *MySQL*. Она также является свободной и открытой системой управления базами данных и широко используется для разработки веб-приложений. *MySQL* имеет хорошую производительность, но имеет ограниченную поддержку транзакций и масштабирования.

СУБД *MongoDB* использует документы для хранения данных вместо таблиц. Она обеспечивает гибкость и масштабируемость при работе с большим объемом неструктурированных данных. *MongoDB* также обеспечивает высокую доступность и отказоустойчивость, что делает ее хорошим выбором для приложений, которые нуждаются в быстром доступе к данным.

Однако, для нашей информационной системы на основе телеграм-бота мы выбрали *PostgreSQL*, которая является свободной и открытой системой управления базами данных, которая хорошо подходит для крупных и сложных

проектов. *PostgreSQL* имеет множество функций, включая поддержку транзакций, полнотекстовый поиск, встроенную репликацию и масштабирование [6]. Она также имеет отличную производительность и надежность, что делает ее идеальной для бизнес-приложений. Кроме того, *PostgreSQL* имеет высокую производительность и надежность, что позволяет обрабатывать большие объемы данных без потери производительности. Также *PostgreSQL* имеет хорошую поддержку сообщества и широкий набор инструментов для работы с данными, что упрощает разработку и поддержку нашей информационной системы.

1.7.4. Выбор хостинга

Выбор подходящего хостинга является важным этапом при переносе ИС на сервер. Хостинг представляет собой услугу предоставления вычислительных ресурсов и пространства для размещения ИС в интернете. Главными качествами для выбора подходящего сервиса стали: надежность, высокая доступность, масштабируемость, безопасность, качественная техническая поддержка, а также финансовая выгодность использования.

Я выбрал сервис *TimeWeb.Cloud*, поскольку он предлагает ряд преимуществ, соответствующих моим требованиям.

Во-первых, *TimeWeb.Cloud* обладает высокой надежностью и обеспечивает стабильную доступность моей информационной системы, что критически важно для ее бесперебойной работы.

Во-вторых, этот хостинг предлагает масштабируемое решение, позволяющее легко адаптировать мою систему к возрастающим потребностям и нагрузке.

Кроме того, *TimeWeb.Cloud* обеспечивает высокий уровень безопасности, что гарантирует защиту моих данных и обеспечивает конфиденциальность информации.

Важным фактором является также качественная техническая поддержка, которую предлагает *TimeWeb.Cloud*, готовая оказать помощь и решить любые вопросы, связанные с хостингом.

Наконец, главным из преимуществ *TimeWeb.Cloud* стала доступная цена, что сделало его окончательным выбором с точки зрения соотношения цены и качества предоставляемых услуг.

1.7.5. Выбранные технологии

В ходе проведенных анализов, было принято следующие решения:

- выбрать *Telegram* как мессенджер для базирования чат-бота;
- выбрать среду разработки *VS Code* и язык программирования *python* с последующим применением библиотеки *psycopg2* и *aiogram*;
- выбрать *PostgreSQL* для управления базами данных;
- выбрать *TimeWeb.Cloud* в качестве хостинга сервера.

1.8 Реализация

1.8.1. Регистрация телеграм-бота

Регистрация телеграм-бота является первым шагом для создания и интеграции бота в информационную систему. В этом разделе будет описана подробная процедура регистрации телеграм-бота.

Для начала нужно открыть официальное приложение *Telegram* на мобильном устройстве или компьютере и создать новый аккаунт, если его еще нет. Если уже есть аккаунт, то нужно войти в него. Главное меню показано на рисунке 1.8.

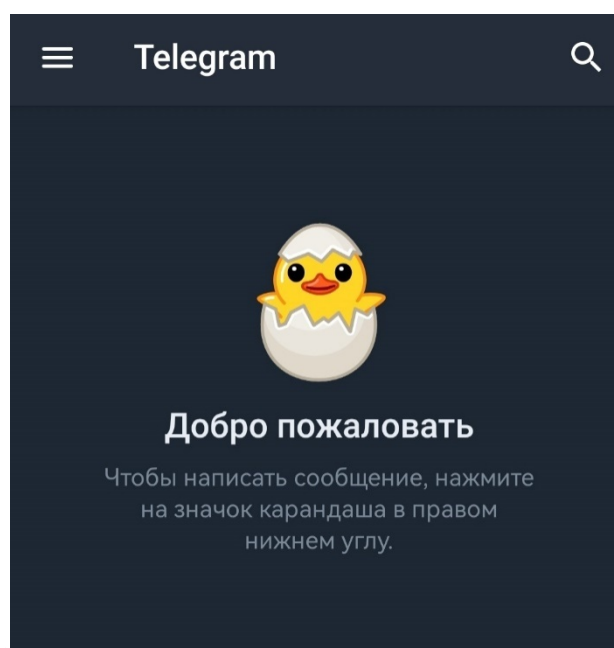


Рисунок 1.8– Главное меню

Далее нужно найти в поиске *Telegram* бота с именем "*BotFather*" — это официальный бот *Telegram* для создания новых ботов. Пример показан на рисунке 1.9.



Рисунок 1.9 –Поиск *BotFather*

После того, как чат с ботом открыт, необходимо начать диалог с ботом командой “/start”. Это видно на рисунке 1.10.



Рисунок 1.10 – Начало диалога с *BotFather*

Для создания любого бота необходимо выполнить команду */newbot*, а затем присвоить боту имя, которое будет отображаться в поиске, и логин, который будет отображаться в ссылке. Как это сделал я, показано на рисунке 1.11.

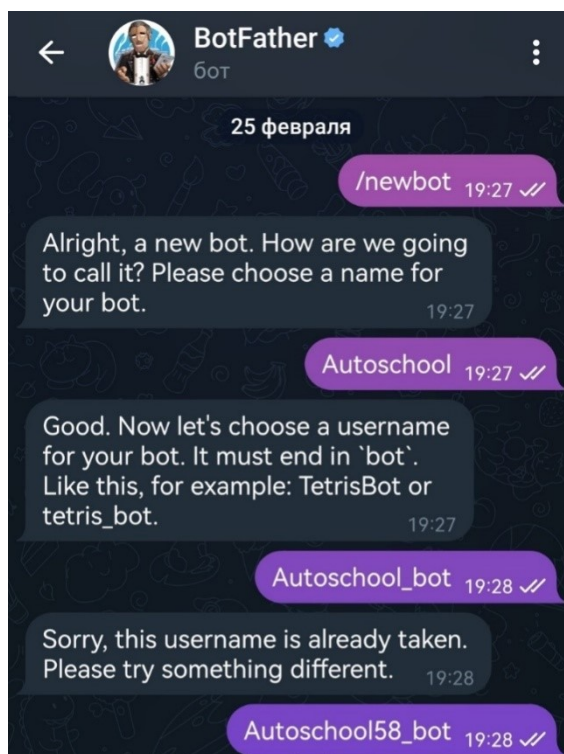


Рисунок 1.11 – Выбор имени бота

Далее *BotFather* предоставляет токен, который является ключом доступа к вашему телеграм-боту. Необходимо сохранить полученный от *BotFather* токен в надежном и безопасном месте. Токен будет использоваться для аутентификации вашего бота при взаимодействии с *Telegram API*. Информация, полученная от бота показана на рисунке 1.12.



Рисунок 1.12 – Получение ссылки на бота

После завершения всех этих шагов телеграм-бот успешно зарегистрирован. Теперь можно использовать токен бота для настройки и интеграции бота в информационную систему автошколы.

1.8.2. Работа с БД

Для хранения информации необходимо реализовать базу данных со следующими таблицами:

- *applications* - таблица, содержащая информацию о заявках студентов;
- *exam_pdd* - таблица, содержащая информацию о всех вопросах для экзаменационного теста;
- *students* - таблица, содержащая информацию о активных студентах;
- *payment* - таблица, содержащая информацию о проведенных транзакциях;
- *cars* - таблица, содержащая информацию о машинах, находящихся на учете автошколы;
- *groups* - таблица, содержащая информацию о группах обучения;
- *students_of_exam* - таблица, содержащая информацию об активных записях на экзамен студентов;
- *teachers* - таблица, содержащая информацию об инструкторах автошколы;
- *lesson* - таблица, содержащая информацию о типах занятий;

- *class* - таблица, информацию о местах проведения занятий;
- *category*- таблица, содержащая информацию о категориях обучения;
- *timetable_tory* - таблица, содержащая расписание теоретических занятий;
- *timetable_practic* - таблица, содержащая расписание практических занятий;
- *program* - таблица, содержащая информацию о программах обучения;
- *gibdd_exam* - таблица, содержащая информацию о датах проведения экзамена, сформированная МРЭО ГИБДД;
- *result_quiz* - таблица, содержащая информацию о прохождении тестирования в боте.

Для наглядного понимания работы БД, на рисунке 1.13 представлена схема физической модели базы данных.

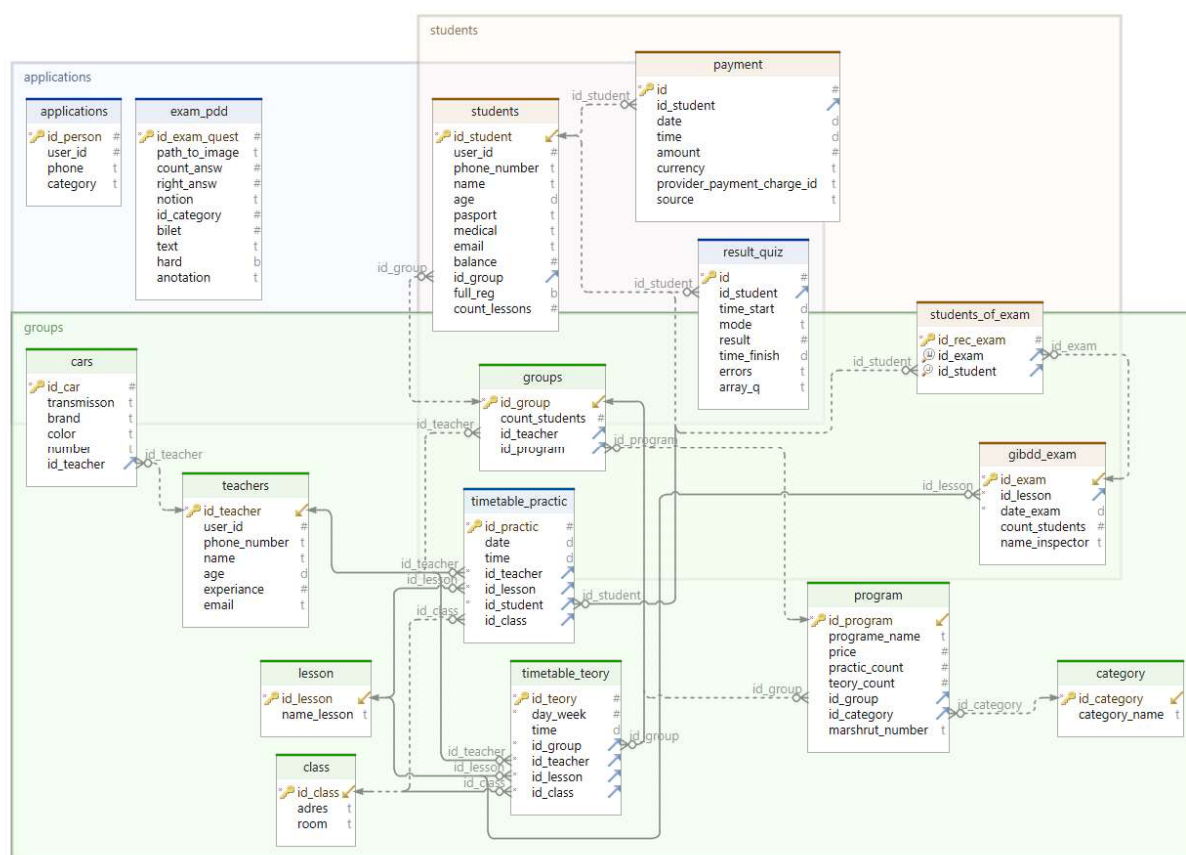


Рисунок 1.13–Физическая модель базы данных

1.8.3. Описание доступных функций

Информационная система предполагает реализацию таких функций, как:

- сбор заявок;
- авторизация;
- управление и просмотр данных;
- управление и просмотр расписания;
- подготовка к экзамену;
- просмотр материалов нужных для сдачи теоретического и практического экзамена;
- управление и учет транзакций;
- информирование по *mail*.

Перед использованием бота, его нужно найти, это можно сделать 3 способами: перейти непосредственно по ссылке, найти в поиске, как показано на рисунке 10, или воспользоваться QR-кодом, показанным на рисунке 1.14.

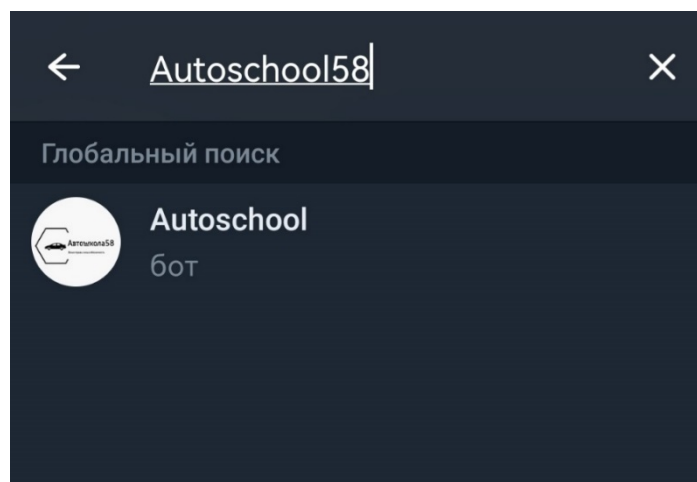


Рисунок 1.14 – Поиск бота в поиске

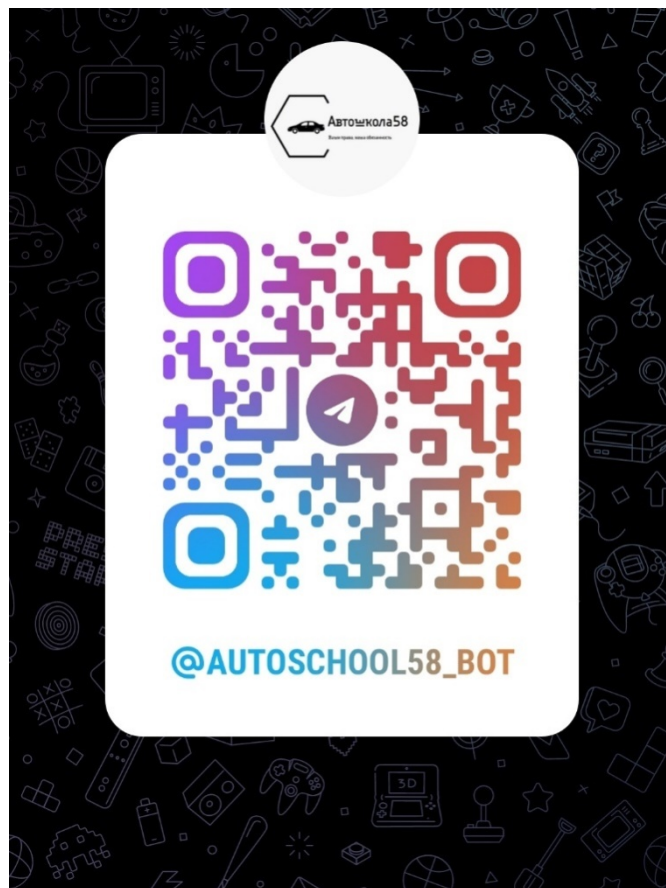


Рисунок 1.15 – Доступ к боту по *QR*

Далее бот дает краткое описание, интуитивно понятное для любого пользователя. Нужно ознакомиться с описанием и нажать на кнопку «Старт» для начала использования бота. Пример начала работы показан на рисунке 1.16.

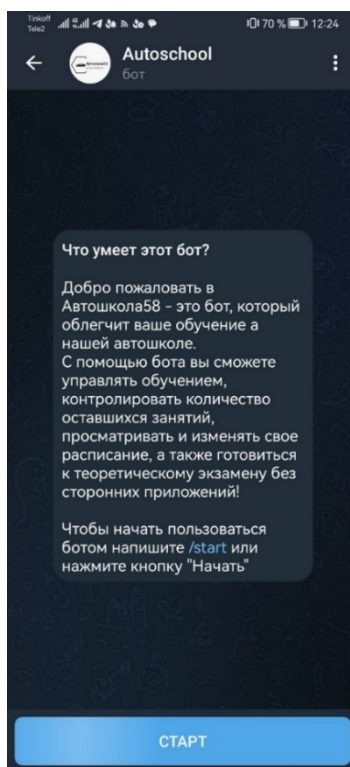


Рисунок 1.16 – Начало работы с ботом

После старта работы, бот просит предоставить номер телефона для авторизации, как показано на рисунке 1.17, тем самым проверить является ли пользователь учеником автошколы или только хочет начать обучение. Также пользователь может ознакомиться с информацией об автошколе, нажав на соответствующую кнопку. Пример работы этой кнопки, будет рассмотрен ниже.

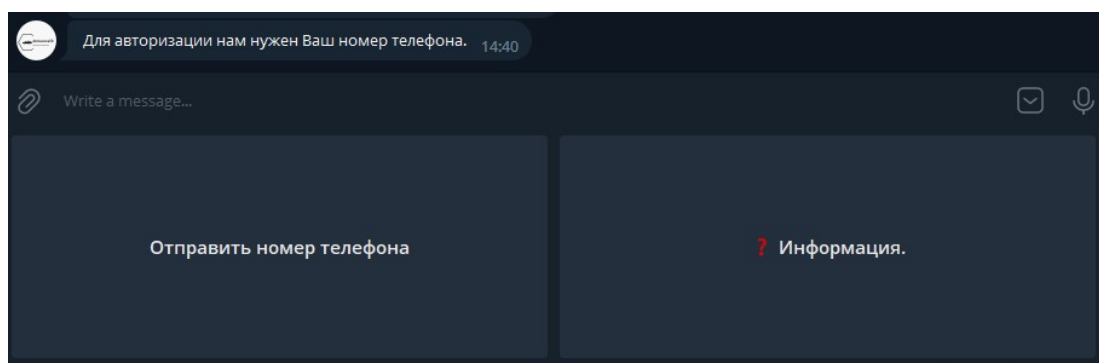


Рисунок 1.17 – Авторизация

Если человека еще нет в списках студентов, то бот информирует его об этом, как на рисунке №, и просит оставить заявку на обучение, для этого

человеку нужно выбрать интересующую категорию, после того как категория выбрана, администратор автошколы, просматривает заявки и после связи с клиентом добавляет его в список студентов, присваивая ему группу. Процесс показан на рисунках 1.18, 1.19.

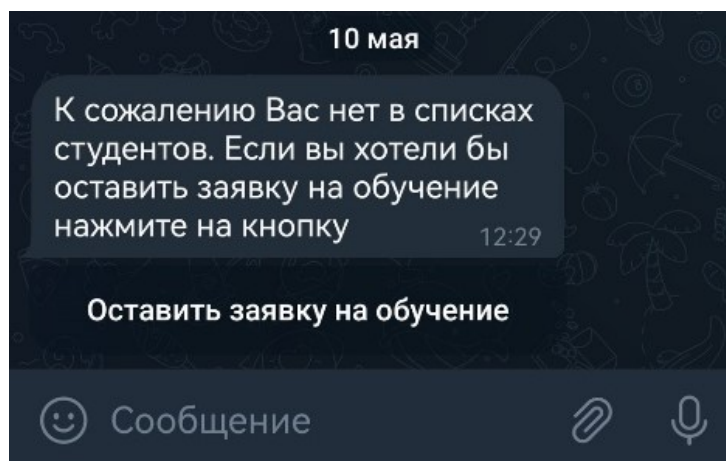


Рисунок 1.18 – Неудачная авторизация.

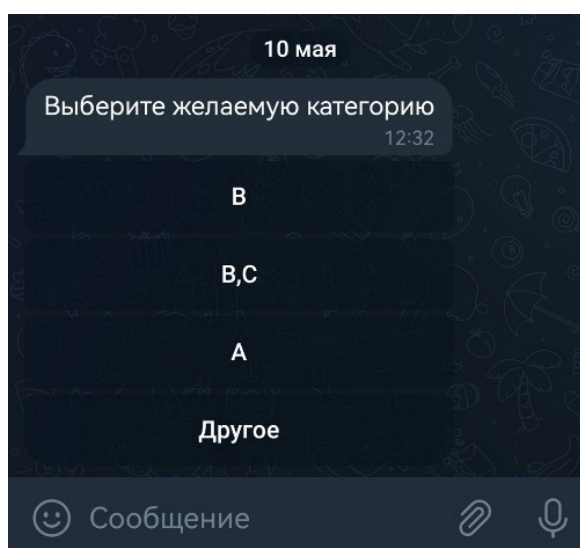


Рисунок 1.19 – Выбор категории.

Когда человек зачислен в список студентов, ему еще раз нужно авторизоваться, повторив алгоритм, описанный выше. После авторизации, бот информирует об успешной процедуре, и студент получает доступ к главному меню. Рисунок 1.20 это демонстрирует.

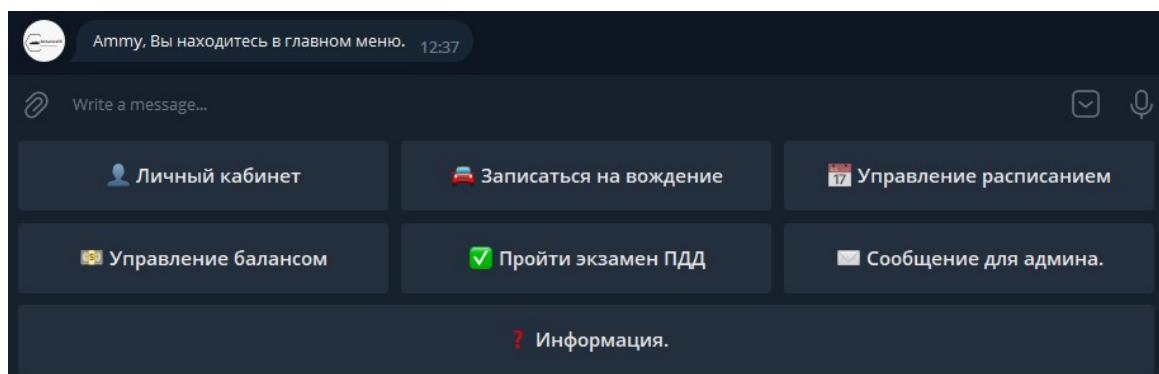


Рисунок 1.20 – Главное меню

Далее рассмотрим все пункты по порядку, для успешного обучения и использования бота необходимо заполнить личные данные, это можно сделать в разделе «Личный кабинет», с него и начнем. Рисунок 1.21 показывает, как выглядит этот раздел.

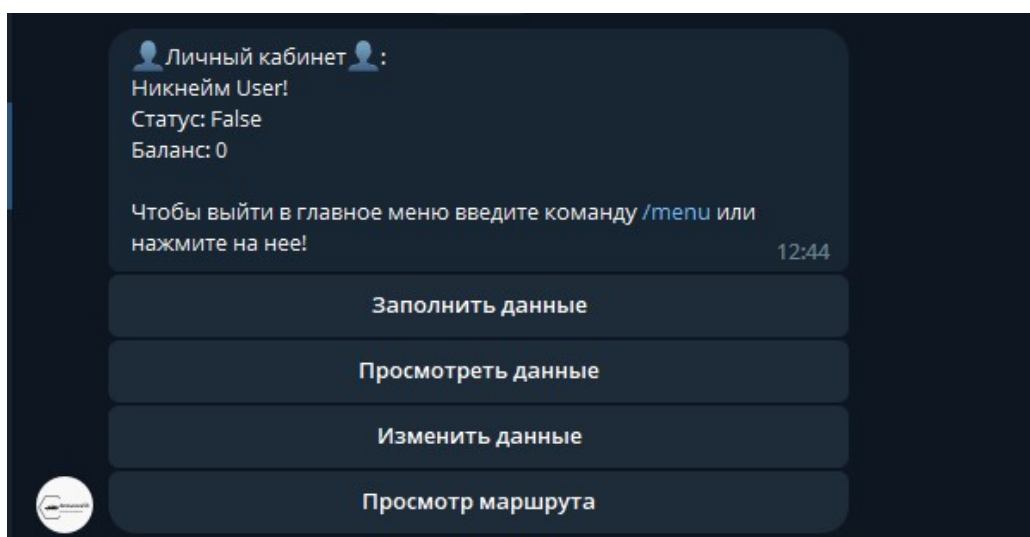


Рисунок 1.21 – Личный кабинет.

Итак, раздел имеет 4 кнопки, как и сказано ранее, для начала студент должен заполнить свои данные нажав на кнопку «Заполнить данные». Бот уточняет какое поле пользователь хочет заполнить. Это продемонстрировано на рисунке 1.22

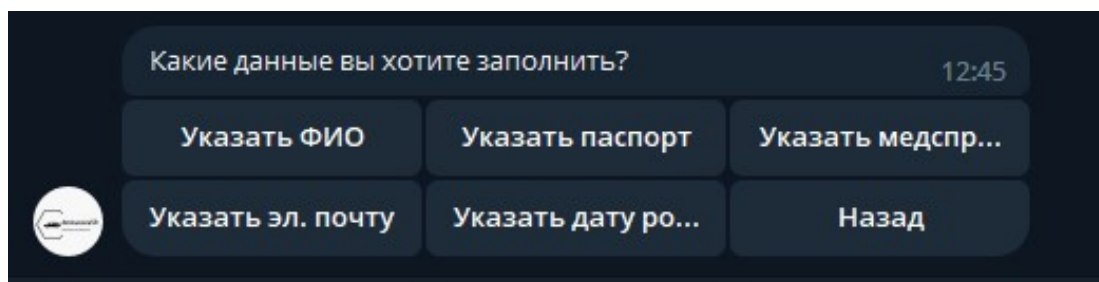


Рисунок 1.22 – Заполнение данных.

Далее студент указывает нужное поле, заполняет его и подтверждает заполнение или повторно указывает значение, если была ошибка. Пример показан на рисунке 1.23. В дальнейшем это поле можно только изменить, из раздела «Заполнить данные» это поле пропадает.

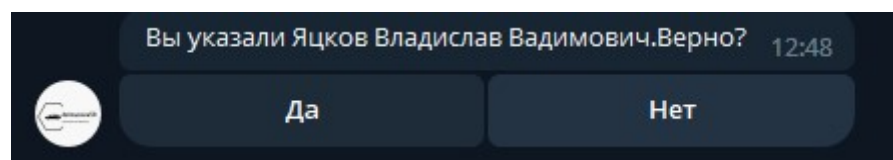


Рисунок 1.23 – Проверка данных

Далее пользователь также заполняет остальные данные. Когда все данные будут заполнены, бот информирует студента об этом. Рассмотрено на рисунке 1.24.

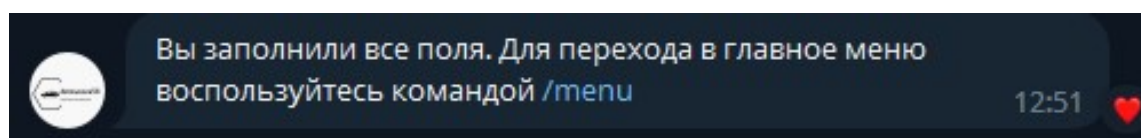


Рисунок 1.24 – Кнопка возврата в меню

Затем, рассмотрим кнопку «Просмотреть данные», по нажатию на нее пользователь просматривает свои данные. Работа кнопки показана на рисунке 1.25.

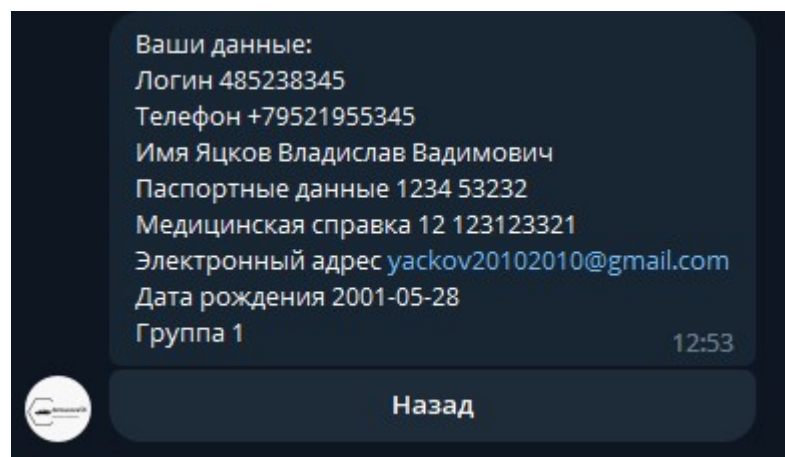


Рисунок 1.25– Просмотр данных

После этого рассмотрим кнопку «Изменить данные». В этом случае пользователь может изменить данные, которые он уже заполнен. Алгоритм работы схожий, что и в первоначальном заполнении данных. Окно показано на рисунке 1.26.

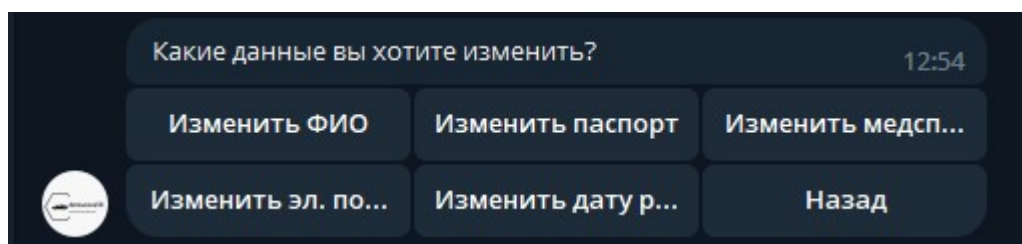


Рисунок 1.26 – Изменение данных

Последней кнопкой является «Просмотреть маршрут», в данном случае студент может ознакомиться с экзаменационным маршрутом, для более успешной подготовки. На данный момент при сдаче экзамена в ГИБДД, маршрут формируется по местонахождения автошколы, поэтому при формировании группы маршрут известен заранее и не изменяется до сдачи экзамена. Это показано на рисунке 1.27.

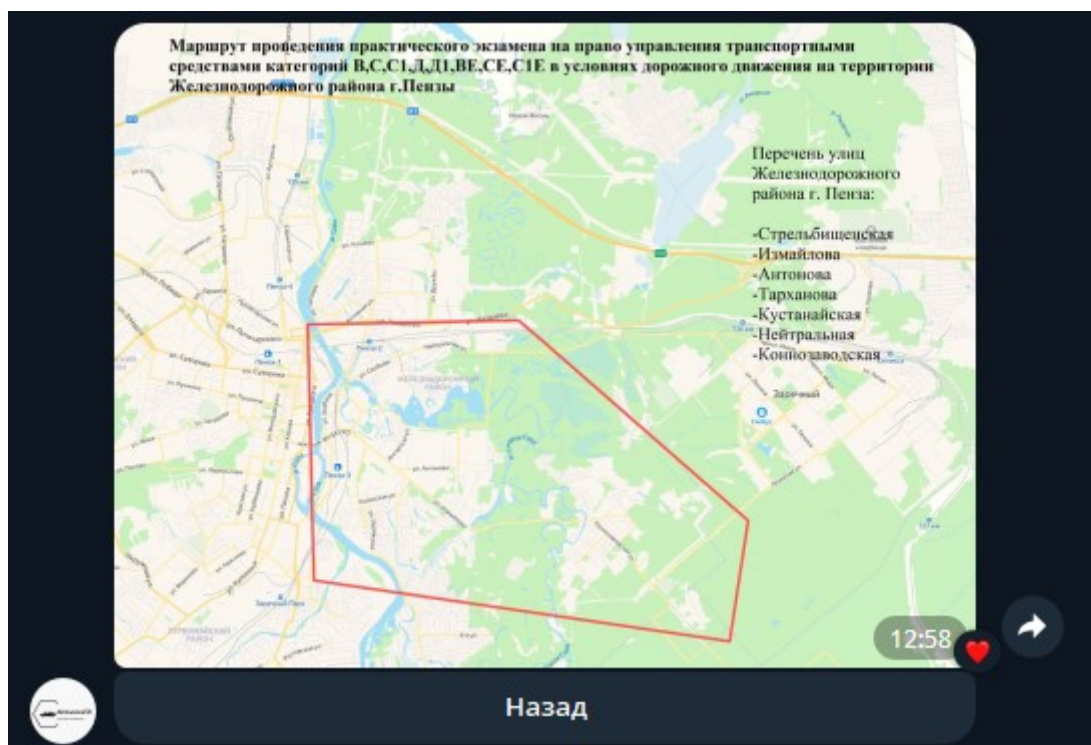


Рисунок 1.27 – Просмотр маршрута

Теперь рассмотрим следующий раздел, это «Управление балансом». Перейдя в этот раздел, мы видим подменю, указанное на рисунке 1.28.

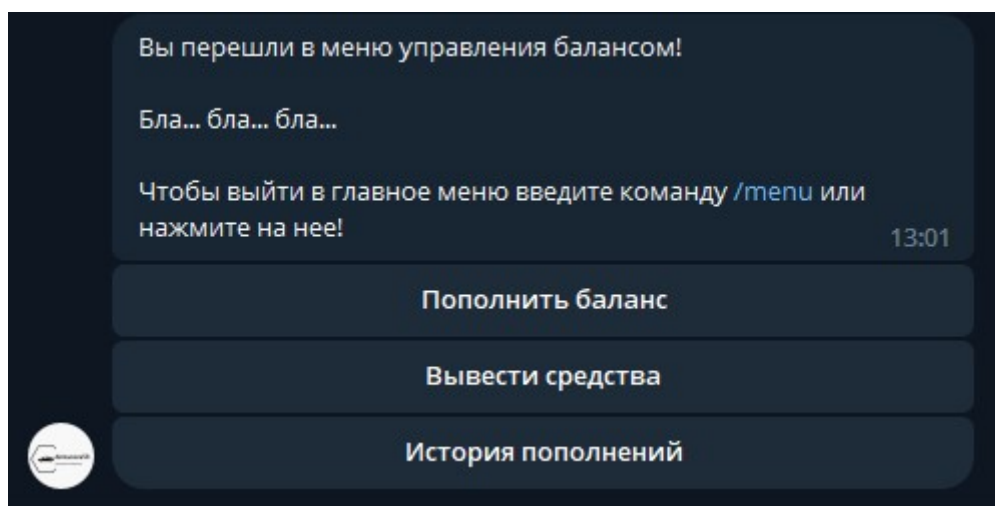


Рисунок 1.28 – Управление балансом

Для начала, студент может пополнить баланс своего счета, нажав на кнопку «Пополнить баланс». Далее ему нужно выбрать сумму для пополнения, работа продемонстрирована на рисунке 1.29.

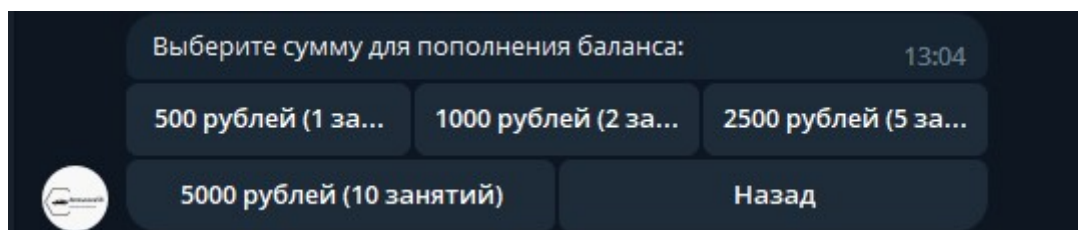


Рисунок 1.29 – Выбор суммы пополнения

Далее, студент выбирает сумму и нажимает оплатить, как показано на рисунке 1.30.

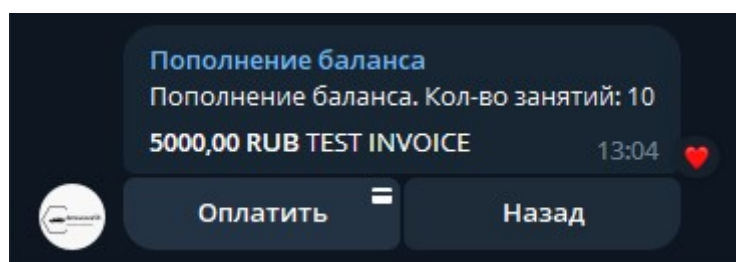


Рисунок 1.30 – Подтверждение пополнения

После этого пользователь, указывает метод оплаты, и вводит данные своей карты. Как показано на рисунке 1.31. После чего вводит код из смс, и оплата поступает на счет ученика. Это показано на рисунке 1.32

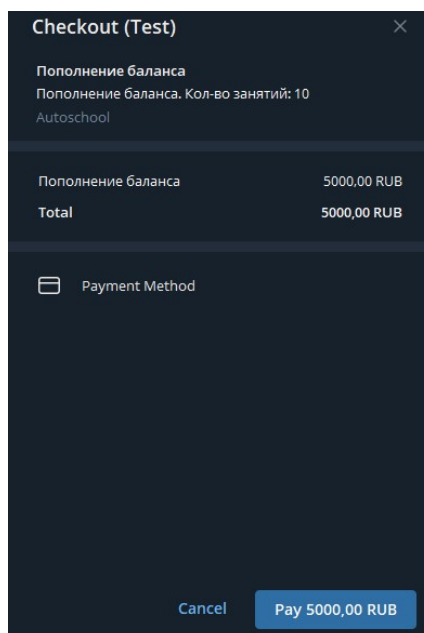


Рисунок 1.31 – Процесс оплаты

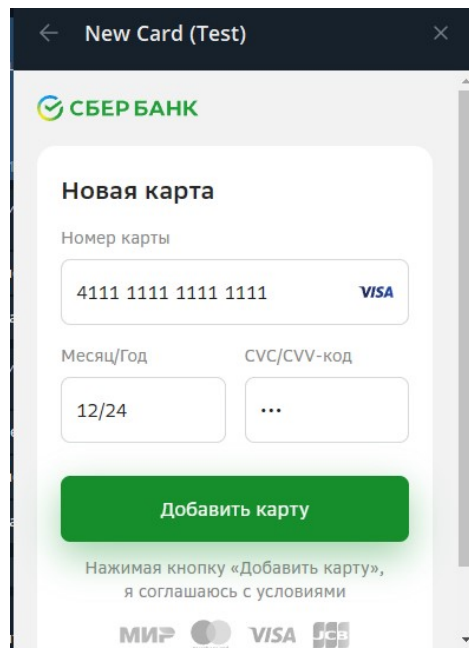


Рисунок 1.32– Процесс оплаты

Если транзакция прошла успешно, то пользователь получает информационное сообщение. Оно продемонстрировано на рисунке 1.33.

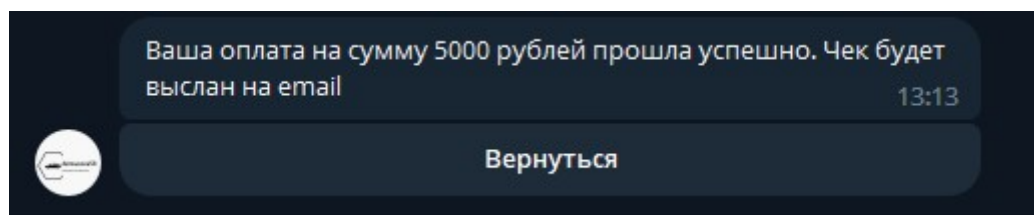


Рисунок 1.33 – Уведомление об успешной оплате

Далее пользователь, может просмотреть чек об оплате, который был выслан на почту, указанную при регистрации и проверить свой баланс. Пример чека показан на рисунке 1.34, а также на рисунке 1.35 показана история пополнений.

Электронный чек № 56

Информация о покупателе:
Яцков Владислав Вадимович
485238345
yackov20102010@gmail.com
Тел: +79521955345

Продавец:
Автошкола58
Россия, г. Пенза, Пушкина 37
000012345
Тел: +7(8412)357546

Метод оплаты
Карта **** 4 цифры карты
[yackov20102010@gmail.com]

Дата покупки
2023-05-10 13:13:25.119460

Состав покупки

#	Описание	Кол-во	Цена	Сумма
1	Пополнение через telegram Пополнение баланса. Кол-во занятий: 10	1	Р5000.0	Р5000.0
			Вид транзакции	Пополнение
			Баланс	Р10500.0

Желаем успехов!
Ваша "Автошкола58"

Рисунок 1.34 – Чек высланный на *mail*

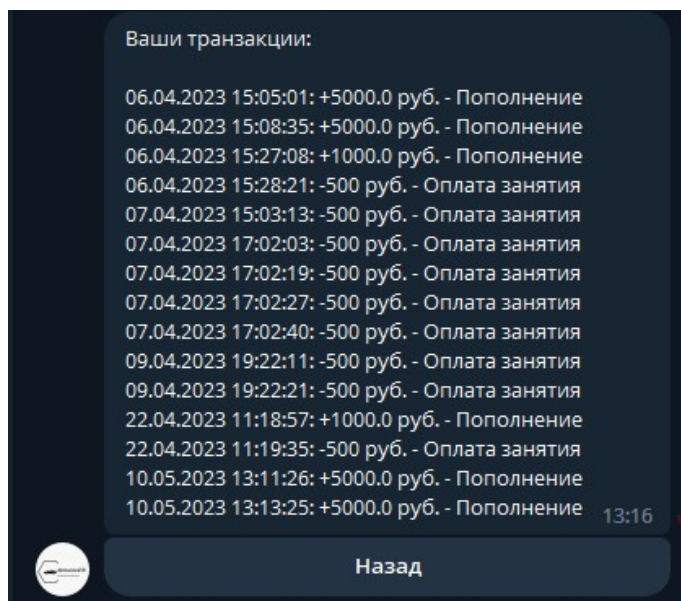


Рисунок 1.35 – История транзакций

Также в этом разделе, пользователь может запросить вывод средств нажав на кнопку «Вывести средства». Далее бот указывает, что вывод доступен только для полной суммы баланса, а также информирует о балансе. Рисунок 1.36 это показывает.

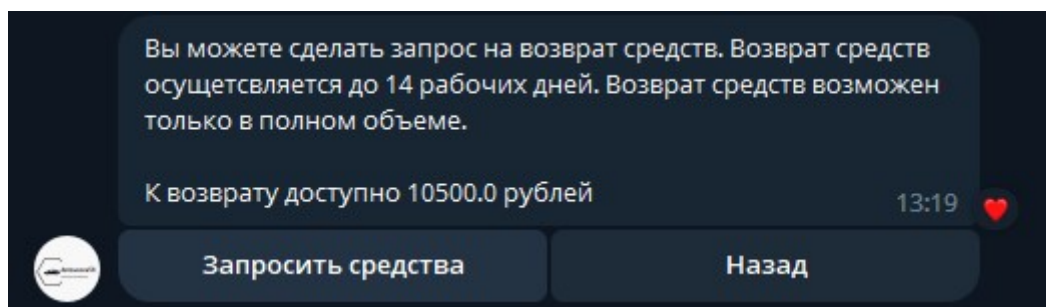


Рисунок 1.36 – Вывод средств

Если заявка на вывод средств прошла успешно, то бот отправляет информационное -всплывающее сообщение. Это видно на рисунке 1.37.

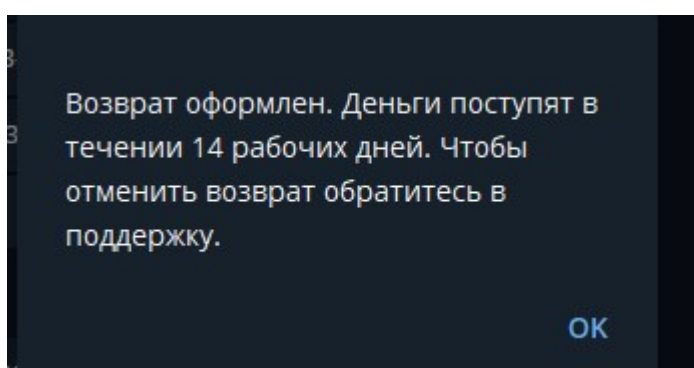


Рисунок 1.37 – Уведомление об успешном выводе средств

Далее нужно вернуться в главное меню и перейти в раздел «Запись на занятия». В этом разделе мы можем записаться на 3 варианта мероприятия, это занятие, практический экзамен или теоретический экзамен, также бот дает информацию о записи. Это видно на рисунке 1.38.

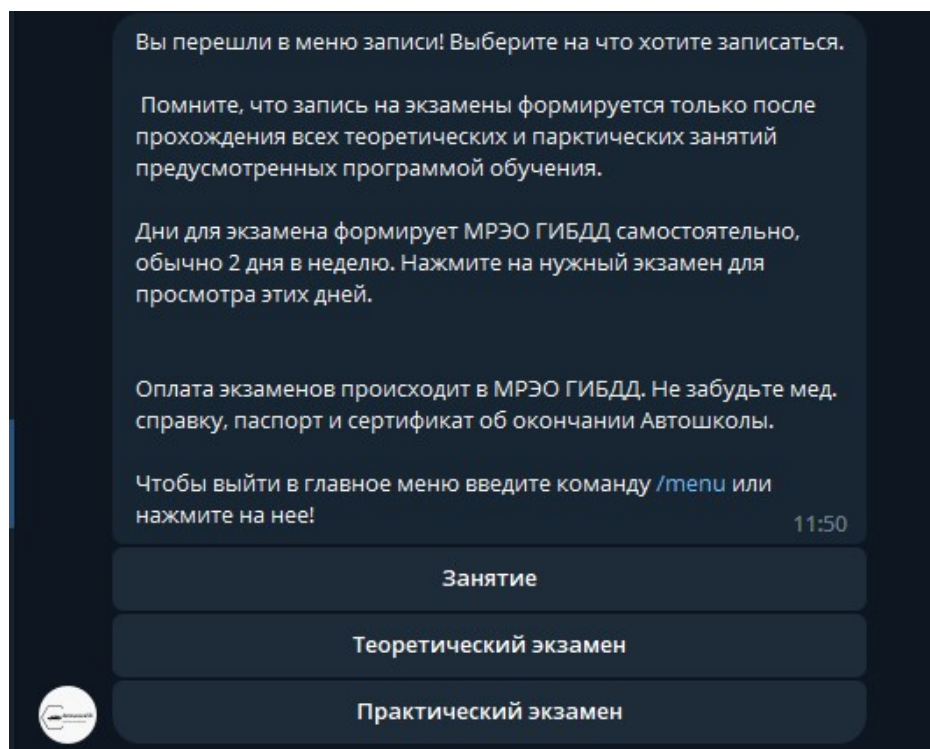


Рисунок 1.38 – Меню записи

Рассмотрим подробно запись на занятие, для этого нажмем на кнопку «Занятие» и перейдем к выбору дня, пример показан на рисунке 1.39.

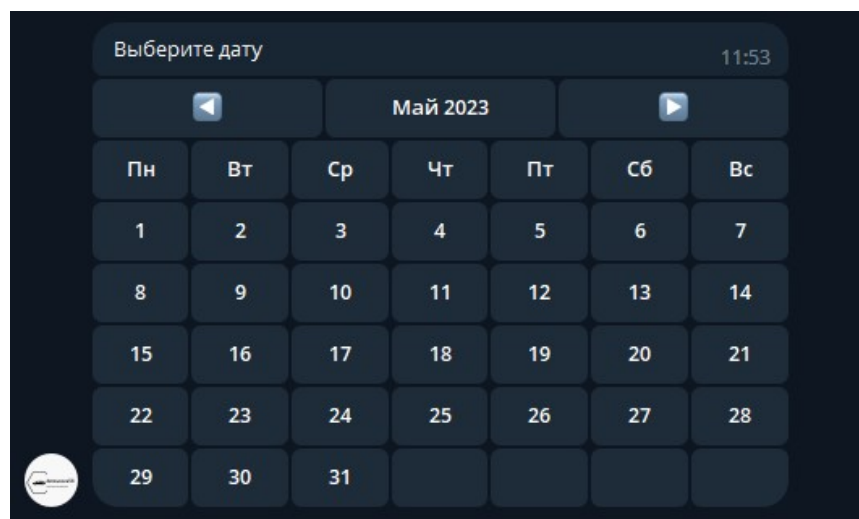


Рисунок 1.39 – Календарь

Далее рисунки 1.40, 1.41, 1.42 демонстрируют, что студент, выбрав интересующий день, может записаться только на время, которое свободно, не является временем, которое прошло и не выходным днем.

Выберите время 11:56

08:00 (уже прошло)	09:00 (уже прошло)
10:00 (уже прошло)	11:00 (уже прошло)
13:00 (уже прошло)	14:00 (уже прошло)
15:00 (уже прошло)	16:00 (уже прошло)
17:00 (уже прошло)	18:00 (уже прошло)

Назад

Рисунок 1.40 – Запись на прошедшую дату

Выберите время 11:56

08:00 (выходной)	09:00 (выходной)
10:00 (выходной)	11:00 (выходной)
13:00 (выходной)	14:00 (выходной)
15:00 (выходной)	16:00 (выходной)
17:00 (выходной)	18:00 (выходной)

Назад

Рисунок 1.41 – Запись на выходную дату

Выберите время 11:58

08:00	09:00
10:00	11:00
13:00 (занято)	14:00
15:00	16:00
17:00	18:00

Назад

Рисунок 1.42 – Запись на корректную дату

Когда студент выбрал подходящее время, и убедился, что оно свободно, он нажимает на него и бот уточняет точную дату и время занятия, если пользователя устраивает, то с его баланса списывается необходимое количество средств, а чек отправляется на *email*. Это показано на рисунках 1.43, 1.44.

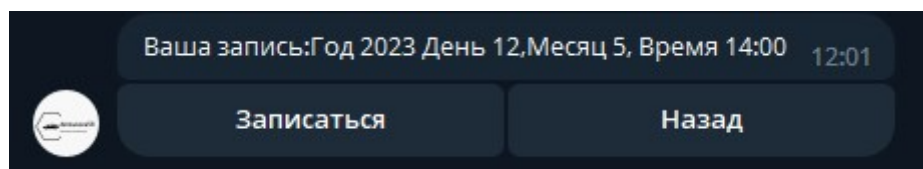


Рисунок 1.43 – Подтверждение записи

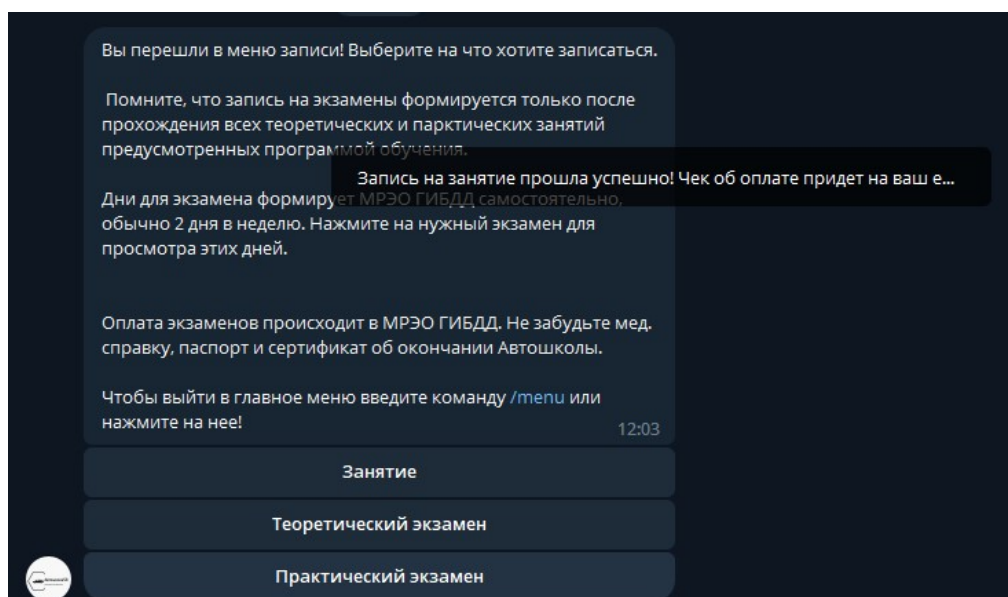


Рисунок 1.44 – Уведомление об успешной записи

Если же у пользователя не хватает средств на балансе, то бот информирует его об этом, как на рисунке 1.45.

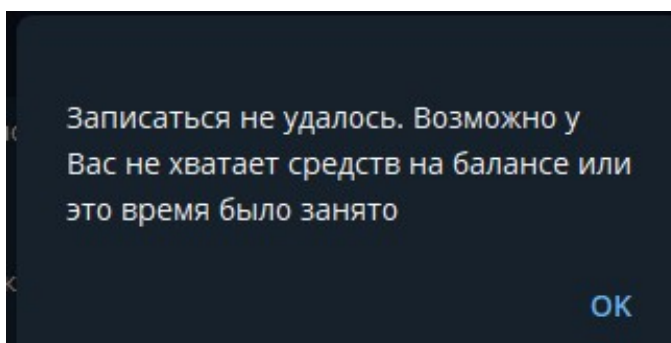


Рисунок 1.45 – Ошибка записи

Следующим рассмотрим запись на экзамен, стоит отметить, что даты на экзамены формируются ГИБДД и отправляются в автошколу, после чего администратор добавляет соответствующие даты в базу данных. Количество мест ограничено, поэтому было решено ограничить одновременную запись только на один экзамен, для одного пользователя. Как выглядит пример предложенных дат, показано на рисунке 1.46.

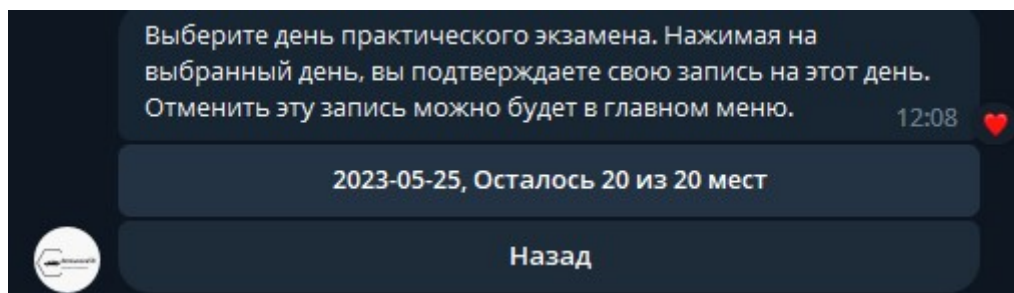


Рисунок 1.46 – Запись на экзамен

Также стоит сказать, что для записи, студенту требуется откатать положенное по закону часов вождения. Рассмотрим ситуацию, когда у студента не хватает часов вождения или он уже был записан на экзамен, рассмотрим это на рисунке 1.47.

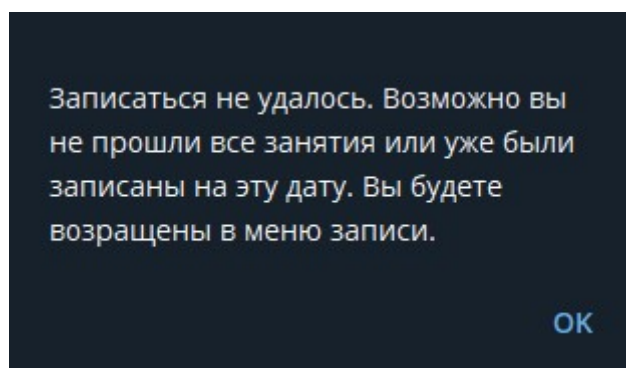


Рисунок 1.47 – Ошибка записи

Теперь рассмотрим ситуацию, когда пользователь не был записан на экзамен и все необходимые часы вождения откатаны. Это показано на рисунке 1.48.

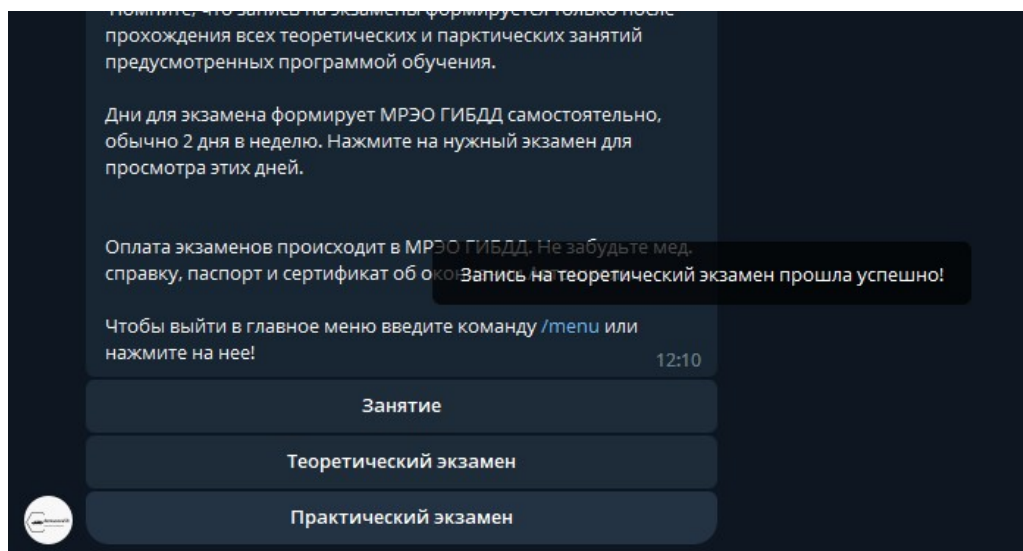


Рисунок 1.48 – Уведомление об успешной записи

Следующий раздел, который рассмотрим, это управление расписанием. В этом разделе пользователь может корректировать и просматривать актуальные занятия. Меню выглядит так, как показано на рисунке 1.49.

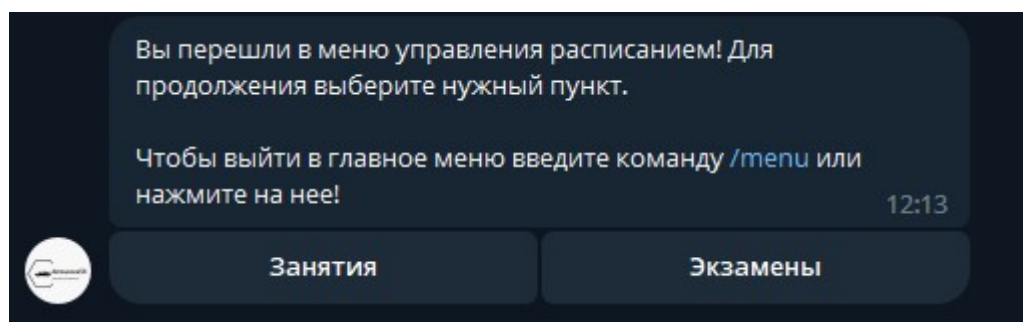


Рисунок 49 – Управление расписанием

Для начала рассмотрим управление занятиями. После нажатия на кнопку «Занятия», студенту доступно к просмотру расписание его активных занятий. Он может просмотреть актуальную информацию по занятию, а также отменить занятие, если день этого занятия не наступил. Как выглядит расписание показано на рисунке 1.50.

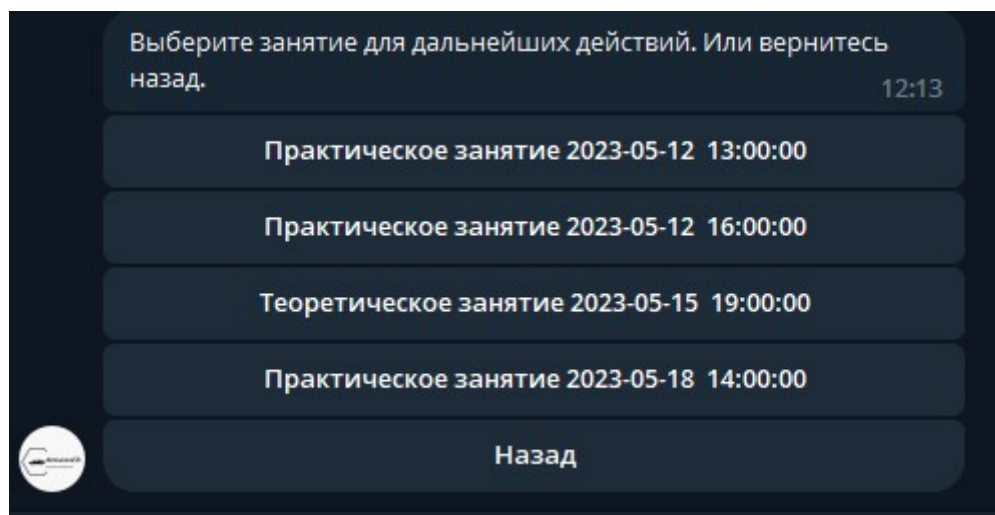


Рисунок 1.50 – Просмотр расписания

Мы видим, что занятия сортируются по возрастанию по дате, для просмотра подробной информации или отмене занятия, нужно нажать на соответствующую кнопку. На рисунке 1.51 рассмотрен пример занятия. Стоит отметить, что теоретическое занятие — это групповое занятие, которое пользователь не может отменить.

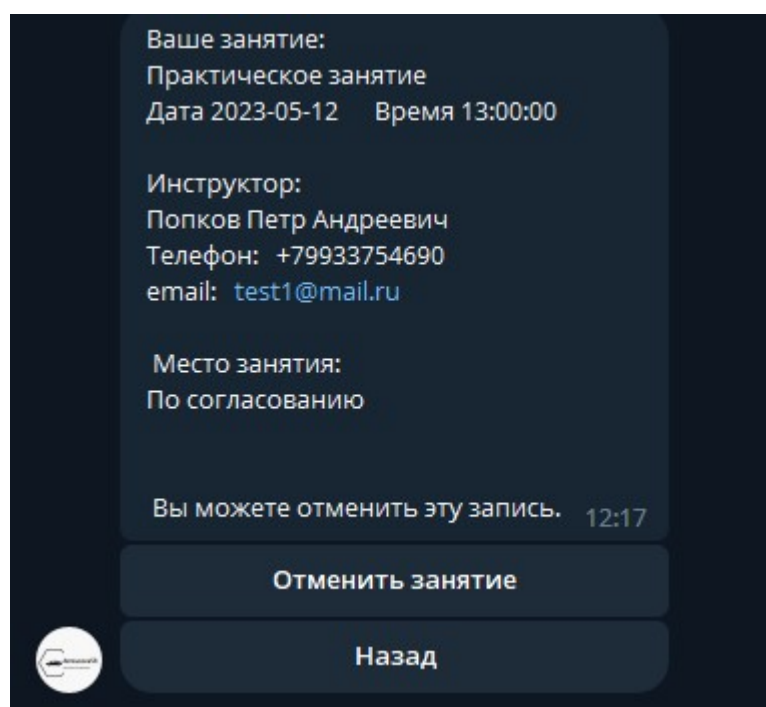


Рисунок 1.51 – Просмотр практического занятия

Далее рассмотрим, меню управления экзаменами. Оно работает аналогично занятиям, это видно на рисунках 1.52,1.53.

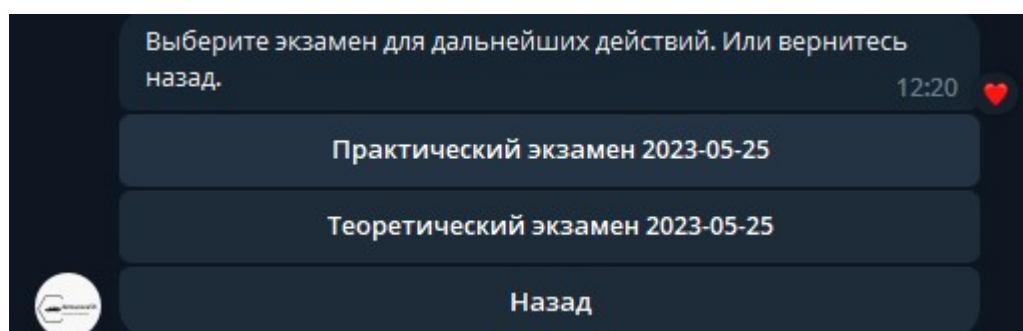


Рисунок 1.52 – Управление экзаменами

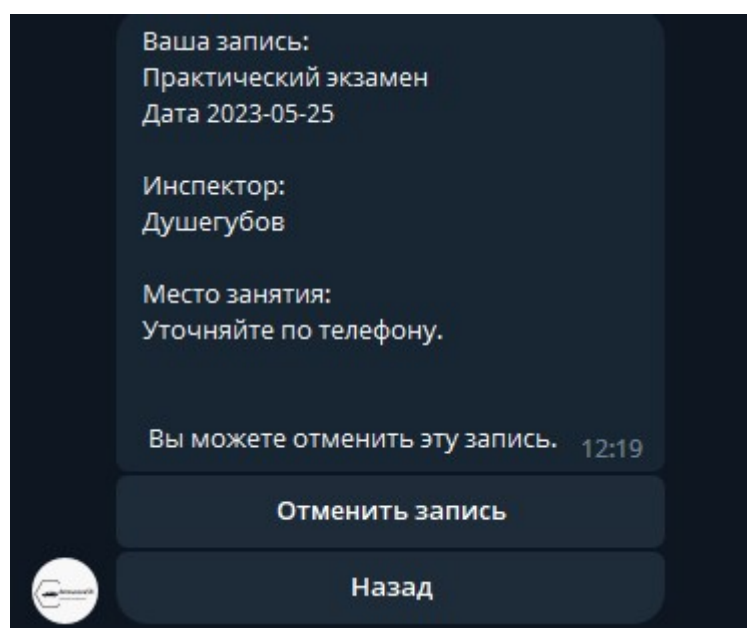


Рисунок 1.53 – Подробный просмотр экзамена

Следующий раздел, это раздел теоретического тестирования, необходимого для подготовки к реальному теоретическому экзамену. Данный раздел делится на необходимые подразделы для подготовки. А режим «Экзамен», сделан на основе реального алгоритма создания экзаменационного билета. Меню показано на рисунке 1.54.

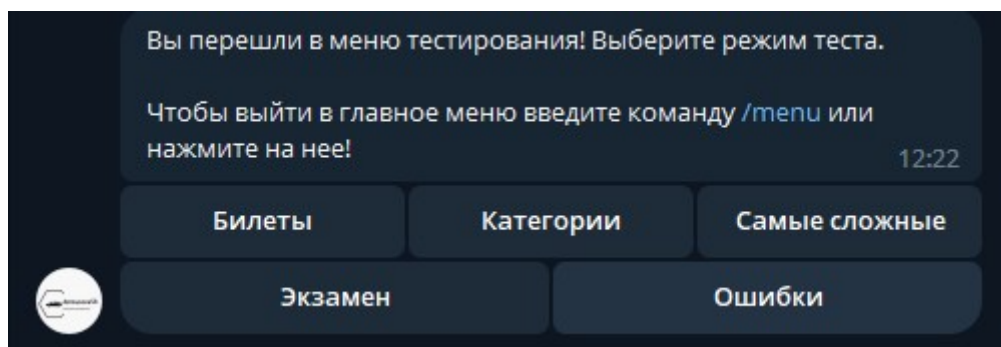


Рисунок 1.54 – Меню тестирования

Первый режим, это «Билеты», в данном режиме студент сам может прорешать необходимый ему билет, билеты для подготовки формируются ГИБДД и имеют единый вид. Процедура выбора билета, и ответов на вопросы показана на рисунках 1.55, 1.56. Стоит заметить, что в данном режиме присутствуют подсказки-аннотации. Которые можно увидеть только если нажать на них, как показано на рисунке 1.57.

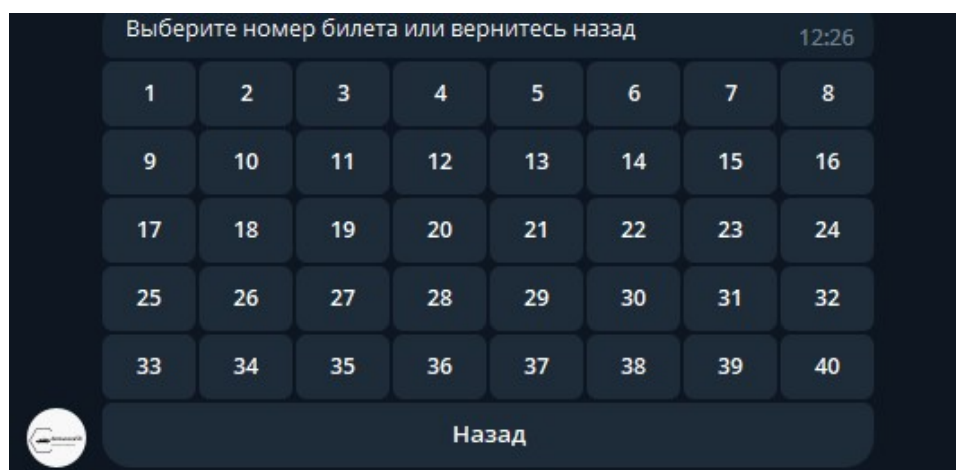


Рисунок 1.55 – Выбор билета

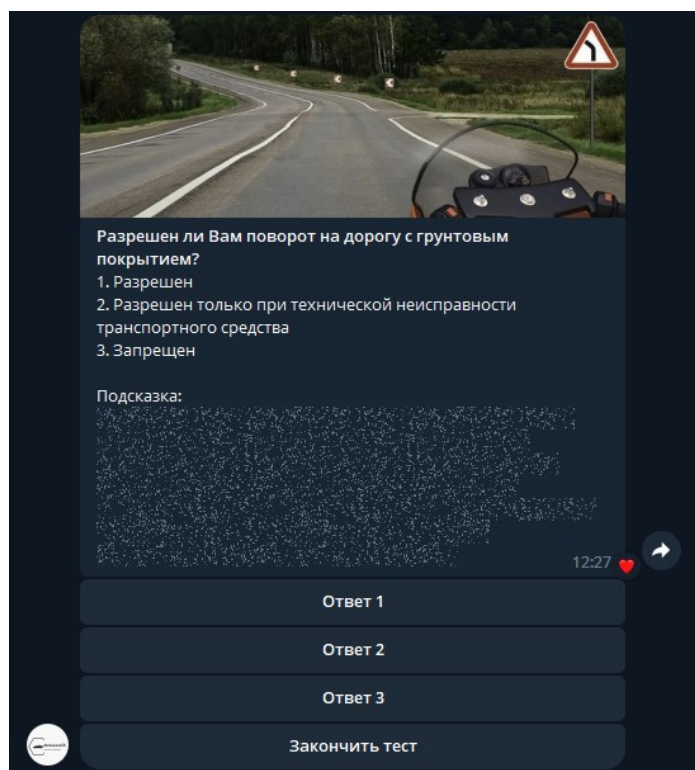


Рисунок 1.56 – Просмотр вопроса билета

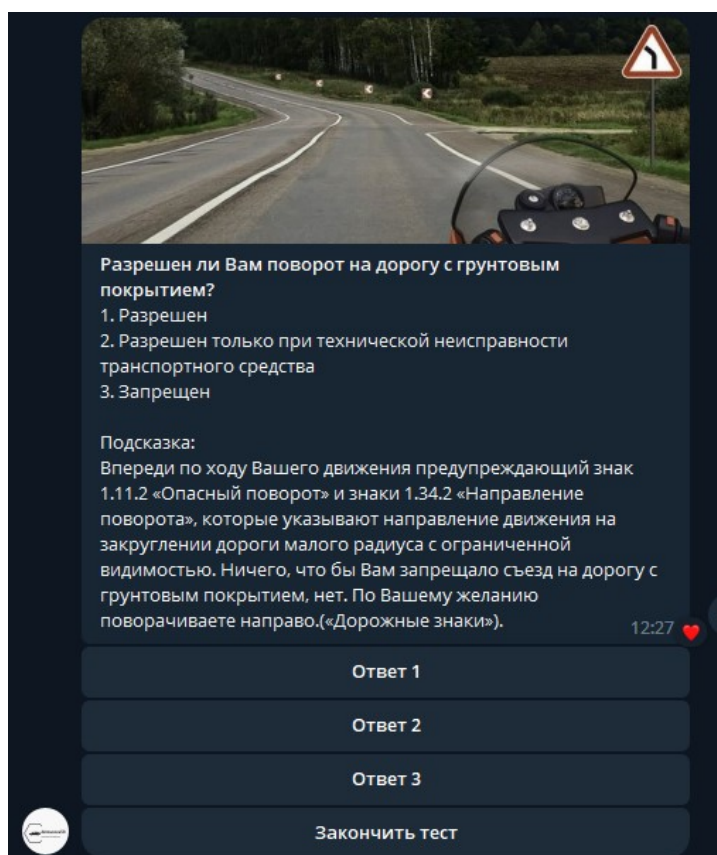


Рисунок 1.57 – Просмотр подсказки

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.39.001 ПЗ

Лист

52

Пользователь может в любой момент завершить тест или прорешать его до конца, в конце бот считает количество верных и не верных ответов. И показывает его пользователю, это видно на рисунке 1.58.

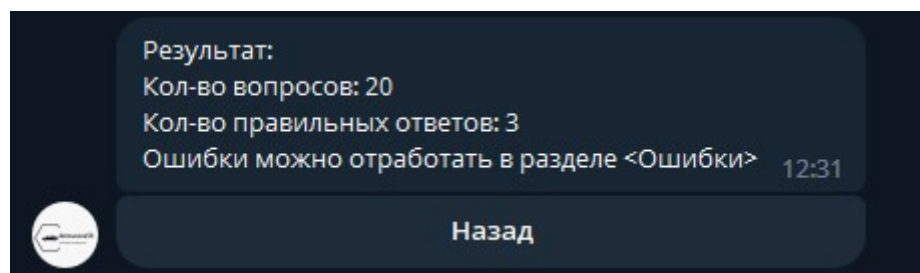


Рисунок 1.58 – Результат тестирования

Также пользователь может ознакомиться со своими ошибками и прорешать их в разделе «Ошибки», Данный раздел показан на рисунке 1.59. Заметим, что теперь в этом разделе присутствует кнопка показать ответ.

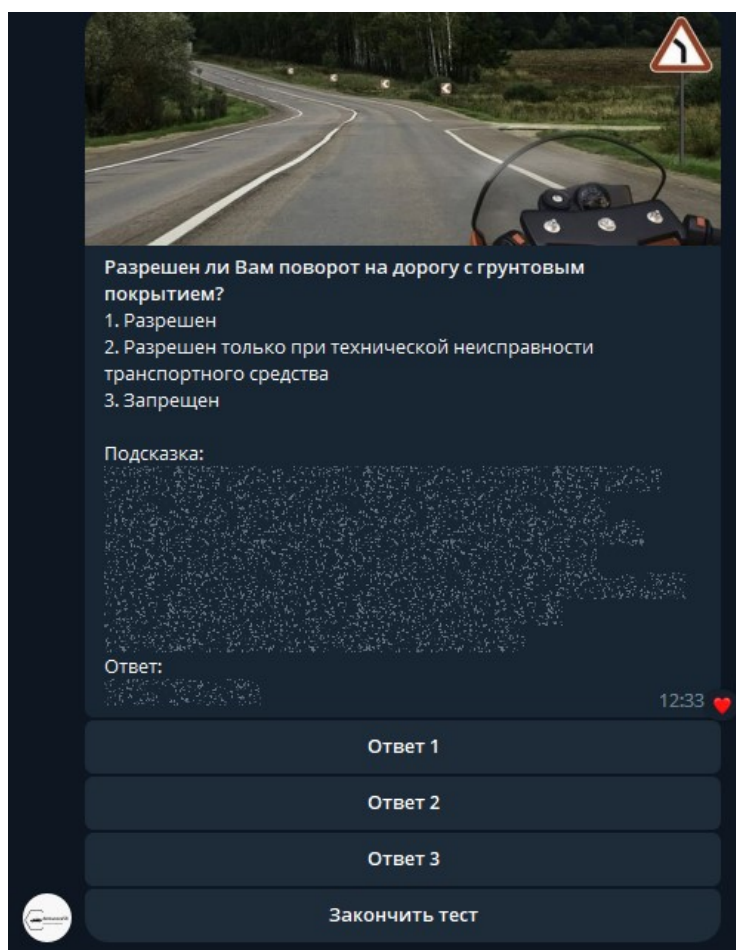


Рисунок 1.59 – Просмотр вопроса в режиме «Ошибки»

Следующий раздел, это «Категории», в этом разделе студент может решать вопросы, разделяя их по категориям, меню сделано удобно, с возможностью листать список категорий и выбирать необходимую. Пример показан на рисунках 1.60,1.61.

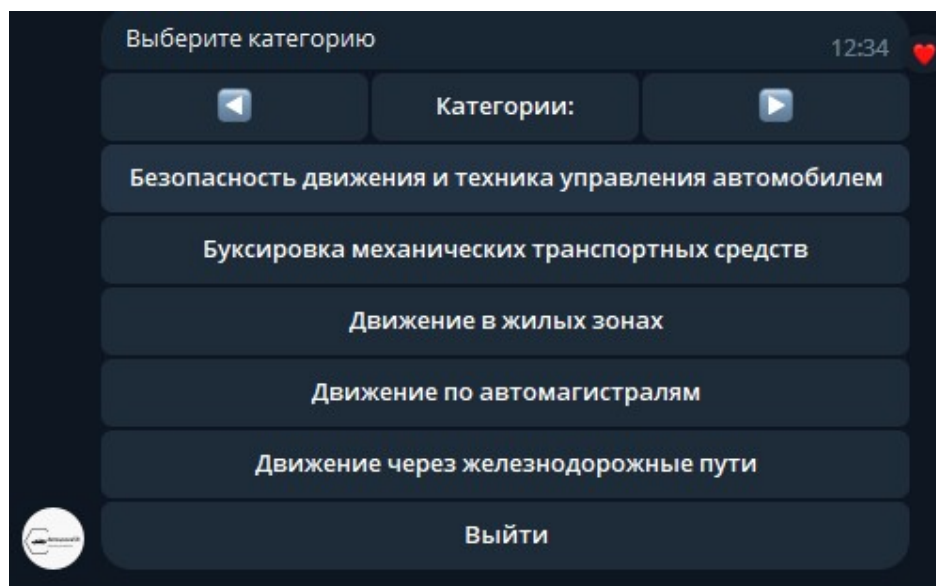


Рисунок 1.60 – Просмотр категорий

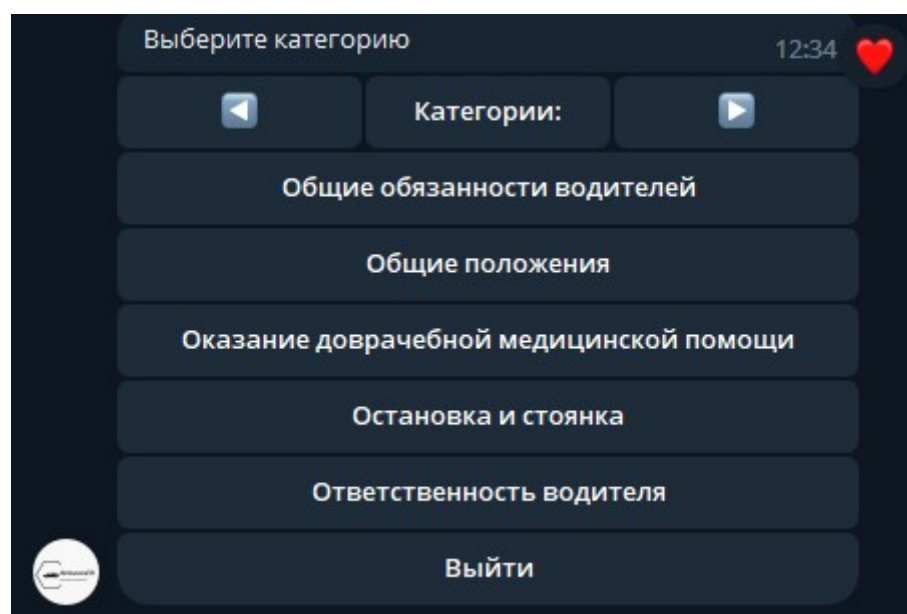


Рисунок 1.61 – Просмотр категорий

Пример теста и вопросов аналогичен прошлому, также есть раздел «Самые сложные», этот раздел включает в себя 40 самых сложных вопросов, это

вопросы, в которых студенты делают чаще всего ошибки по статистике ГИБДД. Этот раздел аналогичен предыдущим.

Также рассмотрим раздел «Экзамен». В этом разделе, как было сказано ранее, вопросы формируются на основе реального алгоритма создания экзаменационного билета. По всем нормам и правилам. В этом режиме отсутствуют подсказки-аннотации, чтобы как можно ближе приблизиться к условиям реального экзамена. Ошибки после решения этого раздела, также можно просмотреть и решить в разделе «Ошибки». Пример вопросов можно увидеть на рисунке 1.62.

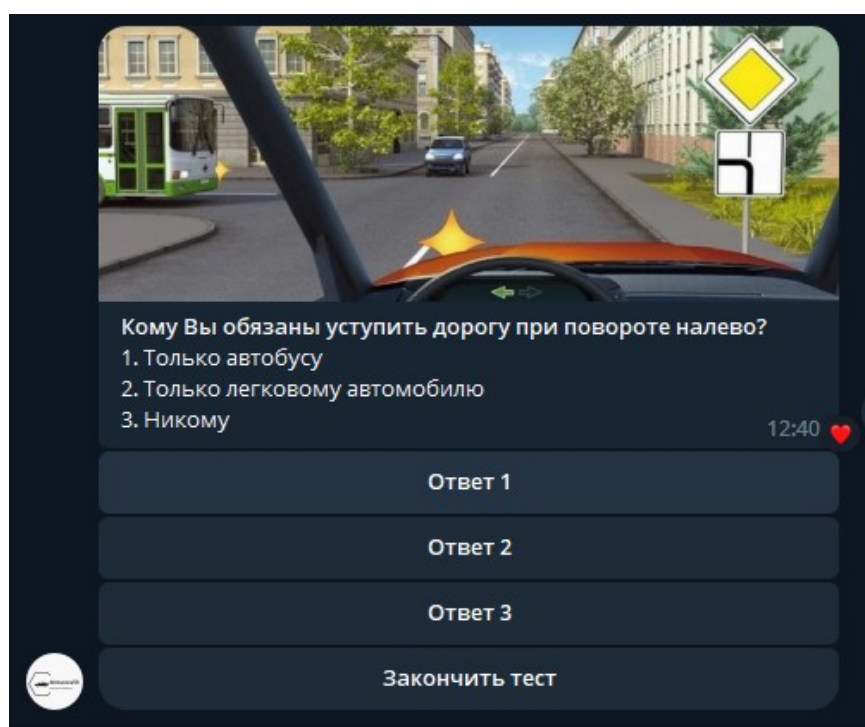


Рисунок 1.62 – Просмотр вопроса из режима «Экзамен»

Наконец рассмотрим последний небольшой раздел, доступный студентам, это «Информация об автошколе». Эта кнопка доступна как авторизованным пользователям, так и не авторизованным, как это было показано выше. Нажав на эту кнопку, пользователь может ознакомиться с контактной и общей информацией об организации. Пример работы показан на рисунке 1.63.

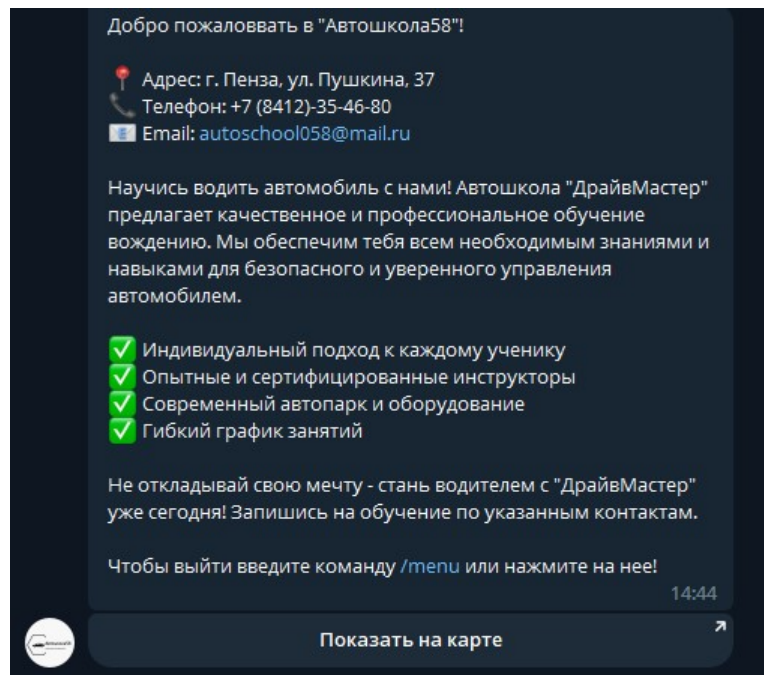


Рисунок 1.63 – Информация об автошколе

Также пользователю доступна кнопка «Показать на карте», которая в свою очередь открывает «Яндекс карты» и показывает местоположение на карте, при необходимости, пользователь может проложить маршрут к данному адресу. Работа кнопки показана на рисунке 1.64.

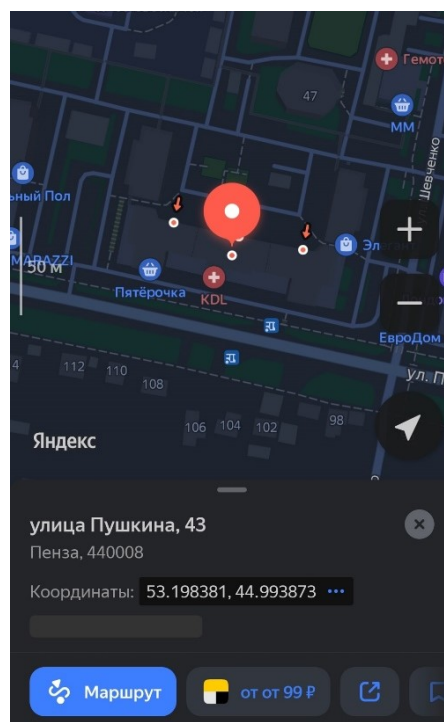


Рисунок 1.64 – Автошкола на карте

Теперь рассмотрим разделы, скрытые от студентов, предназначенные для инструктора и администратора автошколы. Чтобы попасть в раздел инструктора, в боте нужно прописать команду */teacher*. Уникальные логины инструкторов хранятся в базе данных, и после вызова команды, бот проверяет права доступа к этому разделу. Если у пользователя есть права доступа инструктора, то перед ним открывается меню, показанное на рисунке 1.65.

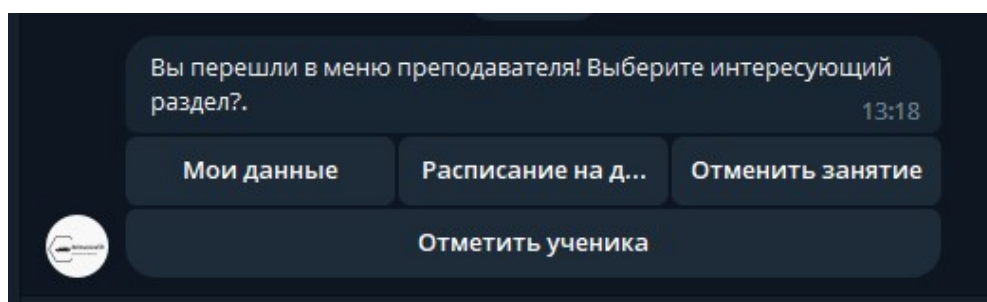


Рисунок 1.65 – Меню инструктора

Преподаватель может просмотреть свои данные, нажав на соответствующую кнопку. Пример показан на рисунке 1.66.

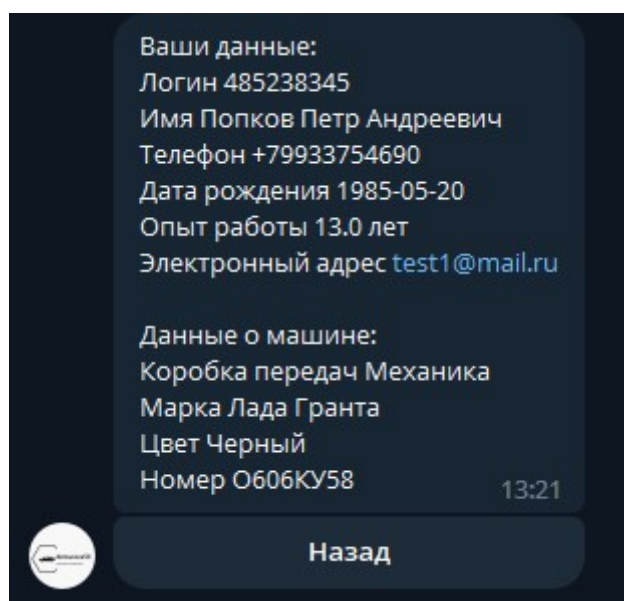


Рисунок 1.66 – Просмотр данных

Также инструктор может просмотреть свое расписание на ближайшие 4 дня, если есть на них запись. Пример просмотра расписания показан на рисунках 1.67, 1.68.

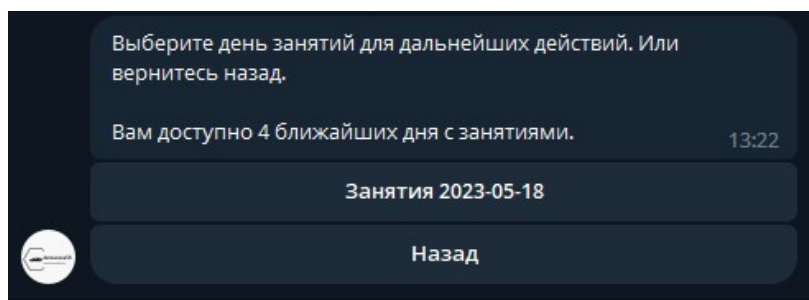


Рисунок 1.67 – Просмотр расписания

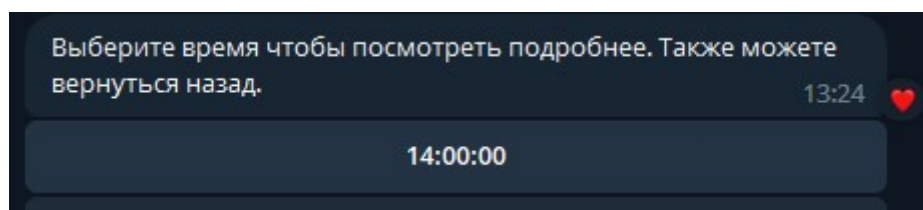


Рисунок 1.68 – Просмотр расписания на день

Далее инструктор может просмотреть подробное описание занятия нажав на время, это показано на рисунке 1.69.

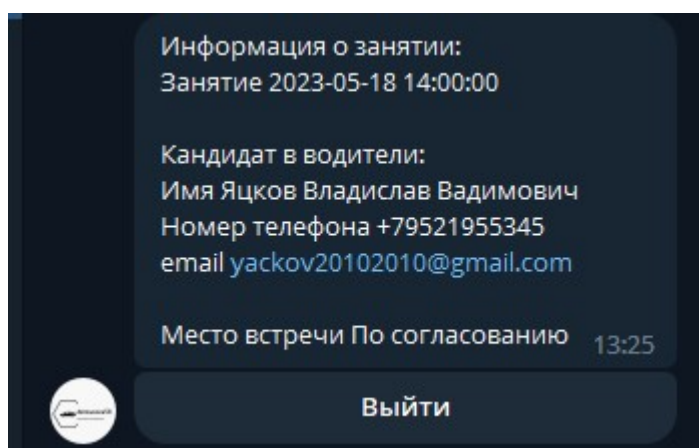


Рисунок 1.69 – Подробная информация о занятии

Также инструктор может отменить занятие, нажав на соответствующую кнопку. К отмене доступны 7 ближайших дней с активной записью. Учитель выбирает день, и время отмены занятия. Преподаватель может отменить все занятия на выбранный день. Процесс отмены показан на рисунках 1.70.

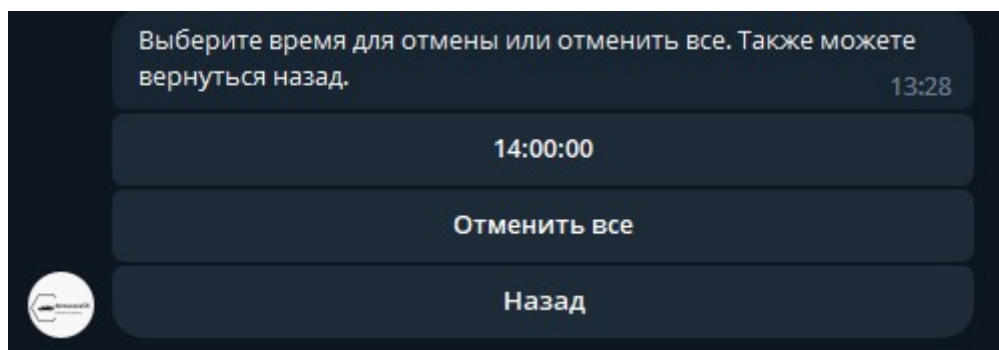


Рисунок 1.70 – Отмена занятия

После отмены занятия, соответствующему ученику, придет на почту чек с оповещением отмены занятия, а также вернутся средства на баланс. Следующим, рассмотрим процесс отметки ученика, это нужно для того, чтобы ученику, который откатал занятие, начислялось время, необходимое для того, чтобы получить право быть допущенным на реальный экзамен. По нажатию на соответствующую кнопку, инструктор должен выбрать день, за который он должен отметить ученика, после этого выбрать время для отметки или отметить всех. Процесс отметки показан на рисунках 1.71, 1.72, 1.73.

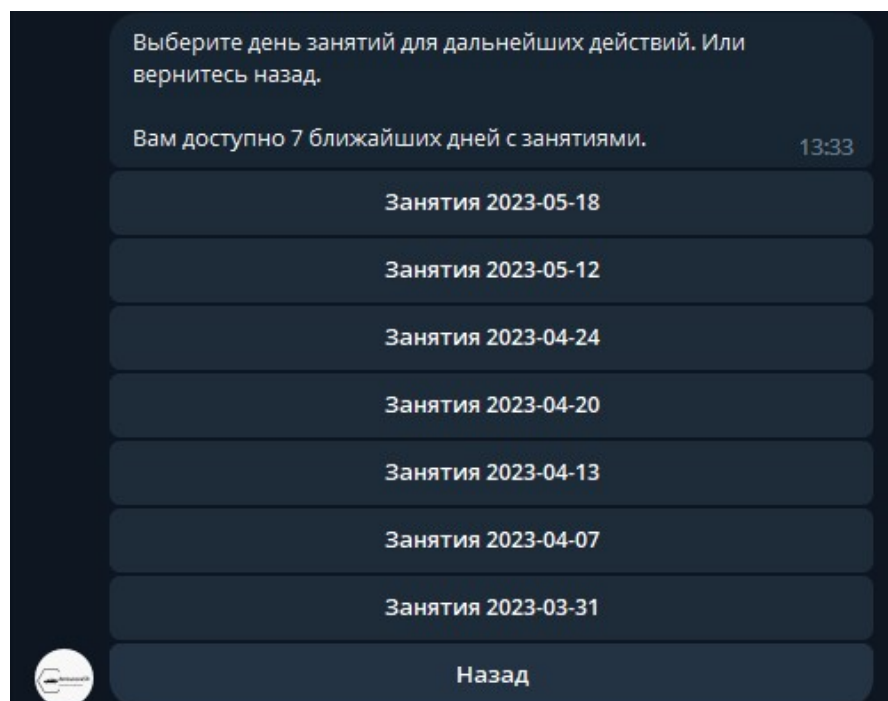


Рисунок 1.71 – Процесс отметки ученика

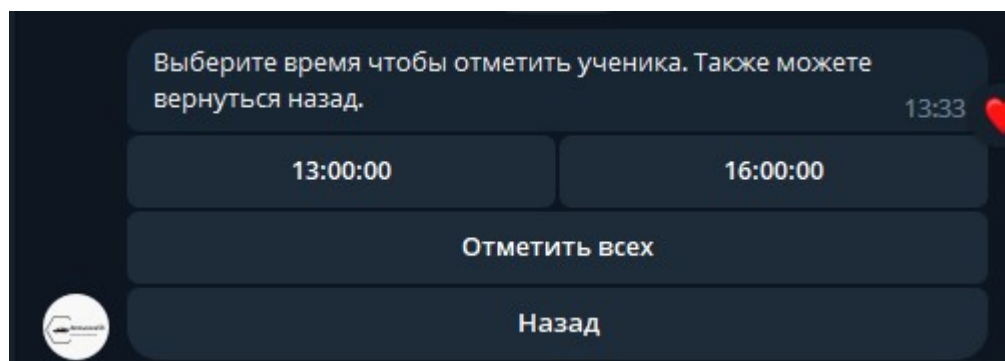


Рисунок 1.72 – Процесс отметки ученика

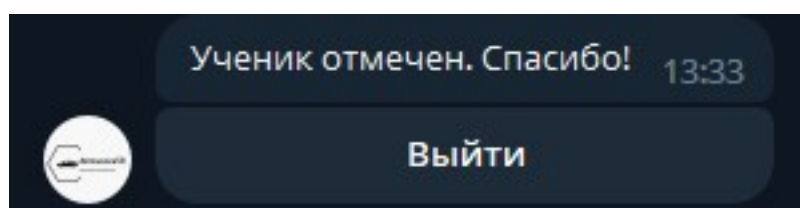


Рисунок 1.73 – Процесс отметки ученика

Следующим, рассмотрим блок управления администратора автошколы. Администратор может управлять всеми базами данных, а именно, добавлять, удалять и изменять поля определённой таблицы. Для доступа к этому меню, администратор пишет команду */admin*. После этого бот запрашивает пароль, хэширует и сверяет его с хэшем, записанным в конфигурационном файле с паролем. Процесс доступа к данному меню показан на рисунках 1.74, 1.75.

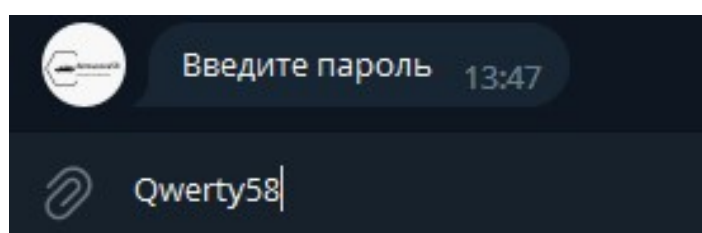


Рисунок 1.74 – Авторизация администратора

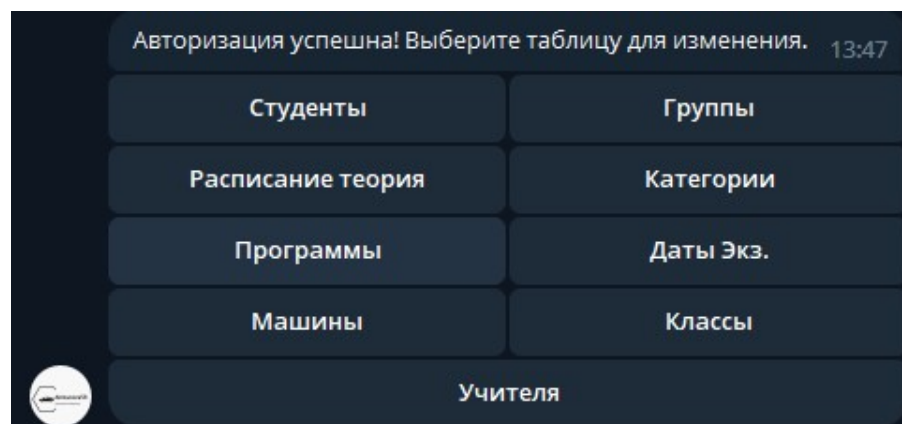


Рисунок 1.75 – Выбор таблицы

Теперь, для примера рассмотрим полное изменение таблицы «Студенты». Для этого нужно нажать на соответствующую кнопку «Студенты». А далее выбрать план действий, для начала добавить студента. Процесс показан на рисунке 1.76.

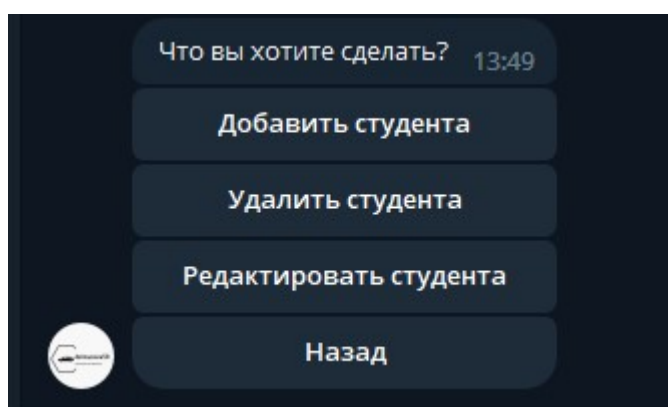


Рисунок 1.76 – Выбор действия

Далее бот информирует о том, как нужно передать данные для новой записи. В данном случае это показано на рисунке 1.77.

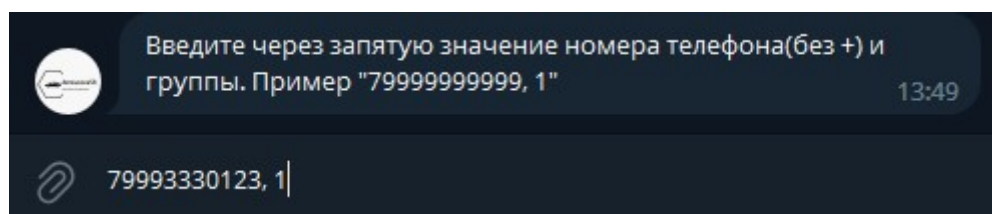


Рисунок 1.77 – Ввод новых данных

Если данные будут переданы неверно, то бот не запишет новую строку и оповестит об ошибке. Это показано на рисунке 1.78.

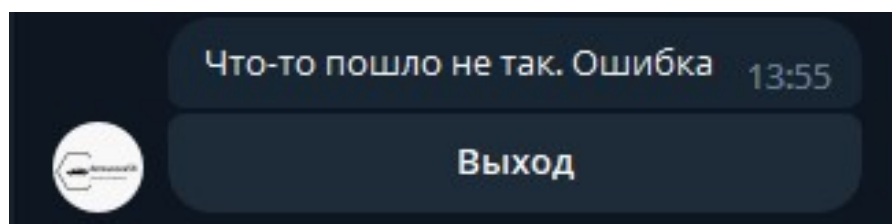


Рисунок 1.78 – Уведомление об ошибке

Если все данные корректны, то бот оповещает об успешной записи. Это демонстрирует рисунок 1.79.

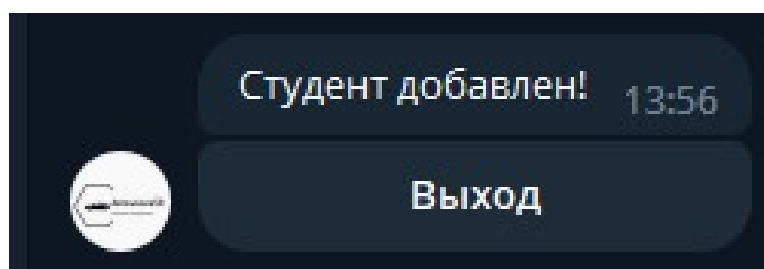


Рисунок 1.79 – Успешное добавление

Следующим, рассмотрим процесс редактирования студента. Для этого нажимаем на кнопку «Редактировать студента». Бот просит указать *id* студента, для поиска нужной строки, в нашем случае это 5. Далее нужно выбрать колонку для изменения, из предложенных, изменим колонку «*name*». После этого нужно передать новое значение и бот изменит это поле. Рисунки 1.80, 1.81, 1.82, 1.83 это демонстрируют.

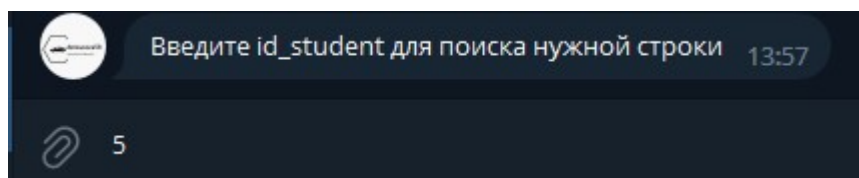


Рисунок 1.80 – Поиск нужного поля



Рисунок 1.81 – Выбор колонки

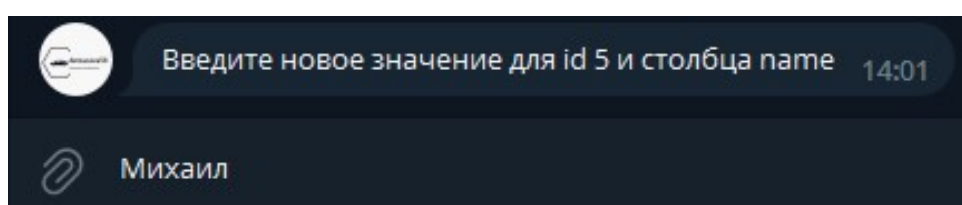


Рисунок 1.82 – Ввод значения

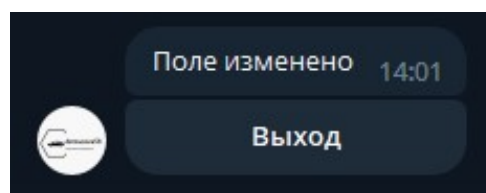


Рисунок 1.83 – Уведомление об изменении

Теперь проверим нашу запись непосредственно в СУБД, для этого найдем студента с *id* 5. Результат показа на рисунке 1.84.

	id_student [PK] integer	user_id bigint	phone_number text	name text	age date	passport text	medical text
1	1	485238345	79521955345	Яцков Владислав Вадимович	2001-05-28	1234 53232	12 123123321
2	5	[null]	79993330123	Михаил	[null]	[null]	[null]

Рисунок 1.84 – Просмотр БД

Теперь нужно удалить этого студента, для этого нажимаем на кнопку «Удалить студента» и вводим *id* студента, которого мы хотим удалить. В нашем случае это 5. Процесс удаления показан на рисунках 1.85.

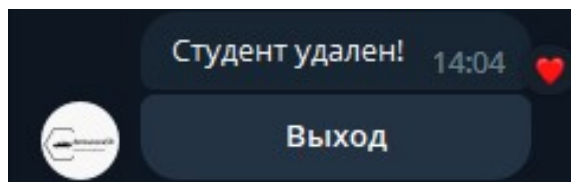


Рисунок 1.85 – Успешное удаление

Аналогичным образом администратор может взаимодействовать с другими таблицами.

1.8.4. Перенос ИС на сервер

После того, как написан весь функционал ИС, нужно описать процесс ее переноса с локального устройства на удаленный сервер. Перенос ИС на сервер нужен для повышения доступности, масштабируемости, улучшения безопасности и удобства обслуживания. После выбора подходящего хостинга и параметров сервера, нужно перейти к его настройке. Для начала подключаемся и обновляем пакеты на сервере командой *sudo apt -y upgrade*, также устанавливаем нужные языки и выбираем кодировку эти делается с помощью команд *dpkg-reconfigure locales* и *LANG=en_US.UTF-8*. После этого нужно перейти к настройке *pip3* и средств разработки, это показано на рисунке 1.86.

```
Setting up libexpat1-dev:amd64 (2.2.9-1ubuntu0.4) ...
Setting up libpython3.8-dev:amd64 (3.8.10-0ubuntu1-20.04.5) ...
Setting up zlib1g-dev:amd64 (1:1.2.11.dfsg-2ubuntu1.4) ...
Setting up cpp (4:9.3.0-1ubuntu2) ...
Setting up libalgorithm-diff-xs-perl (0.04-6) ...
Setting up libalgorithm-merge-perl (0.08-3) ...
Setting up libkrb5-26-heimdal:amd64 (7.7.0+dfsg-lubuntu) ...
Setting up gcc-9 (9.4.0-1ubuntu1-20.04.1) ...
Setting up libpython3-dev:amd64 (3.8.2-0ubuntu2) ...
Setting up libstdc++-9-dev:amd64 (9.4.0-1ubuntu1-20.04.1) ...
Setting up libheimntlm0-heimdal:amd64 (7.7.0+dfsg-lubuntu) ...
Setting up gcc (4:9.3.0-1ubuntu2) ...
Setting up libgssapi3-heimdal:amd64 (7.7.0+dfsg-lubuntu) ...
Setting up g++-9 (9.4.0-1ubuntu1-20.04.1) ...
Setting up python3.8-dev (3.8.10-0ubuntu1-20.04.5) ...
Setting up g++ (4:9.3.0-1ubuntu2) ...
update-alternatives: using /usr/bin/g++ to provide /usr/bin/c++ (c++) in auto mode
Setting up build-essential (12.8ubuntu1.1) ...
Setting up libldap-2.4-2:amd64 (2.4.49+dfsg-2ubuntu1.9) ...
Setting up dirmngr (2.2.19-3ubuntu2.2) ...
Created symlink /etc/systemd/user/sockets.target.wants/dirmngr.socket → /usr/lib/systemd/user/dirmngr.socket
Setting up python3-dev (3.8.2-0ubuntu2) ...
Setting up gpg-wks-client (2.2.19-3ubuntu2.2) ...
Setting up gnupg (2.2.19-3ubuntu2.2) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for install-info (6.7.0.dfsg.2-5) ...
root@vm3681951:~# sudo apt -y install build-essential libssl-dev libffi-dev python3-dev
```

Рисунок 1.86 – Установка *pip3* и *python*

Следующим этапом будет установка виртуального окружения *venv* и установка *PostgreSQL*. Это показано на рисунках 1.87 и 1.88.

```
root@vm3681951:~# sudo apt install -y python3-venv
Reading package lists... Done
Building dependency tree
Reading state information... Done

```

Рисунок 1.87 – Установка *venv*

```
root@vm3681951:~# sudo apt -y install postgresql postgresql-contrib
Reading package lists... Done
Building dependency tree... 50%

```

Рисунок 1.88 – Установка *PostgreSQL*

Далее нужно настроить пользователя *Postgres*, этот этап показан на рисунке 1.89.

```
root@vm3681951:~# sudo -i -u postgres
postgres@vm3681951:~$ createuser --interactive
Enter name of role to add: gino
Shall the new role be a superuser? (y/n)

```

Рисунок 1.89 – Настройка пользователя

Далее нам нужно перенести все файлы программы на сервер, а также создать резервную копию БД и установить ее на сервер. Перенесенные файлы показаны на рисунке 1.90.

Имя файла	Размер	Тип файла	Последнее из...	Права	Владелец/Г...
..					
__pycache__		Папка с ф...	25.05.2023 13:1...	drwxr-xr-x	root root
databases		Папка с ф...	25.05.2023 8:44...	drwxr-xr-x	root root
handlers		Папка с ф...	25.05.2023 8:44...	drwxr-xr-x	root root
images		Папка с ф...	25.05.2023 8:45...	drwxr-xr-x	root root
invoice		Папка с ф...	25.05.2023 8:46...	drwxr-xr-x	root root
marshrut		Папка с ф...	25.05.2023 8:46...	drwxr-xr-x	root root
venv		Папка с ф...	25.05.2023 12:1...	drwxr-xr-x	root root
__init__.py	88	Исходный...	25.05.2023 8:44...	-rw-r--r--	root root
admin.txt	87	Текстовый...	25.05.2023 8:44...	-rw-r--r--	root root
backup.sql	849 620	Исходный...	25.05.2023 11:5...	-rw-r--r--	root root
create_bot.py	352	Исходный...	25.05.2023 8:55...	-rw-r--r--	root root
db.py	22 013	Исходный...	25.05.2023 13:1...	-rw-r--r--	root root
invoice53485238345.h...	4 034	Yandex Bro...	25.05.2023 8:44...	-rw-r--r--	root root

Рисунок 1.90 – Файлы программы на сервере.

После этого, нужно создать новую БД на сервере с помощью команды *createdb autoschool58* и поместить в нее резервную копию БД, командой *pg_dump -Upostgres -dautoschool58 >backup.sql*. Когда БД перенесена, нужно установить все пакеты, которые нужны для успешной компиляции программы, все используемые библиотеки хранятся в файле *requirements.txt*. Для установки нужно войти в виртуальное окружение и воспользоваться командой *pip3 install -rrequirements.txt*. Пример показан на рисунке 1.91.

```
(venv) root@vm3681951:~/bot# pip3 install -r requirements.txt
Collecting python-dotenv==0.19.2
  Downloading python_dotenv-0.19.2-py2.py3-none-any.whl (17 kB)
Collecting SQLAlchemy==1.3.24
  Downloading SQLAlchemy-1.3.24-cp38-cp38-manylinux2010_x86_64.whl (1.3 MB)
    | ██████████ | 1.3 MB 10.6 MB/s
```

Рисунок 1.91 – Установка нужных пакетов

Когда все готово к запуску программы, нужно создать скрин, с помощью которого бот будет работать автономно. Это делается с помощью команды *screen -S bot*. Далее нужно запустить бот и убедиться в правильности его работы, пример показан на рисунке 1.92.

```
root@1549517-ci73280:~/bot# python3 start_bot.py
Updates were skipped successfully.
```

Рисунок 1.92 – Запуск бота на сервере

1.8.5. Описание программы

Диаграмма развертывания, приведенная на рисунке 1.93, используется для визуализации физического размещения компонентов системы и их взаимодействия на уровне аппаратного и программного обеспечения. Она позволяет представить архитектуру системы, указывая, на каких устройствах или серверах расположены компоненты приложения и как они связаны друг с другом.

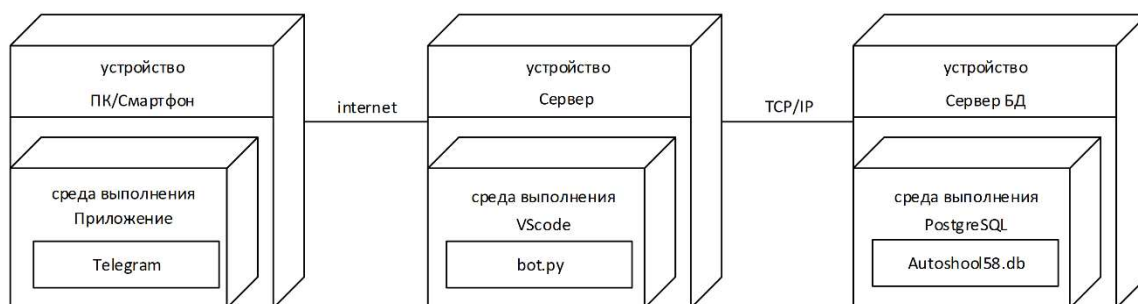


Рисунок 1.93 - UML-диаграмма развертывания

Файл *gitignore*— это файл конфигурации, используемый в системе контроля версий *Git* для указания файлов и папок, которые должны быть проигнорированы и не отслеживаемы *Git* при выполнении операций коммита и других действий.

Файл *admin.txt* - файл, в котором хранится логин и хэш пароля админа, используемый для аутентификации и авторизации администратора в системе.

Файл *create_bot.py* - файл, в котором хранятся конфигурационные константы для успешного запуска и использования бота.

```
from db import Database
host_db="localhost",
port_db = "5432",
database="Autoshool58",
user="postgres",
password_db="Qwerty58"

payment_token = '401643678:TEST:c7d71cee-0944-40ea-9955-3a04a75bc28a'
bot =
Bot(token='6132924794:AAEGdkuvbN_lOCnlThLVFHCPwvNKafdgmic')
dp = Dispatcher(bot,storage=MemoryStorage())
mail = 'autoschool058@mail.ru'
email_token= 'cdjMzQQ96qt6fHnMbxss'
host='smtp.mail.ru'
port=465
```

Файл *start_bot.py* содержит в себе *main* функцию, а также инициализирует все переданные в бот хэндлеры

```
from handlers import menu, lk, rec_lesson, exam_pdd,
other, payment, edit_lesson, command_menu
from create_bot import dp
```

```

menu.register_handlers(dp)
lk.register_handlers(dp)
rec_lesson.register_handlers(dp)
payment.register_handlers(dp)
edit_lesson.register_handlers(dp)
exam_pdd.register_handlers(dp)
command_menu.register_handlers(dp)
other.register_handlers(dp)

if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)

```

Визуальный интерфейс телеграм-бота в *Python* можно создавать с использованием *TelegramBotAPI* и различных библиотек, таких как *python-telegram-bot*. Ниже перечислены некоторые виды визуальных элементов, которые можно использовать в интерфейсе телеграм-бота.

- текстовые сообщения. Бот может отправлять текстовые сообщения пользователю. Это может быть простой текстовый ответ или инструкции для пользователя.

- кнопки *Inline*. Кнопки могут использоваться для предоставления пользователю опций выбора или для выполнения определенных действий. *Inline* кнопки можно размещать непосредственно в тексте сообщения.

- *Reply* кнопки. *Reply* кнопки показываются внизу экрана в виде списка.

- инлайн-клавиатура. Инлайн-клавиатура — это набор кнопок, которые отображаются внизу экрана и позволяют пользователю выбирать опции или выполнять действия в контексте текущего сообщения. Таким образом сделан календарь, а также выбор категории для тестирования.

- всплывающие сообщения. Всплывающие сообщения обычно реализуются с использованием *Inline*-режима или *Inline*-уведомлений. Всплывающие сообщения могут содержать дополнительную информацию или предложения для пользователя и отображаться поверх текущего чата или сообщения.

- вложения (фото, видео, аудио, файлы). Бот может отправлять пользователю вложения, такие как фотографии, видео, аудио или файлы.

– скрытый текст в сообщении. Скрытый текст в сообщении телеграм-бота представляет собой текстовую информацию, которая скрыта от обычного отображения и требует специальных действий или команд пользователя для его просмотра. Это может быть полезно, когда вы хотите предоставить пользователю дополнительную информацию или контекст, но не хотите перегружать основное сообщение.

После того как пользователь отправляет какой-либо текст, команду или нажатие на кнопку, действие фильтруется и обрабатывается в определенном модуле. Ниже перечислены модули, которые разделены для каждого подменю интерфейса и включают в себя необходимые функции.

Модуль *menu.py* – главное меню, в которое попадает пользователь после команды */start*, а также проходит авторизацию или подает заявку на обучение. Функция *cmd_menu* отображает главное меню в телеграм-боте, предоставляя пользователю доступ к различным функциям, в зависимости от наличия авторизации. Функция *cmd_start* используется для обработки команды старта. Ее цель – авторизовать пользователя и предоставить доступ к главному меню, если пользователь не был ранее зарегистрирован. Функция *get_auto* используется в дипломной работе для обработки получения авторизационных данных от пользователя. Она проверяет наличие номера телефона в базе данных и предоставляет доступ к регистрации в личном кабинете или возможность оставить заявку на обучение, в зависимости от результата проверки. Функция *info_menu* используется для отображения информации о автошколе и ее контактных данных. Она также предоставляет кнопку для просмотра местоположения на карте. Ниже приведен пример вызова этих функций с помощью хэндлеров.

```
def register_handlers(dp: Dispatcher):  
    dp.register_message_handler(cmd_menu, commands=['menu'] )  
    dp.register_message_handler(cmd_menu, lambda msg: msg.text  
== "Отклонить" )  
    dp.register_message_handler(info_menu, lambda msg: msg.text  
== "? Информация." )  
    dp.register_message_handler(cmd_start, commands=['start'] )
```

```

dp.register_callback_query_handler(sign, lambda c:
c.data.startswith('sign_'))
dp.register_callback_query_handler(bd_add, lambda c:
c.data.startswith('s_'))
dp.register_callback_query_handler(delete, lambda c:
c.data.startswith('del'))
dp.register_message_handler(get_auto,
content_types=types.ContentType.CONTACT )

```

Модуль *lk.py* – личный кабинет, подменю в котором пользователь может взаимодействовать с личными данными. Функция *with_puree* используется для обработки запроса в личный кабинет пользователя. Она отображает информацию о пользователе, такую как никнейм, статус и баланс, а также предоставляет кнопки для заполнения, просмотра и изменения данных, а также просмотра маршрута. Функция *auto* выполняет действия в личном кабинете пользователя, такие как заполнение данных, изменение уже заполненных данных, просмотр данных и просмотр маршрута, в зависимости от значения *callback.data*. Функция *set_data* обрабатывает нажатия на кнопки для заполнения персональных данных, в зависимости от выбранной кнопки задает соответствующий вопрос и устанавливает состояние пользовательского состояния. Функция *set_personal_data* сохраняет введенные пользователем данные в базе данных и выводит подтверждение, предлагая пользователю верифицировать данные. Аналогичным образом работают функции *edit_data* и *edit_personal_data*, но предназначены для изменения уже заполненных данных. Ниже приведен пример вызова этих функций с помощью хэндлеров.

```

def register_handlers(dp: Dispatcher):

    dp.register_message_handler(lk, lambda
message: "👤 Личный кабинет" in message.text )
    dp.register_callback_query_handler(with_puree, lambda
callback: callback.data in ["back_to_lk"] )
    dp.register_callback_query_handler(auto, lambda callback:
callback.data in ["log
in", "view_data", "edit_data", "view_marshrut", "history_balance"] )
    dp.register_callback_query_handler(edit_data, lambda
callback: callback.data in
["edit_name", "edit_pasport", "edit_medical", "edit_email", "edit_age"]
)

```

```

dp.register_callback_query_handler(set_data, lambda
callback: callback.data in
["set_name","set_pasport","set_medical","set_email","set_age"] )
dp.register_message_handler(set_personal_data,
state=[UserState.name, UserState.passport,
UserState.medical,UserState.email,UserState.age],
content_types=types.ContentTypes.TEXT)
dp.register_message_handler(edit_personal_data,
state=[UserState.name_edit, UserState.passport_edit,
UserState.medical_edit,UserState.email_edit,UserState.age_edit],
content_types=types.ContentTypes.TEXT)

```

Модуль *rec_lesson.py* – подменю, в котором пользователь может записаться на занятия и экзамены. В функции *rec_menu* обрабатывается сообщение пользователя в меню записи на вождение. Затем формируется клавиатура с кнопками "Занятие", "Теоретический экзамен" и "Практический экзамен". После этого отправляется сообщение с информацией о записи, инструкциями и доступными опциями. Пользователь также получает указание о том, как вернуться в главное меню. В функции *rec* обрабатывается выбор пользователя в меню записи на вождение. В зависимости от выбранной опции ("Занятие", "Теоретический экзамен" или "Практический экзамен"), устанавливаются соответствующие флаги *lesson*, *t_exam* и *p_exam*. Если выбрана опция "Занятие", проверяется количество уже записанных занятий. Если пользователь еще может записаться на занятие, вызывается функция *show_calendar* для отображения календаря и возможности выбора даты занятия. В противном случае пользователю выводится сообщение о том, что он не может записаться на больше 5 занятий одновременно, и вызывается функция *rec_menu* для возврата к меню записи. Если выбрана опция "Теоретический экзамен" или "Практический экзамен", вызывается функция *exams*, которая отображает доступные даты для экзамена. В зависимости от выбранного типа экзамена (*t_exam* или *p_exam*), формируется соответствующая клавиатура с датами экзаменов. После выбора даты пользователем вызывается функция *back_to_rec* для возврата к меню записи. Если выбрана опция "Назад", вызывается функция *rec_menu* для возврата к меню записи.

Функция *show_calendar* отображает календарь пользователю, чтобы он мог выбрать дату. Внутри функции вызывается *create_calendar*, которая создает клавиатуру с датами. Клавиатура состоит из трех строк: первая строка содержит стрелки для перехода к предыдущему и следующему месяцам, вторая строка содержит названия дней недели, а остальные строки содержат дни месяца. Каждая кнопка с днем содержит соответствующий *callback_data*, в котором указывается тип записи (*lesson*, *t_exam* или *p_exam*) и выбранная дата. Функции *calendar_prev_month_callback* и *calendar_next_month_callback* обрабатывают нажатия на кнопки "вперед" и "назад" для переключения между предыдущим и следующим месяцами в календаре. В *callback_data* передаются текущий год, месяц и тип записи (*lesson*, *t_exam* или *p_exam*), а затем вызывается функция *create_calendar* для создания новой клавиатуры с выбранным месяцем.

Функция *show_time_picker* отображает пользователю выбор времени на определенный день. Она получает массив доступных временных слотов для выбранного дня из базы данных и создает клавиатуру с кнопками для каждого доступного времени. Клавиатура также содержит кнопку "Назад" для возврата к меню записи.

Функция *process_lesson_callback* обрабатывает нажатия на кнопки выбора времени в календаре. В зависимости от выбранного времени, функция проверяет, доступно ли выбранное время для записи. Если время уже занято, прошло или является выходным днем, выводится соответствующее сообщение и предлагается вернуться назад. Если время доступно для записи, выводится сообщение с подтверждением выбранной даты и времени, а также кнопкой "Записаться" и кнопкой "Назад".

Функция *recover* обрабатывает обратные вызовы при записи пользователя на занятие, теоретический экзамен или практический экзамен, добавляет запись в базу данных, а также формирует чек оплаты и высылает его на привязанный электронный адрес. Ниже приведен пример вызова этих функций с помощью хэндлеров.

```

def register_handlers(dp: Dispatcher):
    dp.register_message_handler(rec_menu, lambda message:
message.text == "🚗Записаться на вождение" )
    dp.register_callback_query_handler(rec, lambda callback:
callback.data in ["lesson", "t_exam", "p_exam", "back_to_rec"] )
    dp.register_callback_query_handler(recover, lambda query:
query.data.startswith('rec'))

dp.register_callback_query_handler(calendar_next_month_callback,
lambda query: query.data.startswith('calendar-next-month'))

dp.register_callback_query_handler(calendar_prev_month_callback,
lambda query: query.data.startswith('calendar-prev-month'))

dp.register_callback_query_handler(process_calendar_day,
lambda c: c.data.startswith('calendar-day'))
dp.register_callback_query_handler(process_lesson_callback,
lambda c: c.data.startswith('lesson-time-'))

```

Модуль *edit_lesson.py* – подменю, в котором пользователь может просматривать и редактировать свое расписание.

Функция *les_menu* используется для отображения меню управления расписанием. Затем функция отвечает пользователю, предлагая выбрать нужный пункт меню.

Функция *view_works* используется для отображения списка доступных занятий или экзаменов для пользователя. В зависимости от выбранного типа (занятия или экзамены), функция извлекает соответствующую информацию из базы данных и формирует соответствующую клавиатуру с кнопками для каждого элемента списка. Затем функция отправляет сообщение с выбором пользователю.

Функция *edit_works* используется для отображения информации о выбранном занятии или экзамене. В зависимости от типа (занятие или экзамен) и даты, функция формирует сообщение с информацией о записи и правилах отмены. Если запись может быть отменена (если дата еще не наступила), функция добавляет кнопку "Отменить запись". Затем функция отправляет сообщение с информацией пользователю.

Функция *return_works* используется для обработки отмены занятия или экзамена. В зависимости от выбранного типа (занятие или экзамен), функция

вызывает соответствующие методы базы данных для удаления записи. При отмене занятия также происходит обновление баланса пользователя, генерация и отправка чека о возврате денежных средств. Затем функция отправляет ответное сообщение с подтверждением отмены и возвращает пользователя в меню управления расписанием. Ниже приведен пример вызова этих функций с помощью хэндлеров.

```
def register_handlers(dp: Dispatcher):
    dp.register_message_handler(les_menu, lambda message:
message.text == "📅Управление расписанием" )
    dp.register_callback_query_handler(view_works, lambda
query: query.data.startswith('view_'))
    dp.register_callback_query_handler(edit_works, lambda
query: query.data.startswith('work_'))
    dp.register_callback_query_handler(return_works, lambda
query: query.data.startswith('return_'))
```

Модуль *exam_pdd.py* – подменю, в котором пользователь может проходить теоретическое тестирование и готовиться к экзамену.

Функция *menu_exam* используется для отображения меню тестирования. При вызове функции происходит удаление предыдущего сообщения, если пользователь выбрал опцию "Пройти экзамен ПДД". Затем функция создает клавиатуру с пунктами меню, такими как "Билеты", "Категории", "Самые сложные", "Экзамен" и "Ошибки". Каждая кнопка имеет соответствующий *callback_data*. После создания клавиатуры, функция отправляет сообщение с меню тестирования и клавиатурой пользователю.

Функция *mode_quiz* обрабатывает выбор пользователя в меню тестирования. В зависимости от выбранной опции, функция выполняет определенные действия. Если выбрана опция "Экзамен", функция генерирует список вопросов экзамена и сохраняет его в базе данных. Затем она вызывает функцию *send_question*, чтобы отправить первый вопрос экзамена пользователю.

Если выбрана опция "Билеты", функция создает клавиатуру с номерами билетов и отправляет ее пользователю. Если выбрана опция "Сложные вопросы", функция получает сложные вопросы из базы данных, сохраняет их список вопросов в базе данных и отправляет первый вопрос пользователю.

Если выбрана опция "Категории", функция вызывает функцию *show_category*, которая отображает список доступных категорий вопросов.

Если выбрана опция "Ошибки", функция проверяет наличие ошибок в предыдущем тесте пользователя. Если ошибок нет, отправляется соответствующее сообщение. Если есть ошибки, функция сохраняет список вопросов с ошибками в базе данных и вызывает функцию *send_question* для отправки первого вопроса пользователю.

Функция *bilet_mode* отвечает за обработку команды пользователя для режима "Билет". Вначале она удаляет сообщение, в котором была нажата кнопка. Затем из базы данных получает все вопросы для выбранного билета. Полученные вопросы сортируются по порядку. Далее создается идентификатор теста, вызывается функция *set_questions_quiz* для сохранения вопросов теста и устанавливается начальный индекс вопроса. Затем отправляется первый вопрос с помощью функции *send_question*.

Функция *category_mode* отвечает за обработку команды пользователя для режима "Категория". Она также удаляет сообщение, в котором была нажата кнопка. Затем получает индекс выбранной категории и из базы данных получает все вопросы для этой категории. Полученные вопросы сортируются по порядку. Создается идентификатор теста, вызывается функция *set_questions_quiz* для сохранения вопросов теста и устанавливается начальный индекс вопроса. Затем отправляется первый вопрос с помощью функции *send_question*.

Функция *send_question* используется для отправки вопроса пользователю. Она получает информацию о текущем вопросе из списка ответов. Затем формирует сообщение с текстом вопроса и вариантами ответов. Если есть картинка, она добавляется к сообщению. Правильный ответ сохраняется. Создается клавиатура с вариантами ответов и кнопкой "Закончить тест". Затем сообщение с вопросом и клавиатурой отправляется пользователю.

Функция *check_answer* используется для обработки выбора ответа пользователем. Она получает выбранный ответ, правильный ответ, индекс

вопроса, идентификатор теста, идентификатор ответа и аннотацию (способ отображения подсказки). Если выбранный ответ верный, увеличивается количество правильных ответов. Затем получают все вопросы теста из базы данных. Если есть еще вопросы, отправляется следующий вопрос с помощью функции *send_question*. Если вопросы закончились, выводятся результаты теста.

Функция *show_category* используется для отображения категорий пользователю. Она создает клавиатуру с категориями и отправляет ее пользователю. Клавиатура содержит кнопки для выбора категорий, кнопку "назад" для перехода к предыдущим категориям, кнопку "вперед" для перехода к следующим категориям. Ниже приведен пример вызова этих функций с помощью хэндлеров.

```
def register_handlers(dp: Dispatcher):
    dp.register_message_handler(menu_exam, lambda message:
message.text == "✓Пройти экзамен ПДД" )
    dp.register_callback_query_handler(mode_quiz, lambda query:
query.data.startswith('mode_'))
    dp.register_callback_query_handler(bilet_mode, lambda
query: query.data.startswith('bilet_'))
    dp.register_callback_query_handler(check_answer, lambda
query: query.data.startswith('answer_'))
    dp.register_callback_query_handler(cat_next_callback,
lambda query: query.data.startswith('cat-next-'))
    dp.register_callback_query_handler(cat_prev_callback,
lambda query: query.data.startswith('cat-prev-'))
    dp.register_callback_query_handler(category_mode, lambda
query: query.data.startswith('category_'))
```

Модуль *payment.py* – раздел, в котором студент может пополнять баланс, просматривать его историю, а также выводить деньги на карту.

Функция *payment_menu* отвечает за отображение меню управления балансом пользователя. Затем она создает клавиатуру с кнопками для пополнения баланса, вывода средств и просмотра истории пополнений. Функция отправляет сообщение с этой клавиатурой пользователю.

Функция *confirm_return_money* отвечает за подтверждение запроса на возврат средств. Она получает баланс пользователя, сохраняет информацию о возврате в базе данных и обновляет баланс пользователя. Затем функция

отправляет ответ пользователю с помощью метода *answer* и вызывает функцию *payment_menu* для отображения меню управления балансом.

Функция *operation_money_callback* обрабатывает нажатие кнопок в меню управления балансом. Если пользователь выбирает опцию пополнения баланса, функция проверяет наличие корректного *email* адреса у пользователя. Если *email* адрес указан некорректно или не указан вовсе, функция отправляет сообщение с кнопкой "Назад". В противном случае, функция создает клавиатуру с кнопками для выбора суммы пополнения и отправляет сообщение пользователю.

Если пользователь выбирает опцию запроса на возврат средств, функция проверяет баланс пользователя. Если баланс равен 0, функция отправляет сообщение с кнопкой "Назад". В противном случае, функция создает клавиатуру с кнопкой для подтверждения возврата и отправляет сообщение пользователю.

Если пользователь выбирает опцию просмотра истории пополнений, функция получает информацию о транзакциях пользователя из базы данных. Затем функция формирует текст сообщения с информацией о транзакциях и отправляет его пользователю. Если у пользователя нет транзакций, функция отправляет сообщение о том, что у него еще не было транзакций.

Функция *add_balance* обрабатывает выбор суммы пополнения баланса. Если пользователь выбирает кнопку "Назад", функция вызывает функцию *payment_menu* для отображения меню управления балансом. В противном случае, функция удаляет сообщение с кнопками пополнения, создает объект с информацией о платеже и отправляет запрос на оплату пользователю с помощью метода *send_invoice*.

Функция *payment* формирует объект с информацией о платеже на основе данных из выбранной кнопки пополнения. Она возвращает *chat_id* и словарь с информацией о платеже.

Функция *checkout_process* обрабатывает результат предварительной проверки платежа. Если платеж не прошел успешно, функция отправляет ответ с ошибкой. В противном случае, функция отправляет успешный ответ.

					ПГУ1.090301.39.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

Функция *process_successful_payment* обрабатывает успешный платеж. Она получает информацию о платеже из сообщения и сохраняет его данные в базе данных с помощью функции *set_payment*. Затем функция обновляет баланс пользователя и генерирует, и отправляет чек на *email*. Ниже приведен пример вызова этих функций с помощью хэндлеров.

```
def register_handlers(dp: Dispatcher):
    dp.register_message_handler(payment_menu, lambda message:
    message.text == "❶Управление балансом" )

    dp.register_callback_query_handler(operation_money_callback, lambda
    query: query.data.startswith('operation_'))
    dp.register_callback_query_handler(confirm_return_money,
    lambda query: query.data == 'confirm_return')
    dp.register_callback_query_handler(add_balance, lambda
    query: query.data.startswith('add_money_'))

    dp.register_pre_checkout_query_handler(checkout_process, lambda q:
    True)
```

Модуль *command_menu.py* – раздел, в котором объединены интерфейсы инструктора и администратора.

Функция *admin_sign* отвечает за авторизацию администратора. Она проверяет, является ли отправитель сообщения администратором, и, если это так, запрашивает пароль. Функция принимает сообщение *message*, объект состояния *state* и необязательный параметр *id*, который используется для указания альтернативного идентификатора пользователя (например, при входе в систему от имени другого пользователя).

Функция начинается с удаления сообщения, чтобы скрыть информацию о логине и пароле администратора. Затем она читает данные администратора из файла *admin.txt*, включая логин и хэшированный пароль.

Далее функция проверяет, является ли отправитель сообщения администратором, сравнивая идентификатор отправителя с логином администратора. Если идентификатор совпадает, функция переходит к следующему шагу и сохраняет хэшированный пароль администратора в состоянии *state*. Затем она отправляет сообщение с просьбой ввести пароль и устанавливает состояние *LoginStates.PASSWORD_STATE*.

Функция *password_handler* является обработчиком сообщений, которые ожидают пароль администратора. Она получает введенный пароль от пользователя и сравнивает его с сохраненным хэшированным паролем администратора. Если пароль верный, функция завершает состояние *state*, иначе она отправляет сообщение о отказе в доступе.

Функция *admin_menu* отображает меню администратора после успешной авторизации. Она создает и отправляет клавиатуру с опциями для изменения различных таблиц (студенты, группы, расписание и т. д.) и ожидает ответа от пользователя.

Функция *teacher_menu* представляет меню преподавателя. Если пользователь имеет права преподавателя, отображается меню с различными функциями. Если пользователь не имеет прав доступа, выводится сообщение о отсутствии доступа.

Функция *teacher_mode* обрабатывает запросы, связанные с функциями меню преподавателя. Она принимает объект *types.CallbackQuery*, который представляет *callback*-запрос от пользователя.

Функция содержит несколько условий для различных типов запросов, например, запросы для просмотра данных преподавателя, просмотра расписания на день, отмены занятий или отметки ученика.

В каждом условии выполняются соответствующие действия, такие как удаление сообщения, получение данных из базы данных, создание клавиатуры с кнопками и отправка сообщения с клавиатурой.

Функция *get_lessons_day* обрабатывает запросы, связанные с выбором дня и просмотром/отменой занятий в этот день. Она также принимает объект *types.CallbackQuery*.

Функция проверяет тип запроса (просмотр, отмена или отметка) и выполняет соответствующие действия. Например, если запрос связан с просмотром занятий, получается список занятий на выбранный день, создается

клавиатура с кнопками для каждого занятия, и отправляется сообщение с клавиатурой.

Аналогичные действия выполняются для других типов запросов, таких как отмена занятий или отметка ученика. Функции *teacher_menu*, *teacher_mode* и *get_lessons_day* используются вместе для создания меню преподавателя и обработки связанных запросов. Ниже приведен пример вызова этих функций с помощью хэндлеров.

```
def register_handlers(dp: Dispatcher):
    dp.register_message_handler(teacher_menu,
    commands=['teacher'] )
    dp.register_message_handler(admin_sign, commands=['admin']
    )
    dp.register_callback_query_handler(teacher_mode, lambda
    query: query.data.startswith('teach_'))
    dp.register_callback_query_handler(admin_mode, lambda
    query: query.data.startswith('admin_'))
    dp.register_callback_query_handler(get_lessons_day, lambda
    query: query.data.startswith('tach-les'))
    dp.register_callback_query_handler(admin_option, lambda
    query: query.data.startswith('adm_'))
    dp.register_callback_query_handler(return_mode, lambda
    query: query.data.startswith('t-return'))
    dp.register_callback_query_handler(view_mode, lambda query:
    query.data.startswith('t-view'))
    dp.register_callback_query_handler(ed_mode, lambda query:
    query.data.startswith('ed-'))
    dp.register_callback_query_handler(rules_mode, lambda
    query: query.data.startswith('t-rul'))
    dp.register_message_handler(add_mode,
    state=[DBStates.newProgram, DBStates.newExam, DBStates.newTable,
    DBStates.newGroup, DBStates.newStudent, DBStates.newCategory,
    DBStates.newCar, DBStates.newClass, DBStates.newTeacher],
    content_types=types.ContentTypes.TEXT)
    dp.register_message_handler(del_mode,
    state=[DBStates.delProgram, DBStates.delExam, DBStates.delTable,
    DBStates.delGroup, DBStates.delStudent, DBStates.delCategory,
    DBStates.delCar, DBStates.delClass, DBStates.delTeacher],
    content_types=types.ContentTypes.TEXT)
    dp.register_message_handler(edit_mode,
    state=[DBStates.editProgram, DBStates.editExam, DBStates.editTable,
    DBStates.editGroup, DBStates.editStudent, DBStates.editCategory,
    DBStates.editCar, DBStates.editClass, DBStates.editTeacher],
    content_types=types.ContentTypes.TEXT)
    dp.register_message_handler(ed_op,
    state=[DBStates.newrowProgram, DBStates.newrowExam, DBStates.newrowTab
    le, DBStates.newrowGroup, DBStates.newrowStudent,
```

```
DBStates.newrowCategory, DBStates.newrowCar, DBStates.newrowClass,  
DBStates.newrowTeacher], content_types=types.ContentTypes.TEXT
```

Модуль *other.py* – раздел, в который попадают все сообщения, которые не попали в предыдущие модули, обработка ошибочного сообщения.

Функция *echo* используется для обработки сообщений, которые не могут быть распознаны или поняты моделью. Она принимает сообщение пользователя и отправляет обратно то же самое сообщение с призывом вернуться в главное меню командой */menu*.

```
def register_handlers(dp: Dispatcher):  
    dp.register_message_handler(echo)
```

2 Безопасность жизнедеятельности

2.1 Правила безопасности при использовании ПК

В данном разделе необходимо учесть риски, перечисленные в ГОСТ 12.0.003-2015 "Система стандартов безопасности труда. Электронно-вычислительная техника. Общие требования безопасности". Ниже приведен перечень рисков, которым следует придать особое значение:

– риск повреждения зрения и утомления глаз:

1) длительное использование ПК без перерывов и отдыха глаз;

2) плохая эргономика рабочего места, включая неправильное расположение монитора, отражение света на экране и недостаточное освещение;

– риск травм и заболеваний опорно-двигательной системы:

1) неправильное положение тела при работе с ПК, например, неправильная осанка, неправильная поза или неправильное расположение рук и ног;

2) отсутствие эргономической мебели и принадлежностей, таких как стулья с подлокотниками и подставки для рук;

– риск воздействия электромагнитных полей:

1) использование устаревшего или неисправного оборудования, которое может создавать сильные электромагнитные поля;

2) неправильное расположение оборудования, например, слишком близко к рабочему месту или другим электронным устройствам;

– риск пожара и электрического удара:

1) неправильное подключение и использование электрооборудования, включая перегрузку розеток и использование поврежденных кабелей;

2) нарушение правил пожарной безопасности, включая использование запрещенных электроприборов вблизи ПК и наличие горючих материалов рядом с оборудованием;

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		82

– риск нарушения информационной безопасности:

1) небезопасное использование паролей и учетных записей, включая слабые пароли или их совместное использование;

2) несанкционированный доступ к компьютерной информации, включая утечку конфиденциальных данных;

– риск воздействия шума:

1) использование шумных компьютерных компонентов или работа в шумной среде.

2.2 Правила безопасности при использовании смартфона

В этом разделе следует уделить внимание рискам, связанным с безопасностью при использовании данного устройства. А также разработать свод правил, чтобы гарантировать предотвратить следующие риски:

– риск зависимости и негативного воздействия на психологическое и физическое здоровье:

1) чрезмерное использование смартфона, включая зависимость от социальных сетей, игр и постоянного онлайн-присутствия, что может негативно сказываться на здоровье и социальной активности.

– риск отрицательного влияния на сон и здоровье:

1) использование смартфона перед сном, из-за светового излучения экрана, может нарушать естественный сон и вызывать бессонницу;

2) чрезмерное использование смартфона в темное время суток может снижать уровень мелатонина, гормона сна;

– риск потери или кражи смартфона:

1) несанкционированное получение доступа к устройству, возможность потери или кражи смартфона, что может привести к утечке личных данных или использованию их в недобросовестных целях;

2) нарушение конфиденциальности и безопасности данных в случае потери или кражи смартфона;

– риск воздействия радиочастотного излучения:

1) возможные воздействия радиочастотного излучения на организм при длительном использовании смартфона, хотя точные последствия такого воздействия все еще находятся в стадии исследования;

2) при работе в плохо покрытых районах мобильной связи смартфон может выделять больше энергии для подключения к сети, что может привести к повышенному радиочастотному излучению;

– риск отвлечения водителя:

1) использование смартфона во время вождения может отвлекать водителя и повышать риск дорожно-транспортных происшествий. Это особенно актуально в случае чтения сообщений, отправки сообщений или проведения телефонных разговоров во время движения.

С учетом указанных рисков, следует разработать и рекомендовать следующие правила безопасности при использовании смартфона:

- ограничьте время использования смартфона, особенно перед сном;
- соблюдайте эргономику и правильное положение тела при работе со смартфоном;
- используйте оригинальные зарядные устройства и кабели, избегайте перегрева и короткого замыкания;
- устанавливайте надежные пароли и используйте функции блокировки экрана для защиты данных;
- храните смартфон в безопасном месте и будьте бдительны в общественных местах;
- регулярно делайте резервные копии данных и используйте антивирусное программное обеспечение для защиты от вредоносных программ;
- ограничьте использование смартфона во время вождения и соблюдайте правила безопасности на дороге.

Правильное соблюдение этих правил поможет минимизировать риски и обеспечить безопасное использование смартфона.

3 Стартап проект

Инициатор проекта планирует разработку информационной системы на основе телеграм-бота.

Разместить бизнес планируется в г. Пенза помещении, соответствующий всем нормам и требованиям РК.

Целью деятельности предприятия является извлечение дохода для улучшения материального благосостояния его участника. Также упрощение рутинных задач, повышение качества обучения и обеспечение простоты и безопасности взаимодействия между клиентами и организацией. Нужно предоставить автошколам современное решение, которое позволит им эффективно управлять расписанием, изменять личные данные, принимать и обрабатывать платежи, а также проводить тестирование по вопросам ПДД.

Целевой группой планируемого предприятия будут являться автошколы, а при дальнейшем развитии другие коммерческие образовательные учреждения.

3.1 Сегменты ЦА

Целевая аудитория состоит из автошкол и при последующем масштабировании, других коммерческих образовательных учреждений. Для более эффективного взаимодействия с каждым сегментом ЦА, необходимо учитывать их потребности и предпочтения.

Автошколы стремятся предложить своим учащимся качественное обучение вождению, сопровождение на всех этапах подготовки к получению водительского удостоверения, эффективное использование времени и ресурсов. Предпочитают гибкую систему управления расписанием занятий, удобное взаимодействие с учащимися и инструкторами, а также функциональные инструменты для отслеживания прогресса и оценки успехов учащихся.

Каждому сегменту ЦА будет предложено индивидуальное коммерческое предложение, учитывающее их потребности и предпочтения. Стоимость

разработки информационной системы будет варьироваться в зависимости от объема и специфических требований каждого клиента.

Перед запуском проекта важно найти ключевых партнеров для ведения бизнеса, в них должны входить поставщики технического оборудования, автошколы и коммерческие образовательные учреждения.

Поставщиками технического оборудования могут стать компании, которые предоставляют компьютеры, серверы и другое необходимое оборудование для функционирования ИС. Взаимная выгода заключается в обеспечении надежного и качественного оборудования для разработки и предоставления продукта клиентам.

Автошколы и коммерческие образовательные учреждения станут основными клиентами ИС. Сотрудничество с ними позволит предоставить им удобное и эффективное решение для управления и автоматизации процессов автошколы. Взаимная выгода заключается в предоставлении продукта, который поможет им улучшить качество обучения, оптимизировать бизнес-процессы и сэкономить время, финансы и ресурсы.

3.2 Ключевые процессы

Ниже перечислены основные ключевые процессы при разработке стартап-проекта:

- создание ИП;
- поиск и создание команды;
- поиск средств;
- разработка ПО;
- поиск каналов сбыта и взаимодействие с целью продажи;
- поддержка проданного продукта;

Для начала процедуры открытия ИП нужно подготовить необходимые документы и данные. В первую очередь нужно найти ОКВЭДы для своего будущего бизнеса. Для этого нужно воспользоваться специальным справочником

и подобрать наиболее подходящий код. В данном случае это будет – 62.01 «Разработка компьютерного программного обеспечения», а также 62.02 «Консультирование по компьютерным вопросам и связанная деятельность». Вторым этапом станет выбор налогового режима. Выбор был сделан в пользу УСН, потому что этот вид предоставляет простоту в расчете и уплате налогов, а также предлагает низкие ставки налога. Можно самостоятельно рассчитывать и платить налоги раз в квартал, а ежегодно подавать декларацию. Доход будет облагаться налогом по ставке от 6% до 15%, в зависимости от региона. Регион размещения – Пензенская область, в нем установлена ставка 6% согласно [7]. Далее заполнить и подать документы, чтобы не оплачивать госпошлину принято решение подать их через МФЦ. Следующим этапом нужно получить справку из ЕГРИП, она придет на электронную почту, указанную в заявлении – обычно на это уходит 3 дня.

Для разработки такой системы нужно правильно подобрать команду, на начальном этапе в нее должны входить: разработчики, бухгалтер, менеджер по продажам. В нашем случае разработчики найдены из числа единомышленников. Заниматься делами бухгалтерии учетом было поручено бухгалтерам из компании «Мое дело». Менеджер по продажам был найден на сайте с резюме.

Этапы разработки ПО описаны в пункте 1.

Поиск каналов сбыта осуществляется с помощью коммерческих предложений. Коммерческое предложение имеет несколько значимых достоинств, которые помогут привлечь покупателей и отличить этот проект от конкурентов.

Во-первых, телеграм-бот предлагает более доступное ценовое предложение по сравнению с полноценными мобильными приложениями для автошкол. Вместо необходимости инвестировать в разработку и поддержку сложного приложения, клиенты могут получить полноценную информационную систему через мессенджер, что позволяет им сэкономить значительные денежные средства.

В-третьих, мессенджер Telegram набирает большую популярность среди различных возрастных групп и интересов. Разработка продукта, на основе Telegram, позволяет привлечь внимание большей части целевой аудитории, в отличие от конкурирующих средств массовой информации, что влечет за собой наиболее высокий спрос.

Можно отметить, что использование передовых и популярных технологий, таких как Telegram, может повысить имидж компании и показать ее инновационный подход к бизнесу. Клиенты могут воспринимать компанию как современную, внимательную к новым тенденциям и готовую предложить им удобные и современные способы взаимодействия.

Также, на рынке отсутствуют аналогичные информационные системы с таким же обширным функционалом, предлагаемым данным телеграм-ботом. Функционал включает в себя все функции, начиная с управления расписанием и контроля прогресса учащихся до взаимодействия с базой от имени администратора. Это позволяет занять уникальную позицию и быть востребованным проектом в данном сегменте рынка. Ниже в таблице 1 приведены услуги, предложенные в коммерческом предложении.

Таблица 1 – Наименование предложенных услуг

Наименование	Цена реализации
Разработка под организацию	20 000р
Внедрение	5000р
Поддержка	1000р/мес
Масштабируемость	5000р

3.3 Каналы взаимодействия

Одним из способов привлечения клиентов станет «Сайт-визитка». Он предоставляет клиентам доступ к информации о компании и о наших продуктах или услугах в любое время. Потенциальные заказчики смогут подробно

ознакомиться со всем предлагаемым функционалом системы. Контактные формы и возможность обратной связи облегчат коммуникацию с будущими клиентами.

Для более эффективной демонстрации функционала и ценности информационной системы нужно, путем обзвона, согласовать личные встречи с директорами автошкол и других коммерческих образовательных учреждений. На таких встречах можно показать полный спектр возможностей продукта и ответить на вопросы потенциальных клиентов.

После заключения сделки, наша команда будет отвечать за поставку и внедрение информационной системы. Это включает установку и настройку бота в соответствии с потребностями и предпочтениями клиента. Четкое планирование и организация этапов внедрения помогут сэкономить время и обеспечить плавный переход к полноценному использованию системы. Также нужно обеспечить поддержку клиентов в течение нескольких месяцев после покупки продукта. Это включает в себя решение технических проблем, обновление и доработку системы в соответствии с потребностями клиента, а также консультации и обучение пользователей по использованию системы. Быстрый и качественный ответ на запросы клиентов поможет создать положительное впечатление от сотрудничества. Чтобы поддерживать долгосрочные отношения с клиентами, нужно стремиться регулярно предоставлять обновления и новости о продукте через электронные рассылки. Также нужно организовывать периодические встречи или вебинары для сбора обратной связи и обсуждения возможностей системы.

3.4 Источники дохода

ИС будет иметь следующие источники доходов:

– продажа. Команда будет получать доход от разработки и настройки информационной системы для каждой автошколы или коммерческого образовательного учреждения. Стоимость разработки для каждого клиента будет составлять примерно 20 000 рублей.

– внедрение. Будет начисляться оплата за процесс внедрения информационной системы в каждой организации. Это включает установку и настройку бота, адаптацию системы под потребности клиента. Стоимость внедрения составит около 5 000 рублей для каждой организации.

– поддержка. Нужно будет предоставлять услуги поддержки клиентам в течение нескольких месяцев после внедрения системы. Плата за поддержку составит 1 000 рублей для каждой организации ежемесячно.

– масштабируемость франшизы. Предусмотрена возможность расширения системы на другие филиалы автошколы и другие коммерческие образовательные учреждения в городе и за его пределами. За каждое новое внедрение системы предусмотрена дополнительная плата в размере 5 000 рублей за филиал.

Исходя из данных о количестве автошкол и других коммерческих образовательных организаций в городе Пенза, а согласно источнику 2ГИС их около 180, и предположении, что около 50% из них готовы приобрести информационную систему, можно оценить, что примерно 90 организаций станут будущими клиентами, а каждая из них имеет минимум 2 филиала.

Доход высчитывается по формуле

$$Д = К * Ц \quad (1)$$

где Д – доход;

К – количество потребителей;

Ц – установленная цена;

Доход от разработки и внедрения будет определяться

$$90 \text{пр} * 25000 \text{р} = 2250000 \text{р}$$

Доход от поддержки будет высчитываться

$$1000 \text{р} * 90 \text{пр} = 90000 \text{р/мес}$$

Доход от масштабируемости высчитывается

$$5000 \text{р} * 180 \text{пр} = 900000 \text{р}$$

Таким образом, общий ожидаемый доход в г. Пенза составит 3 150 000 рублей единоразово и 90 000р ежемесячно. Для наглядности на в таблице 2 представлена прогнозируемая программа производства по месяцам для коммерческих образовательных учреждений по г. Пенза.

Таблица 2 - Прогнозируемая программа производства

Наименование услуги	1 мес.	2 мес.	3 мес.	4 мес.	5 мес.
Разработка	12	28	14	16	11
Внедрение	12	28	14	16	11
Поддержка	22	64	54	54	61
Масштабируемость	10	36	40	38	50

3.5 Ключевые ресурсы

Для запуска, функционирования и развития бизнеса потребуются некоторые ключевые ресурсы. В нашем случае это будут:

- разработчики программного обеспечения, специализирующиеся на создании телеграм-ботов и информационных систем;
- менеджеры по продажам, обладающие навыками продажи и умением вести переговоры с потенциальными клиентами;
- компьютеры, серверы, программное обеспечение и другое техническое оборудование для разработки и обслуживания системы;
- офисное пространство для работы команды и проведения встреч с клиентами;
- коммуникационное оборудование для связи с клиентами, такое как телефоны и компьютеры;
- финансовые ресурсы для финансирования разработки, маркетинга, продаж и операционных расходов бизнеса.

Финансирование проекта планируется осуществить как за счет собственных средств инициатора проекта, так и за счет государственной поддержки «Гранты для молодых предпринимателей». Ниже в таблице 3 можно увидеть программу финансирования.

Таблица 3 – Источники финансирования

Источник финансирования	Сумма
Собственные средства	500 000р
Гос. поддержка	500 000р
Всего	1 000 000р

Приведенные выше ключевые ресурсы можно разделить на финансовые и людские. Подробно направление использования средств указано на таблице 4, 5.

Таблица 4 – Финансовые ключевые ресурсы

Направление использования средств	Кол-во, шт	Цена, руб.	Сумма, руб
Аренда сервера (6мес.)	6	3000	18 000
Аренда помещения (6мес.)	6	4000	24 000
Закупка ноутбуков	2	100 000	200 000
Закупка другого оборудования	1	100 000	100 000
Ком. услуги	6	5 000	30 000

Таблица 5 - Людские ключевые ресурсы

Направление использования средств	Кол-во, шт	Цена, руб.	Сумма, руб
З/п разработчику (6 мес)	12	50 000	600 000
З/п бухгалтерской компании (6 мес)	6	10 000	60 000
З/п менеджеру по продажам (6 мес)	6	30 000	180 000

3.6 Риски проекта

Реализация стартап-проекта может быть подвержена различным рискам и форс-мажорным обстоятельствам, которые могут оказать негативное влияние на его успех.

Возможными рисками являются, например, недостаточный спрос на предлагаемую информационную систему, конкуренция со стороны уже существующих аналогов, изменение законодательства в сфере автошкол, проблемы с финансированием проекта, технические проблемы при разработке и внедрении системы, а также нестабильность экономической и политической ситуации.

Форс-мажорные обстоятельства, такие как природные катастрофы, политические кризисы или пандемии, также могут оказать существенное влияние на реализацию стартап-проекта. Важно провести анализ и предусмотреть меры по минимизации возможных негативных последствий таких обстоятельств.

Заключение

Результат выполнения данной выпускной квалификационной работы можно считать успешнореализованным в соответствии с заданием информационная система для автошколы на основе мессенджера *Telegram*.

Были выполнены следующие задачи:

– анализ процессов и потребностей автошколы. Были исследованы требования и ожидания клиентов, а также особенности работы автошколы. Это позволило определить функциональность и основные модули системы;

– выбор мессенджера. После анализа различных мессенджеров было принято решение использовать *Telegram*, обусловленное его популярностью, удобством использования и наличием широкого функционала;

– выбор языка программирования. Был выбран *Python* в качестве языка программирования для разработки информационной системы. *Python* обладает простым и понятным синтаксисом, множеством библиотек и фреймворков, что облегчает разработку и ускоряет процесс создания системы;

– выбор системы управления базами данных. Была выбрана *PostgreSQL* в качестве СУБД для хранения и управления данными. *PostgreSQL* является мощной и надежной системой, обладающей широкими возможностями, включая поддержку сложных запросов и транзакций;

– выбор хостинга. Выбран сервис *TimeWeb.Cloud*, обеспечивающий надежное размещение информационной системы и обладающий преимуществами в виде высокой производительности, надежности, безопасности, гибкости настройки и доступной цены;

– разработка функционала системы. Были реализованы основные модули системы, такие как регистрация пользователей, расписание занятий, информация об инструкторах и автомобилях, онлайн-оплата и многое другое.

В результате работы была создана информационная система, которая предоставляет удобный и быстрый доступ к информации о курсах автошколы, позволяет клиентам записываться на занятия, оплачивать услуги, получать уведомления и взаимодействовать с представителями автошколы через мессенджер *Telegram*.

Перспективы развития системы включают расширение функционала, добавление новых модулей, улучшение пользовательского интерфейса и оптимизацию процессов. Разработанная информационная система на основе мессенджера *Telegram* готова к использованию в автошколе и может значительно улучшить взаимодействие с клиентами и эффективность работы автошколы.

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		95

Список используемых источников

1. Артем Демиденко. "*Telegram Bot*. Руководство по созданию бота в мессенджере Телеграм", 2023. - 23с.
2. Одинцов Б. Е. "Информационные системы управления эффективностью бизнеса", 2023. – 206с.
3. Habr.com.: [Электронный ресурс]. URL: <https://habr.com/ru/articles/697052> (дата обращения 27.04.2023)
4. Лучано Рамальо "*Python*. К вершинам мастерства", 2018. – 770с.
5. Цилюрик Олег Иванович "Сравнение языков программирования", 2018. -206с.
6. Е. Рогов. "*PostgreSQL*изнутри", 2015.– 661с.
7. Consultant.ru.: [Электронный ресурс]. URL: https://www.consultant.ru/document/cons_doc_LAW(дата обращения 5.05.2023)

Приложение А

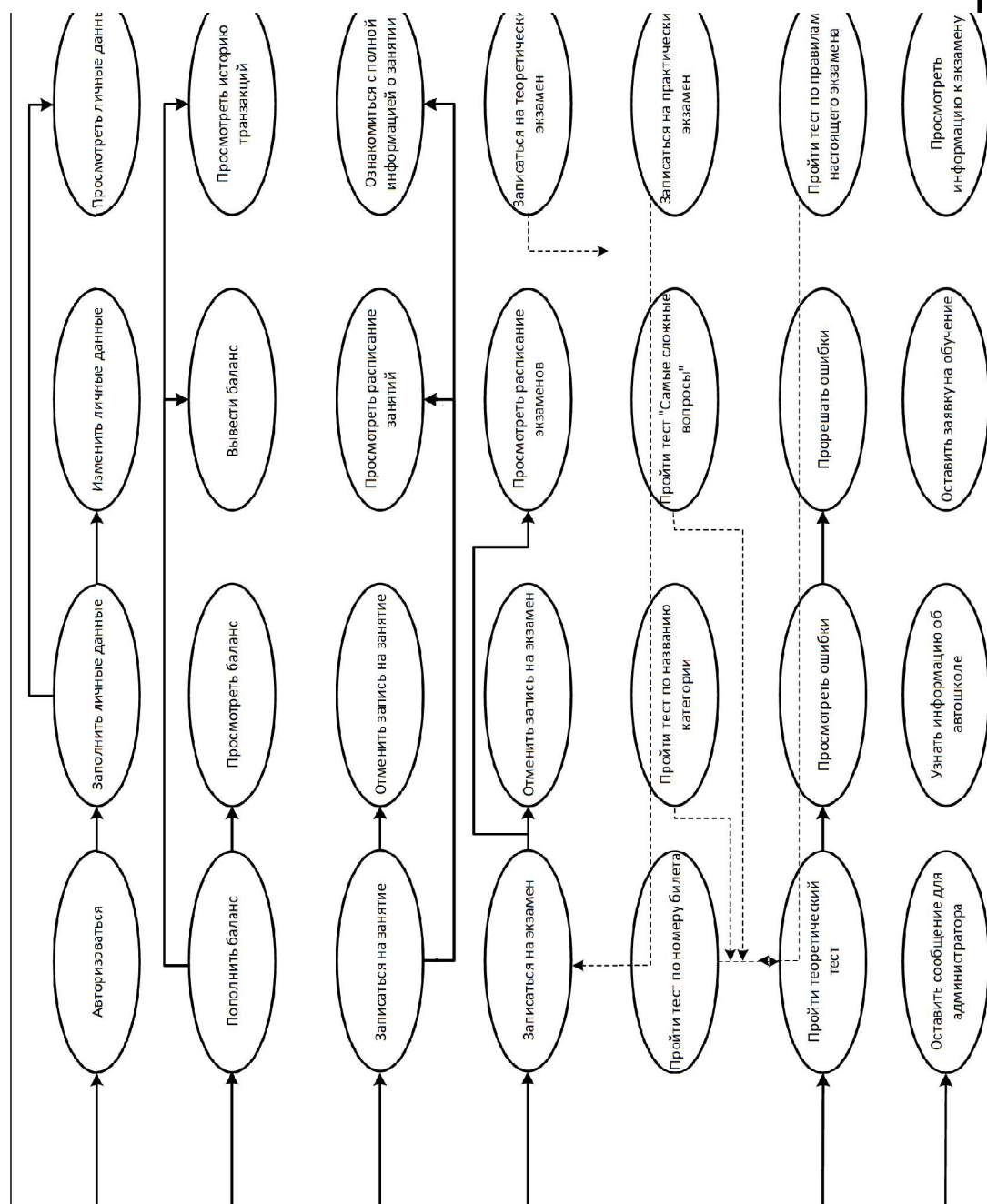
(Обязательное)

Стартап-проект информационная система «Автошкола» на основе телеграм-бота

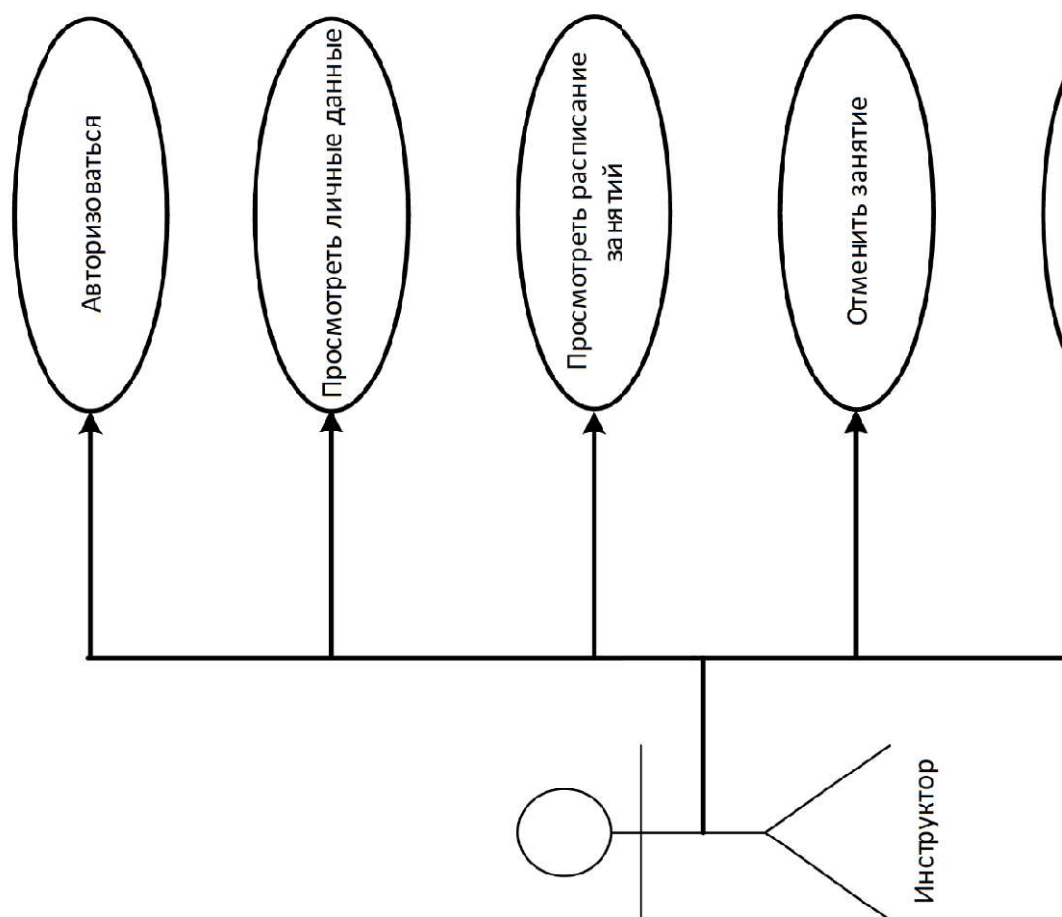
UML диаграммы

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		97

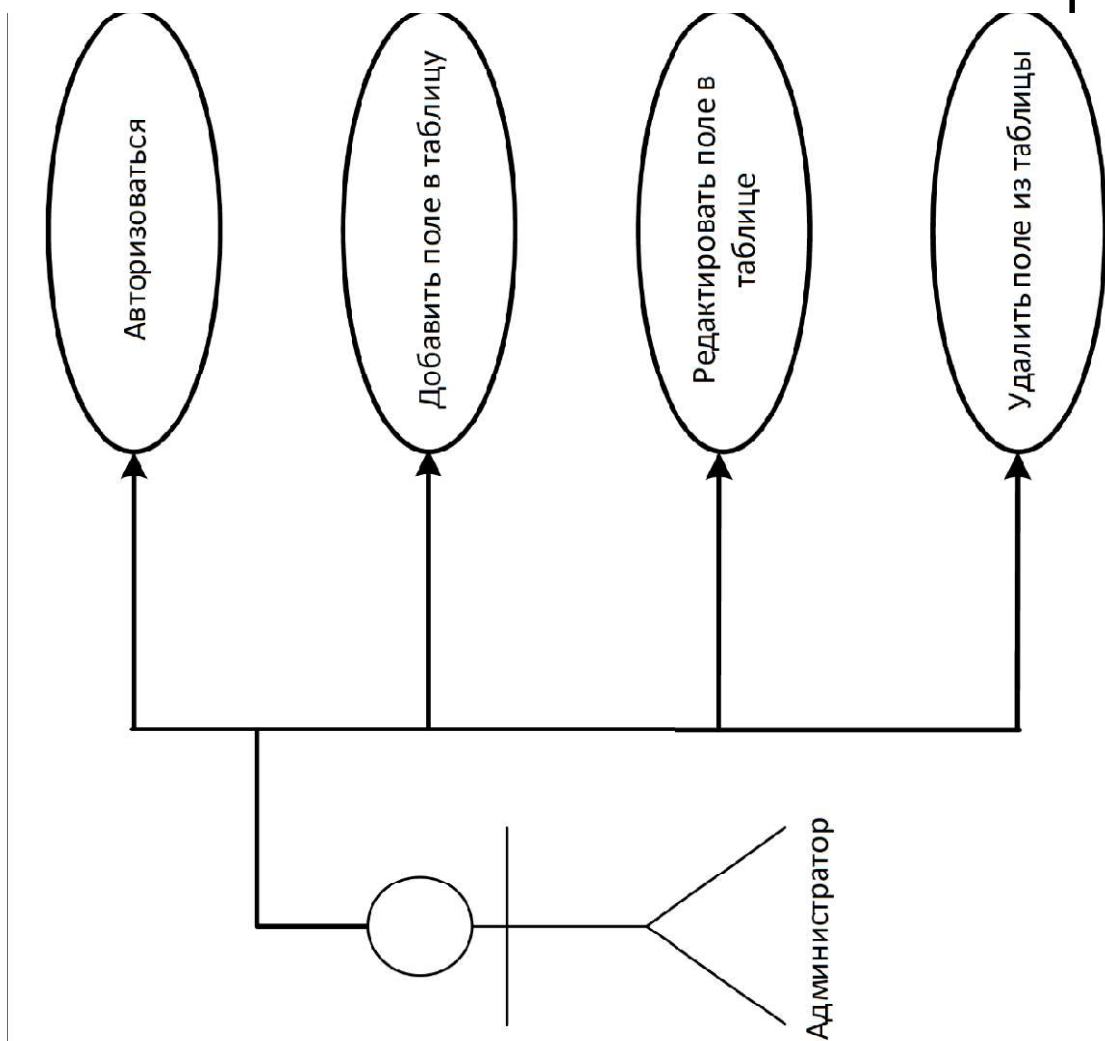
Приложение А.1 – UML-диаграмма прецедентов пользователя



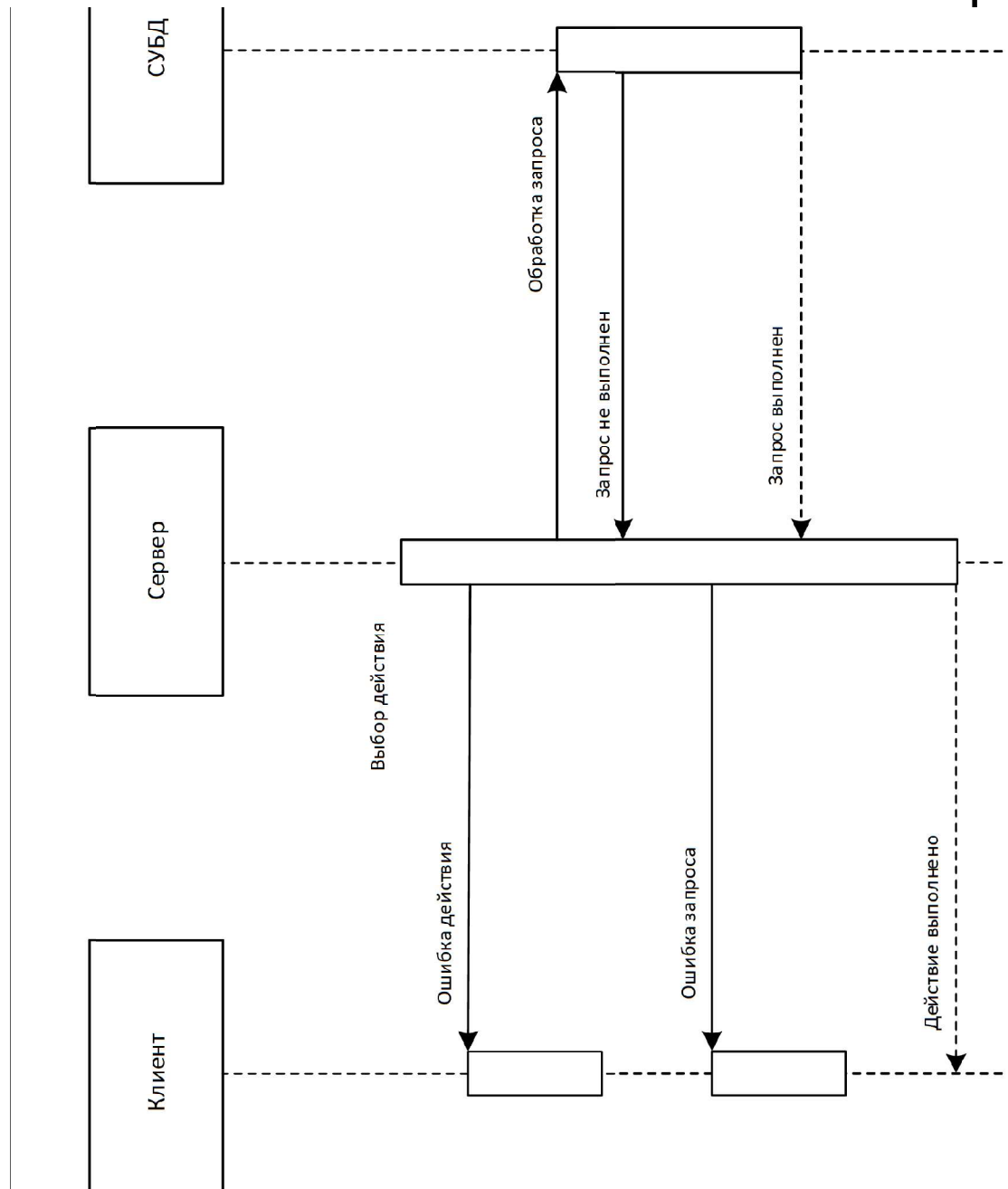
Приложение А.2 – *UML*–диаграмма прецедентов инструктора



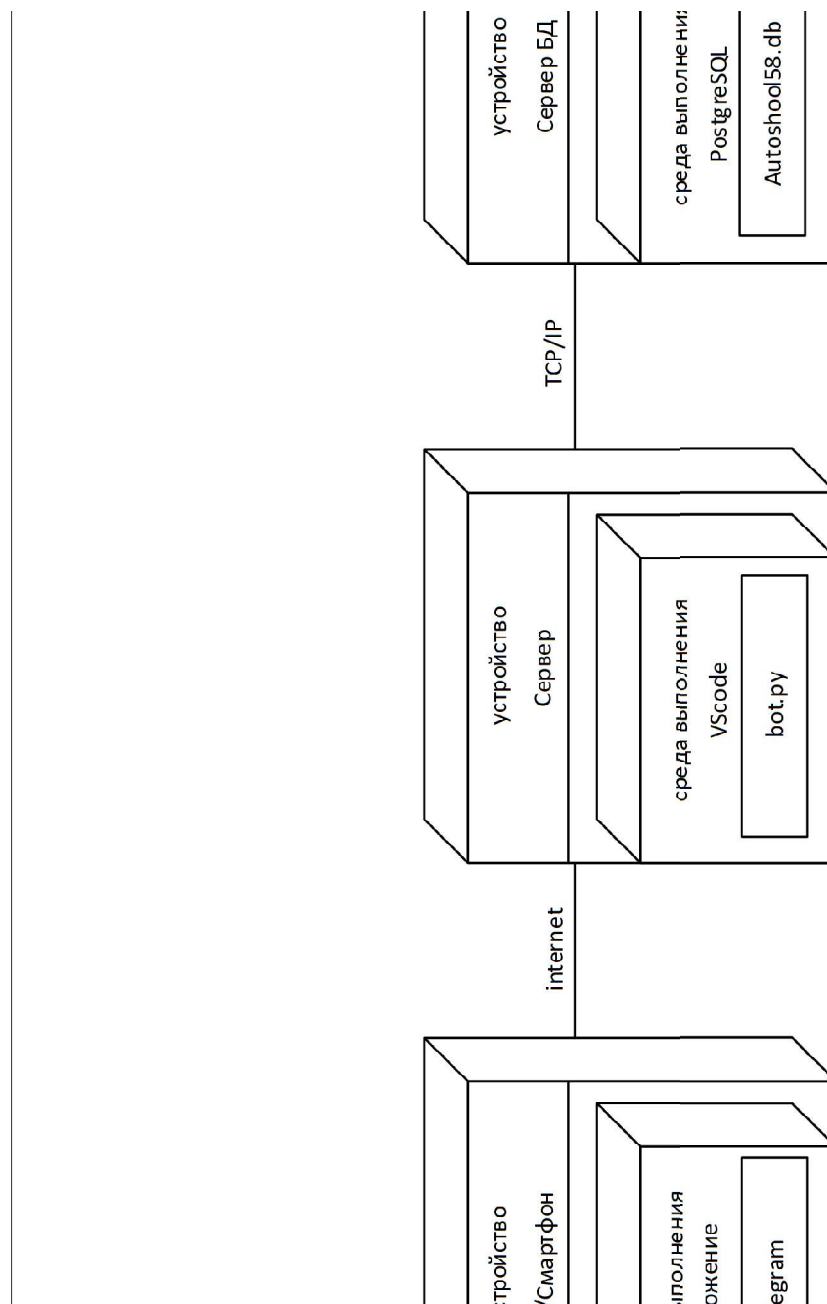
Приложение А.3 – *UML*–диаграмма прецедентов администратора



Приложение А.4 – *UML*–диаграмма последовательности



Приложение А.5 – *UML*–диаграмма развертывания



Приложение Б

(Обязательное)

Стартап-проект информационная система «Автошкола» на основе телеграм-бота

Листинг программы

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.39.001 ПЗ

Лист

102

Файл «start_bot.py»

```
from aiogram import executor
from handlers import menu, lk, rec_lesson, exam_pdd,
other, payment, edit_lesson, command_menu
from create_bot import dp

menu.register_handlers(dp)
lk.register_handlers(dp)
rec_lesson.register_handlers(dp)
payment.register_handlers(dp)
edit_lesson.register_handlers(dp)
exam_pdd.register_handlers(dp)
command_menu.register_handlers(dp)
other.register_handlers(dp)
```

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		103

```
if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```

Файл «create_bot.py»

```
from aiogram import Bot, Dispatcher, types
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from db import Database

payment_token = '401643678:TEST:c7d71cee-0944-40ea-9955-
3a04a75bc28a'
bot =
Bot(token='6132924794:AAEGdkuvbN_lOCnlThLVFHCPwvNKafdgmic')
dp = Dispatcher(bot, storage=MemoryStorage())
```

Файл «db.py»

```
import sqlite3
import psycopg2
class Database:
    def __init__(self, db):
self.connection = psycopg2.connect(

        host="localhost",
        port = "5432",
        database="Autoshool58",
        user="postgres",
        password="Qwerty58"
    )

self.cursor = self.connection.cursor()

    def check_rules_teacher(self, user_id):

        with self.connection:
self.cursor.execute("SELECT id_teacher FROM teachers WHERE
user_id = %s", (user_id,))
        result = self.cursor.fetchone()
        return result[0]

    def get_full_teacher_data(self, user_id):
        with self.connection:
self.cursor.execute(
        """
        SELECT t.id_teacher, t.user_id, t.phone_number,
t.name, t.age, t.experience, t.email,
c.transmission, c.brand, c.color, c.number
FROM teachers AS t
LEFT JOIN cars AS c
ON t.id_teacher = c.id_teacher
WHERE t.user_id = %s
        """
    )
```

```

        """
        (user_id,)
    )
    result = self.cursor.fetchone()
    return result

    def get_lessons_days(self, id_teacher, datetime, ids=False):
        with self.connection:
            coin = ">=" if ids == False else "<="
        self.cursor.execute(
            f"""
            SELECT DISTINCT ON (date) date
            FROM timetable_practic
            WHERE id_teacher = {id_teacher} AND date {coin}
            '{datetime}'
            ORDER BY date ASC
            """
        )
        result = self.cursor.fetchall()
        return result

    def get_lesson_on_day(self, id_teacher, datetime):
        with self.connection:
        self.cursor.execute(
            """
            SELECT id_practic, date, time, id_student
            FROM timetable_practic
            WHERE id_teacher = %s AND date = %s
            """,
            (id_teacher, datetime)
        )
        result = self.cursor.fetchall()
        return result

    def check_rul_examen(self, user_id, name_exam):
        if name_exam == 0:
            with self.connection:
        self.cursor.execute(
            """
            SELECT
            (SELECT count_lessons FROM students WHERE
            user_id = %s) >=
            (SELECT practic_count
            FROM students s
            INNER JOIN groups g ON s.id_group = g.id_group
            INNER JOIN program p ON g.id_program =
            p.id_program
            WHERE s.user_id = %s) as result""",
            (user_id, user_id)
        )
        result = self.cursor.fetchone()
        return result

```

```

def del_lesson_on_day(self,id_teacher,datetime):

    with self.connection:

        return self.cursor.execute("DELETE FROM
timetable_practic WHERE date = %s AND id_teacher
=%s", (datetime,id_teacher))

def update_count_lesson(self,id):
    with self.connection:

        return self.cursor.execute("UPDATE students SET
count_lessons = count_lessons + 1 WHERE id_student = %s", (id,))

def get_info_lesson(self,id):
    with self.connection:
lst = []
self.cursor.execute(
    """
        SELECT tp.date, tp.time, s.name, s.phone_number,
s.email, c.adres, c.room, s.id_student
        FROM timetable_practictp
        JOIN students s ON tp.id_student = s.id_student
        JOIN class c ON tp.id_class = c.id_class
        WHERE tp.id_practic = %s
    """,
    (id,)
)

    result = self.cursor.fetchone()
    return result

def add_person(self,user_id,phone,category):
    with self.connection:
self.cursor.execute("INSERT INTO applications (user_id, phone,
category) SELECT %s, %s, %s WHERE NOT EXISTS (SELECT 1 FROM
applications WHERE user_id = %s AND phone = %s)", (user_id,
phone, category, user_id, phone))
        if self.cursor.rowcount == 0:
            raise ValueError("Person with the same user_id
and phone already exists")
        def add_examday(self,id_lesson, date_exam,count_students,
name_inspector):
            with self.connection:
                return self.cursor.execute("INSERT INTO gibdd_exam
(id_lesson, date_exam,count_students, name_inspector) VALUES

```

```

( %s, %s,%s, %s)", (id_lesson, date_exam,count_students,
name_inspector, ))

def add_user(self, phone,group):
    with self.connection:
        return self.cursor.execute("INSERT INTO students
(phone_number, id_group) VALUES (%s, %s)", (phone,group ))
    def add_group(self, count_students, id_teacher,id_program):
        with self.connection:
            return self.cursor.execute("INSERT INTO groups
(count_students, id_teacher,id_program) VALUES (%s, %s,%s)",
(count_students, id_teacher,id_program ))

def add_table(self, day_week,
time,id_group,id_teacher,id_lesson,id_class):
    with self.connection:
        return self.cursor.execute("INSERT INTO
timetable_tory (day_week,
time,id_group,id_teacher,id_lesson,id_class) VALUES (%s,
%s,%s,%s, %s,%s)", (day_week,
time,id_group,id_teacher,id_lesson,id_class, ))

def add_category(self, name):
    with self.connection:
        return self.cursor.execute("INSERT INTO category
(category_name) VALUES (%s)", (name,))
    def add_programm(self, programe_name, price, practic_count,
teory_count, id_group, id_category,marshrut_number):
        with self.connection:
            return self.cursor.execute("INSERT INTO program
(programe_name, price, practic_count, teory_count, id_group,
id_category,marshrut_number) VALUES (%s, %s,%s, %s,%s, %s,%s)",
(programe_name, price, practic_count, teory_count, id_group,
id_category,marshrut_number, ))

def add_car(self, transmisson, brand, color, number,
id_teacher):
    with self.connection:
        return self.cursor.execute("INSERT INTO cars
(transmisson, brand, color, number, id_teacher) VALUES (%s,
%s,%s, %s,%s)", (transmisson, brand, color, number, id_teacher,
))

def add_class(self, adres, room):
    with self.connection:
        return self.cursor.execute("INSERT INTO class
(adres, room) VALUES (%s, %s)", (adres, room, ))

def add_teacher(self, user_id, phone_number, name, age,
experiance, email):
    with self.connection:

```

```

        return self.cursor.execute("INSERT INTO teachers
(user_id, phone_number, name, age, experiance, email) VALUES
(%s, %s,%s, %s,%s, %s)", (user_id, phone_number, name, age,
experiance, email, ))

    def del_row(self, name, name_column,id):
        with self.connection:
            return self.cursor.execute("DELETE FROM %s WHERE %s
= %s" % (name, name_column, id,))

    def info_column(self,table_name):
        with self.connection:
self.cursor.execute("""
        SELECT column_name
        FROM information_schema.columns
        WHERE table_name = %s
        AND column_name NOT IN (
            SELECT column_name
            FROM information_schema.key_column_usage
            WHERE table_name = %s
        )
        AND column_name NOT IN (
            SELECT conname
            FROM pg_constraint
            WHERE conrelid = %s::regclass
            AND contype IN ('f', 'p')
        );
        """, (table_name, table_name, table_name))
        result = self.cursor.fetchall()
        columns = [row[0] for row in result]
        return columns

    def user_exists(self, user_id):
        with self.connection:
self.cursor.execute("SELECT * FROM students WHERE user_id = %s",
(user_id,))
        result = self.cursor.fetchall()
        return bool(len(result))

    def phone_exists(self,phone):
        with self.connection:
self.cursor.execute("SELECT * FROM students WHERE phone_number =
%s", (phone,))
        result = self.cursor.fetchall()
        # result = self.cursor.execute (f"SELECT * FROM
students WHERE phone_number = '{phone}'").fetchall()
        return bool(len(result))

    def get_full_data(self,user_id,ids = False):
        if ids == False:

```

```

        with self.connection:
self.cursor.execute("SELECT
user_id,phone_number,name,pasport,medical,email,age,id_group,bal
ance FROM students WHERE user_id = %s", (user_id,))
        result = self.cursor.fetchall()
        #result = self.cursor.execute
(f"SELECTuser_id,phone_number,name,pasport,medical,email,age,id_
group FROM students WHERE user_id = '{user_id}'").fetchall()
        return result[0]
elif ids == True:

self.cursor.execute("SELECT
user_id,phone_number,name,pasport,medical,email,age,id_group,bal
ance FROM students WHERE id_student = %s", (user_id,))
        result = self.cursor.fetchall()
        #result = self.cursor.execute
(f"SELECTuser_id,phone_number,name,pasport,medical,email,age,id_
group FROM students WHERE user_id = '{user_id}'").fetchall()
        return result[0]

    def get_full(self,user_id): #Проверка заполненыливседаанные.
        with self.connection:
self.cursor.execute("SELECT full_reg FROM students WHERE user_id
= %s", (user_id,))
        #result = self.cursor.execute (f"SELECTfull_reg FROM
students WHERE user_id = '{user_id}'")
        result = self.cursor.fetchone()
        return result[0]
    def get_email(self, user_id):
        with self.connection:
self.cursor.execute("SELECT email FROM students WHERE user_id =
%s", (user_id,))
        result = self.cursor.fetchone()
        return result[0]

    def set_signup(self, user_id, signup):
        with self.connection:
            return self.cursor.execute("UPDATE 'students' SET
'signup' = ? WHERE 'user_id' = ?", (signup, user_id))

    def set_name(self,user_id, name):
        with self.connection:
            return self.cursor.execute("UPDATE students SET name
= %s WHERE user_id = %s", (name, user_id))

    def set_pasport(self,user_id, pasport):
        with self.connection:
            return self.cursor.execute("UPDATE students SET
pasport = %s WHERE user_id = %s", (pasport, user_id))

    def set_medical(self,user_id, medical):
        with self.connection:

```

```

        return self.cursor.execute("UPDATE students SET
medical = %s WHERE user_id = %s", (medical, user_id))

    def set_email(self, user_id, email):
        with self.connection:
            return self.cursor.execute("UPDATE students SET email
= %s WHERE user_id = %s", (email, user_id))

    def set_age(self, user_id, age):
        with self.connection:
            return self.cursor.execute("UPDATE students SET age =
%s WHERE user_id = %s", (age, user_id))

    def get_data(self, user_id):
        with self.connection:
            self.cursor.execute("SELECT name, passport, medical, email, age FROM
students WHERE user_id = %s", (user_id,))
            result = self.cursor.fetchall()
            return result[0]

    def get_data_of_payment(self, user_id):
        with self.connection:
            self.cursor.execute("SELECT phone_number, name, balance FROM
students WHERE user_id = %s", (user_id,))
            result = self.cursor.fetchall()
            return result[0]

    def set_full(self, user_id):
        with self.connection:
            return self.cursor.execute("UPDATE students SET
full_reg = %s WHERE user_id = %s", (True, user_id))

    def check_table(self, user_id, month, day, year = "2023"):
        data_str = f"{year}-{month.rjust(2, '0')}-{day.rjust(2, '0')}"
        with self.connection:
            self.cursor.execute("SELECT tp.time FROM students s JOIN groups
g ON s.id_group = g.id_group JOIN timetable_practictp ON
g.id_teacher = tp.id_teacher WHERE s.user_id = %s AND tp.date =
%s", (user_id, data_str, ))

            result = self.cursor.fetchall()
            return result

    def rec_lesson(self, user_id, date, time):
        with self.connection:
            self.cursor.execute("INSERT INTO timetable_practic (date, time,
id_teacher, id_lesson, id_student) SELECT %s, %s,
groups.id_teacher, %s, students.id_student FROM students JOIN
groups ON students.id_group = groups.id_group WHERE
students.user_id = %s", (date, time, 1, user_id, ))

```

```

def rec_exam(self, user_id, date, id_lesson):
    with self.connection:
self.cursor.execute("INSERT INTO students_of_exam (id_exam,
id_student) SELECT (SELECT id_exam FROM gibdd_exam WHERE
date_exam = %s AND id_lesson = %s), id_student FROM students
WHERE user_id = %s", (date, id_lesson, user_id,))

def get_lessons_pr(self, user_id, date, id_lesson = 1):
    with self.connection:
self.cursor.execute("SELECT tp.date, tp.time,
tp.id_teacher, tp.id_class, tp.id_lesson, tp.id_practictp FROM
timetable_practictp JOIN students s ON tp.id_student =
s.id_student WHERE s.user_id = %s AND tp.id_lesson = %s AND
tp.date >= %s", (user_id, id_lesson, date))
        result = self.cursor.fetchall()
        return result

def get_lessons_tr(self, user_id, id_lesson = 2):
    with self.connection:
self.cursor.execute("""
        SELECT t.day_week, t.time, t.id_teacher,
t.id_class, t.id_lesson, t.id_tory
        FROM timetable_tory t
        JOIN students s ON t.id_group = s.id_group
        WHERE s.user_id = %s AND t.id_lesson = %s
        """, (user_id, id_lesson))
        result = self.cursor.fetchone()
        return result

def get_info(self, id_teacher, id_class):
    with self.connection:
self.cursor.execute("SELECT name, phone_number, email FROM
teachers WHERE id_teacher = %s", (id_teacher,))
info_teacher = self.cursor.fetchone()
self.cursor.execute("SELECT adres, room FROM class WHERE
id_class = %s", (id_class,))
info_class = self.cursor.fetchone()
        return info_teacher, info_class

def get_exams_student(self, user_id):
    with self.connection:
self.cursor.execute("SELECT gibdd_exam.id_lesson,
gibdd_exam.date_exam, gibdd_exam.name_inspector,
students_of_exam.id_rec_exam FROM gibdd_exam JOIN
students_of_exam ON gibdd_exam.id_exam =
students_of_exam.id_exam JOIN students ON
students_of_exam.id_student = students.id_student WHERE
gibdd_exam.date_exam >= NOW() AND students.user_id = %s",
(user_id,))

```

```

arr = self.cursor.fetchall()
    return arr

def del_exam(self,id):
    with self.connection:
        return self.cursor.execute("DELETE FROM
students_of_exam WHERE id_rec_exam = %s", (id,))

def del_lesson(self, id,ids=False):
    if ids == False:
        with self.connection:
            return self.cursor.execute("DELETE FROM
timetable_practic WHERE id_practic = %s", (id,))

elif ids == True:
    with self.connection:
self.cursor.execute("SELECT id_student FROM timetable_practic
WHERE id_practic = %s", (id,))
id_student = self.cursor.fetchone()[0]
self.cursor.execute("DELETE FROM timetable_practic WHERE
id_practic = %s", (id,))
        return id_student

def show_date_exam(self,id_lesson,date):
    with self.connection:
self.cursor.execute("SELECT id_exam, date_exam ,count_students
FROM gibdd_exam WHERE id_lesson = %s AND date_exam> %s ",
(id_lesson,date,))
        result = self.cursor.fetchall()
        return result

def show_count_slots(self,id_exam):
    with self.connection:
self.cursor.execute("SELECT id_student FROM students_of_exam
WHERE id_exam = %s", (id_exam,))

        result = self.cursor.fetchall()
        return result

def get_payment(self,user_id):
    with self.connection:
self.cursor.execute("SELECT p.date, p.time, p.amount, p.source
FROM payment p JOIN students s ON s.id_student = p.id_student
WHERE s.user_id = %s", (user_id,))
        result = self.cursor.fetchall()
        return result

```

```

def set_payment(self, user_id,
date, time, amount, currency, charge_id, source):
    with self.connection:
self.cursor.execute("INSERT INTO payment (id_student, date,
time, amount, currency, provider_payment_charge_id, source)
SELECT s.id_student, %s, %s, %s, %s, %s, %s FROM students s
WHERE s.user_id = %s RETURNING id",
(date, time, amount, currency, charge_id, source, user_id))
        result = self.cursor.fetchone()
        return result[0]
def update_balance(self, user_id, amount, ids = False):
    if ids == False:
        with self.connection:
self.cursor.execute("UPDATE students SET balance = balance + %s
WHERE id_student = (SELECT id_student FROM students WHERE
user_id = %s);", (amount, user_id))
    elif ids == True:
        with self.connection:
self.cursor.execute("UPDATE students SET balance = balance + %s
WHERE id_student = %s;", (amount, user_id))

def get_answer_exam(self, questions):
    if isinstance(questions, list):
        ids = tuple(questions)
    else:
        ids = questions
    with self.connection:
self.cursor.execute("SELECT path_to_image, count_answ,
right_answ, text, anotation, id_exam_quest FROM exam_pdd WHERE
id_exam_quest IN %s", (ids,))
        result = self.cursor.fetchall()
        return result
def get_gustions(self, id_quiz):
    with self.connection:
self.cursor.execute("SELECT array_q FROM result_quiz WHERE id =
%s", (id_quiz,))
        result = self.cursor.fetchone()
        return result

def get_gustions_error(self, user_id):
    with self.connection:
self.cursor.execute("SELECT errors FROM result_quiz WHERE
id_student = (SELECT id_student FROM students WHERE user_id =
%s) ORDER BY id DESC LIMIT 1", (user_id,))
        result = self.cursor.fetchone()
        return result[0]

def get_answers_hard(self,):
    with self.connection:

```

```

self.cursor.execute("SELECT path_to_image, count_answ,
right_answ, text, anotation, id_exam_quest FROM exam_pdd WHERE
hard = %s", (True,))
        result = self.cursor.fetchall()
        return result

    def get_answers_category(self, category):
        with self.connection:
self.cursor.execute("SELECT path_to_image, count_answ,
right_answ, text, anotation, id_exam_quest FROM exam_pdd WHERE
notion LIKE %s", ("% " + category + "%",))
        result = self.cursor.fetchall()
        return result

    def get_answers_bilet(self, bilet):
        with self.connection:
self.cursor.execute("SELECT path_to_image, count_answ,
right_answ, text, anotation, id_exam_quest FROM exam_pdd WHERE
bilet = %s", (bilet,))
        result = self.cursor.fetchall()
        return result

    def start_quiz(self, user_id, time, mode):
        with self.connection:
self.cursor.execute("INSERT INTO result_quiz(id_student,
time_start, mode, result) VALUES ((SELECT id_student FROM
students WHERE user_id = %s), %s, %s, %s) RETURNING id",
        (user_id, time, mode, 0))
        result = self.cursor.fetchone()
        return result[0]

    def set_questions_quiz(self, array, id_quiz):
        with self.connection:

self.cursor.execute("UPDATE result_quiz SET array_q = %s WHERE
id = %s", (array, id_quiz))

    def get_quiz_result(self, id_quiz):
        with self.connection:
self.cursor.execute("SELECT result FROM result_quiz WHERE id =
%s", (id_quiz,))
        result = self.cursor.fetchone()
        return result[0]

    def right_answer(self, id_quiz):
        with self.connection:
self.cursor.execute("UPDATE result_quiz SET result = result + 1
WHERE id = %s", (id_quiz,))
        def wrong_answer(self, id_answer, id_quiz):
            with self.connection:

```

```
self.cursor.execute("UPDATE result_quiz SET errors =
CONCAT(errors, '%s, ' ) WHERE id = %s", (id_answer, id_quiz,))
```

Файл «меню.py»

```
from db import Database
from aiogram import types, Dispatcher
from aiogram.dispatcher.filters import Text
from create_bot import dp, bot, db

async def cmd_menu(message: types.Message):
    await bot.delete_message(message.chat.id,
message.message_id-1)
    await bot.delete_message(message.chat.id,
message.message_id)
    if ( db.user_exists(message.from_user.id)):
buttons_menu = types.ReplyKeyboardMarkup()
        button_1 = types.KeyboardButton(text="👤 Личный кабинет")
        button_2 = '🚗 Записаться на вождение'
button_3 = '📅 Управление расписанием'
button_4 = '💰 Управление балансом'
button_5 = '✅ Пройти экзамен ПДД'

button_7 = '? Информация.'
buttons_menu.add(button_1, button_2, button_3, button_4, button_5,
button_7)
        await message.answer("{0.first_name}, Вынаходитесь в
главном меню.".format(message.from_user),
reply_markup=buttons_menu)
    else:
await message.answer("Вы не авторизованы. Воспользуйтесь командой
/start.", reply_markup=types.ReplyKeyboardRemove())

#@dp.message_handler(commands=['start']) #Явно указываем в
декораторе, на какую команду реагируем.
async def cmd_start(message: types.Message):
    await bot.delete_message(message.chat.id,
message.message_id)
    if (not db.user_exists(message.from_user.id)):

        keyboard = types.ReplyKeyboardMarkup()
button_phone =
types.KeyboardButton(text="Отправить номер телефона",
request_contact=True)
cancel_phone = types.KeyboardButton(text="? Информация.")
keyboard.add(button_phone, cancel_phone)
        await
message.answer("Для авторизации нам нужен Ваш номер телефона.", reply_m
arkup=keyboard)
    else:
```

```

        await bot.send_message(message.from_user.id,
'Выужезарегистрированы в системе. Для перехода в меню
воспользуйтесь командой /menu') #message.answer(text="Вы уже
авторизованы.", show_alert=True)

#@dp.message_handler(content_types=types.ContentType.CONTACT)
#Еще нужна обработка отмены отправки контакта.
async def get_auto(message: types.Message):
    phone = message.contact.phone_number.replace("+", "")
    await bot.delete_message(message.chat.id,
message.message_id-1)
    await bot.delete_message(message.chat.id,
message.message_id)
    if db.phone_exists(message.contact.phone_number.replace("+",
"")):

db.set_user_id(message.contact.user_id,message.contact.phone_num
ber.replace("+", ""))

        awaitmessage.answer(f"Твой номер успешно авторизован.
Для дальнейшей регистрации заполни данные в личном кабинете.
Используй команду /menu.",
reply_markup=types.ReplyKeyboardRemove())
else:
keyboard_lk = types.InlineKeyboardMarkup(row_width=1)
lk_b1 =
types.InlineKeyboardButton(text="Оставитьзаявкунаобучение",
callback_data=f"sign_{phone}")
keyboard_lk.add(lk_b1)
        awaitmessage.answer("К сожалению Вас нет в списках
студентов. Если вы хотели бы оставить заявку на обучение нажмите
на кнопку", reply_markup=keyboard_lk)

async def sign(callback: types.CallbackQuery):
    await callback.message.delete()
    phone = callback.data.split('_')[-1]
    keyboard_lk = types.InlineKeyboardMarkup(row_width=1)
    b1 = types.InlineKeyboardButton(text="B",
callback_data=f"s_{phone}_B")
    b2 = types.InlineKeyboardButton(text="B,C",
callback_data=f"s_{phone}_B,C")
    b3 = types.InlineKeyboardButton(text="A",
callback_data=f"s_{phone}_A")
    b4 = types.InlineKeyboardButton(text="Другое",
callback_data=f"s_{phone}_Другое")
    keyboard_lk.add(b1,b2,b3,b4)
    await callback.message.answer ('Выберитежелаемуюкатегорию',
reply_markup= keyboard_lk)

async def bd_add(callback: types.CallbackQuery):
    await callback.message.delete()

```

```

    try:
        keyboard_lk = types.InlineKeyboardMarkup(row_width=1)
        b1 = types.InlineKeyboardButton(text="Выход",
        callback_data=f"del")
        keyboard_lk.add(b1)

        db.add_person(callback.from_user.id, callback.data.split('_')[-
        2], callback.data.split('_')[-1])
        await callback.message.answer
        ('Ваша заявка принята.', reply_markup= keyboard_lk)
    except:
        await callback.message.answer ('Что-то пошло не так. Возможно
        ваша заявка уже в рассмотрении.\n\n Если вы еще не оставляли
        заявку или хотели бы уточнить условия, свяжитесь с поддержкой
        88412359046', reply_markup= keyboard_lk)

    async def delete(callback: types.CallbackQuery):

        await callback.message.delete()
    async def info_menu(message: types.Message):

        await bot.delete_message(message.chat.id,
        message.message_id-1) if message.text == "?Информация." else
        None
        await bot.delete_message(message.chat.id,
        message.message_id)

    map_url = "https://yandex.ru/maps/-/CCUoi6s8DA"
    keyboard_rec = types.InlineKeyboardMarkup()
    b1 = types.InlineKeyboardButton(text="Показать на карте",
    url=map_url) #types.InlineKeyboardButton(text="Авторизоваться",
    callback_data="log in")
    keyboard_rec.add(b1)

```

Файл «lk.py»

```

from db import Database
from aiogram import types, Dispatcher
from aiogram.types import InputFile
from aiogram.dispatcher.filters import Text
from create_bot import db, dp, bot
from typing import Union
from aiogram.dispatcher import FSMContext
from aiogram.dispatcher.filters.state import State, StatesGroup

first_name = "User"
balance = 0

class UserState(StatesGroup):
    name = State()

```

```

passport = State()
medical = State()
email = State()
age = State()
name_edit = State()
passport_edit = State()
medical_edit = State()
email_edit = State()
age_edit = State()

#@dp.callback_query_handler(lambda callback: callback.data in
["back_to_lk"])
async def with_puree(callback: types.CallbackQuery, state:
FSMContext):
    await callback.message.delete()
keyboard_lk = types.InlineKeyboardMarkup(row_width=1)
    lk_b1 = types.InlineKeyboardButton(text="Заполнить данные",
callback_data="log
in") #types.InlineKeyboardButton(text="Авторизоваться",
callback_data="log in")
    lk_b2 = types.InlineKeyboardButton(text="Просмотреть данные",
callback_data="view_data")
    lk_b3 = types.InlineKeyboardButton(text="Изменить данные",
callback_data="edit_data")
    lk_b4 = types.InlineKeyboardButton(text="Просмотр маршрута",
callback_data="view_marshrut")

keyboard_lk.add(lk_b1, lk_b2, lk_b3, lk_b4) if
db.get_full(callback.from_user.id) == False else
keyboard_lk.add(lk_b2, lk_b3, lk_b4)

    await callback.message.answer(f'👤 Личный кабинет 👤: \n Никнейм
{first_name}! \n Статус:
{db.get_full(callback.from_user.id)} \n Баланс:
{balance} \n \n Чтобы выйти в главное меню введите команду /menu
или нажмите на нее!',
reply_markup=keyboard_lk) #reply_markup=types.ReplyKeyboardRemove
()

# from aiogram.dispatcher.filters import Text
#@dp.message_handler(Text(equals="👤 Личный кабинет"))
async def lk(message: Union[types.Message, types.CallbackQuery],
state: FSMContext, previous_state=None):
    await bot.delete_message(message.chat.id,
message.message_id-1)
    await bot.delete_message(message.chat.id,
message.message_id)

keyboard_lk = types.InlineKeyboardMarkup(row_width=1)

```

```

lk_b1 = types.InlineKeyboardButton(text="Заполнить данные",
callback_data="log
in")#types.InlineKeyboardButton(text="Авторизоваться",
callback_data="log in")
lk_b2 = types.InlineKeyboardButton(text="Просмотреть данные",
callback_data="view_data")
lk_b3 = types.InlineKeyboardButton(text="Изменить данные",
callback_data="edit_data")
lk_b4 = types.InlineKeyboardButton(text="Просмотр маршрута",
callback_data="view_marshrut")

keyboard_lk.add(lk_b1,lk_b2,lk_b3,lk_b4) if
db.get_full(message.from_user.id) == False else
keyboard_lk.add(lk_b2,lk_b3,lk_b4)

```

```

await message.answer(f'👤 Личный кабинет 👤: \n Никнейм
{first_name}!\n Статус:
{db.get_full(message.from_user.id)}\n Баланс:
{balance}\n \n Чтобы выйти в главное меню введите команду /menu
или нажми те на нее!',
reply_markup=keyboard_lk)#reply_markup=types.ReplyKeyboardRemove
()

```

```

#@dp.callback_query_handler(lambda callback: callback.data in
["log in", "view_data"])
async def auto(callback: types.CallbackQuery, state:
FSMContext):
    if callback.data == "log in":
        await callback.message.delete()
    lst_data = db.get_data(callback.from_user.id)
    keyboard_reg = types.InlineKeyboardMarkup()
    reg_b1 = types.InlineKeyboardButton(text="Указать ФИО",
callback_data="set_name")#types.InlineKeyboardButton(text="Автор
изоваться", callback_data="log in")
    reg_b2 =
types.InlineKeyboardButton(text="Указать паспорт",
callback_data="set_pasport")
    reg_b3 =
types.InlineKeyboardButton(text="Указать медсправку",
callback_data="set_medical")
    reg_b4 = types.InlineKeyboardButton(text="Указать эл.
почту", callback_data="set_email")
    reg_b5 =
types.InlineKeyboardButton(text="Указать дату рождения",
callback_data="set_age")
    reg_b6 = types.InlineKeyboardButton(text="Назад",
callback_data="back_to_lk")
    lst_key = [reg_b1,reg_b2,reg_b3,reg_b4,reg_b5]
    for i in range(len(lst_data)-1,-1,-1):
        if lst_data[i] == None or lst_data[i] == "":
            pass

```

```

        else:
            lst_key.pop(i)
            if len(lst_key) != 0:
                keyboard_reg.add(*lst_key, reg_b6)
                await
            callback.message.answer('Какие данные вы хотите заполнить?', reply_markup=keyboard_reg)
        else:
            db.set_full(callback.from_user.id)
            await callback.message.answer('Вы заполнили все поля. Для перехода в главное меню воспользуйтесь командой /menu', reply_markup=keyboard_reg)

    if callback.data == "edit_data":
        await callback.message.delete()
        lst_data = db.get_data(callback.from_user.id)
        keyboard_reg = types.InlineKeyboardMarkup()
        reg_b1 = types.InlineKeyboardButton(text="Изменить ФИО", callback_data="edit_name")
        #types.InlineKeyboardButton(text="Автоматизироваться", callback_data="log in")
        reg_b2 = types.InlineKeyboardButton(text="Изменить паспорт", callback_data="edit_pasport")
        reg_b3 = types.InlineKeyboardButton(text="Изменить медсправку", callback_data="edit_medical")
        reg_b4 = types.InlineKeyboardButton(text="Изменить эл. почту", callback_data="edit_email")
        reg_b5 = types.InlineKeyboardButton(text="Изменить дату рождения", callback_data="edit_age")
        reg_b6 = types.InlineKeyboardButton(text="Назад", callback_data="back_to_1k")
        lst_key = [reg_b1, reg_b2, reg_b3, reg_b4, reg_b5]
        for i in range(len(lst_data)-1, -1, -1):
            if lst_data[i] != None or lst_data[i] != "":
                pass
            else:
                lst_key.pop(i)
                if len(lst_key) != 0:
                    keyboard_reg.add(*lst_key, reg_b6)
                    await
                callback.message.answer('Какие данные вы хотите изменить?', reply_markup=keyboard_reg)
            else:
                await callback.message.answer('Вы еще не заполнили ни одного поля доступного для изменения', reply_markup=keyboard_reg)

    if callback.data == "view_data":
        await callback.message.delete()
        lst_full_data = db.get_full_data(callback.from_user.id)

```

```

keyboard_reg = types.InlineKeyboardMarkup()
    reg_b1 = types.InlineKeyboardButton(text="Назад",
callback_data="back_to_lk")#types.InlineKeyboardButton(text="Авт
оризоваться", callback_data="log in")

keyboard_reg.add(reg_b1)
    await callback.message.answer(f'Ваши данные:\nЛогин
{lst_full_data[0]}\nТелефон +{lst_full_data[1]}\nИмя
{lst_full_data[2]}\nПаспортные данные
{lst_full_data[3]}\nМедицинская справка
{lst_full_data[4]}\nЭлектронный адрес
{lst_full_data[5]}\nДата рождения {lst_full_data[6]}\nГруппа
{lst_full_data[7]}\n',reply_markup=keyboard_reg)

    if callback.data == "view_marshrut":
        await state.update_data(previous_state="lk")
        await callback.message.delete()
keyboard_reg = types.InlineKeyboardMarkup()
    b1 = types.InlineKeyboardButton(text="Назад",
callback_data="back_to_lk")
keyboard_reg.add(b1)
id_marshrut = db.get_marshrut(callback.from_user.id)

    await
callback.message.answer_photo(photo=open(f'marshrut/{id_marshrut
}.jpg', 'rb'),reply_markup= keyboard_reg)

elif callback.data == "back_to_lk":
# получаем предыдущее состояние и вызываем функцию lk с этим
состоянием
data = await state.get_data()
previous_state = data.get("previous_state")
    await lk(callback.message, state, previous_state)

#@dp.callback_query_handler(text="set_name")
async def set_data(call: types.CallbackQuery, state:
FSMContext):
    await call.message.delete()
    if call.data == "set_name":
        question = "Введите ваше ФИО."
        await UserState.name.set()
    elif call.data == "set_pasport":
        question = "Введите ваше серию и номер паспорта в
формате: 1234 56789123."
        await UserState.passport.set()
    elif call.data == "set_medical":
        question = "Введите номер вашей медсправки в формате: 00
1234567."
        await UserState.medical.set()
    elif call.data == "set_email":

```

```

        question = "Введите ваш электронный адрес."
await UserState.email.set()
elifcall.data == "set_age":
    question = "Введите вашу дату рождения в формате:
01.01.1900"
await UserState.age.set()

    await call.message.answer(question)

async def set_personal_data(message: types.Message, state:
FSMContext):
    # state_key = state.key
state_key = await state.get_state()
if state_key == UserState.name.state:
db.set_name(message.from_user.id, message.text)
call_back_data = "set_name"
elifstate_key == UserState.passport.state:
db.set_pasport(message.from_user.id, message.text)
call_back_data = "set_pasport"

elifstate_key == UserState.medical.state:
db.set_medical(message.from_user.id, message.text)
call_back_data = "set_medical"
elifstate_key == UserState.email.state:
db.set_email(message.from_user.id, message.text)
call_back_data = "set_email"
elifstate_key == UserState.age.state:
db.set_age(message.from_user.id, message.text)
call_back_data = "set_age"

    await bot.delete_message(message.chat.id, message.message_id
- 1)
    await bot.delete_message(message.chat.id,
message.message_id)
keyboard_reg = types.InlineKeyboardMarkup()
    reg_b1 = types.InlineKeyboardButton(text="Да",
callback_data="log in")
    reg_b2 = types.InlineKeyboardButton(text="Нет",
callback_data=call_back_data)
keyboard_reg.add(reg_b1, reg_b2)
    await state.finish()
    await message.answer(f'Вы указали {message.text}. Верно?',
reply_markup=keyboard_reg)

async def edit_data(call: types.CallbackQuery, state:
FSMContext):
    await call.message.delete()
    if call.data == "edit_name":
        question = "Введите ваше ФИО."

```

```

        await UserState.name_edit.set()
    elif call.data == "edit_pasport":
        question = "Введите вашу серию и номер паспорта в
формате: 1234 56789123."
        await UserState.passport_edit.set()
    elif call.data == "edit_medical":
        question = "Введите номер вашей медсправки в формате: 00
1234567."
        await UserState.medical_edit.set()
    elif call.data == "edit_email":
        question = "Введите ваш электронный адрес."
        await UserState.email_edit.set()
    elif call.data == "edit_age":
        question = "Введите вашу дату рождения в формате:
01.01.1900"
        await UserState.age_edit.set()

    await call.message.answer(question)

async def edit_personal_data(message: types.Message, state:
FSMContext):
    # state_key = state.key
    state_key = await state.get_state()
    if state_key == UserState.name_edit.state:
        db.set_name(message.from_user.id, message.text)
        call_back_data = "edit_name"
    elif state_key == UserState.passport_edit.state:
        db.set_pasport(message.from_user.id, message.text)
        call_back_data = "edit_pasport"

    elif state_key == UserState.medical_edit.state:
        db.set_medical(message.from_user.id, message.text)
        call_back_data = "edit_medical"
    elif state_key == UserState.email_edit.state:
        db.set_email(message.from_user.id, message.text)
        call_back_data = "edit_email"
    elif state_key == UserState.age_edit.state:
        db.set_age(message.from_user.id, message.text)
        call_back_data = "edit_age"

    await bot.delete_message(message.chat.id, message.message_id
- 1)
    await bot.delete_message(message.chat.id,
message.message_id)
    keyboard_reg = types.InlineKeyboardMarkup()
    reg_b1 = types.InlineKeyboardButton(text="Да",
callback_data="edit_data")
    reg_b2 = types.InlineKeyboardButton(text="Нет",
callback_data=call_back_data)
    keyboard_reg.add(reg_b1, reg_b2)
    await state.finish()

```

```
await message.answer(f'Выуказали {message.text}.Верно?',
reply_markup=keyboard_reg)
```

Файл «payment.py»

```
import datetime
from create_bot import dp,bot,db, payment_token
import math
import email
import smtplib
from invoice.generate_send import Generate_invoice,
Convert_and_send

async def payment_menu(message: types.Message):

    await bot.delete_message(message.chat.id,
message.message_id-1) if message.text == "❹Управлениебалансом"
else None
    await bot.delete_message(message.chat.id,
message.message_id)

keyboard_balance = types.InlineKeyboardMarkup(row_width=1)
    b1 = types.InlineKeyboardButton(text="Пополнитьбаланс",
callback_data="operation_add_money")#types.InlineKeyboardButton(
text="Авторизоваться", callback_data="log in")
    b2 = types.InlineKeyboardButton(text="Вывестисредства",
callback_data="operation_return_money")
    b3 = types.InlineKeyboardButton(text="Историяпополнений",
callback_data="operation_history_balance")
keyboard_balance.add(b1,b2,b3)

awaitmessage.answer('Вы перешли в меню управления балансом!
\n\nБла... бла... бла...\n\nЧтобы выйти в главное меню введите
команду /menu или нажмите на нее!',
reply_markup=keyboard_balance)#reply_markup=types.ReplyKeyboardR
emove()

async def confirm_return_money(query: types.CallbackQuery):
    balance = db.get_balance(query.from_user.id)

db.set_payment(query.from_user.id,datetime.datetime.now().date()
, datetime.datetime.now().time(),balance,"RUB",
"return_id","return_tg_bot" )
db.update_balance(query.from_user.id,-balance )
    await query.answer(text="Возвратоформлен. Деньги поступят в
течении 14 рабочих дней. Чтобы отменить возврат обратитесь в
поддержку.",show_alert=True)
```

```

awaitpayment_menu(query.message)

async def operation_money_callback(query: types.CallbackQuery):
    await query.message.delete()
    if "add_money" in query.data:
        if "@" in db.get_email(query.from_user.id):

keyboard_add = types.InlineKeyboardMarkup()
        b1 = types.InlineKeyboardButton(text="500 рублей (1
занятие)", callback_data="add_money_500")
        b2 = types.InlineKeyboardButton(text="1000 рублей (2
занятия)", callback_data="add_money_1000")
        b3 = types.InlineKeyboardButton(text="2500 рублей (5
занятий)", callback_data="add_money_2500")
        b4 = types.InlineKeyboardButton(text="5000 рублей
(10 занятий)", callback_data="add_money_5000")
        b5 = types.InlineKeyboardButton(text="Назад",
callback_data="add_money_back")
keyboard_add.add(b1, b2,b3,b4,b5)

        await
query.message.answer('Выберите сумму для пополнения баланса:',
reply_markup=keyboard_add)
    else:
keyboard_add = types.InlineKeyboardMarkup()
        b5 = types.InlineKeyboardButton(text="Назад",
callback_data="add_money_back")
keyboard_add.add(b5)

await query.message.answer('Ваш email указан некорректно,
либо не указан. Проверьте правильность в личном кабинете',
reply_markup=keyboard_add)

elif "return_money" in query.data:

balance = db.get_balance(query.from_user.id)
keyboard_add = types.InlineKeyboardMarkup()
        b1 =
types.InlineKeyboardButton(text="Запросить средства",
callback_data="confirm_return")
        b5 = types.InlineKeyboardButton(text="Назад",
callback_data="add_money_back")
        if balance == 0:
keyboard_add.add(b5)
            await query.message.answer(f'Ваш баланс 0 рублей.
Возврат недоступен. Если произошла ошибка свяжитесь с
поддержкой.', reply_markup=keyboard_add)
        else:
keyboard_add.add(b1,b5)

```

```
await query.message.answer(f'Вы можете сделать запрос на возврат
средств. Возврат средств осуществляется до 14 рабочих дней.
Возврат средств возможен только в полном объеме. \n\nК возврату
доступно {balance} рублей', reply_markup=keyboard_add)
```

```
elif "history_balance" in query.data:
    history_array = db.get_payment(query.from_user.id)
    message_text = 'Ваши транзакции:\n\n'
    keyboard_add = types.InlineKeyboardMarkup()
    b5 = types.InlineKeyboardButton(text="Назад",
    callback_data="add_money_back")
    keyboard_add.add(b5)
    if len(history_array) != 0:
```

```
for transaction in history_array: # PR сюда добавить "-"
    если было списание и + если было пополнение
    date = transaction[0].strftime('%d.%m.%Y')
        time = transaction[1].strftime('%H:%M:%S')
        amount = transaction[2]
        operation = "Пополнение"
        operation = "Оплата занятия" if "pay_lesson" in
transaction[3] else operation
        operation = "Вывод средств" if "return" in
transaction[3] else operation
        operation = "Отмена занятия" if "cancel_lesson"
in transaction[3] else operation
        sign = "+" if operation in
["Пополнение", "Отмена занятия"] else "-"
    message_text += f'{date} {time}: {sign}{amount} руб. -
{operation}\n'
        await
query.message.answer(message_text, reply_markup=keyboard_add)
    else:
        await
query.message.answer("У вас еще было транзакций.", reply_markup=key
board_add)
```

```
async def add_balance(query: types.CallbackQuery):

    if "back" in query.data:
        await payment_menu(query.message)
    else:
        await query.message.delete()
    chat_id, invoice = payment(query)
    keyboard = types.InlineKeyboardMarkup()
    pay_button = types.InlineKeyboardButton(text="Оплатить",
    pay=True)
```

```

back_button = types.InlineKeyboardButton(text="Назад",
callback_data="add_money_back")
keyboard.row(pay_button, back_button)
        await bot.send_invoice(chat_id,
**invoice,reply_markup=keyboard)

def payment(query: types.CallbackQuery):
chat_id = query.message.chat.id
        value = query.data.split("_")[-1]
        invoice = types.Invoice(
title='Пополнение баланса',
description=f'Пополнение баланса. Кол-возанятий:
{math.floor(int(value) / 500)} ',
payload='Пополнениечерезtelegram',
provider_token=payment_token,
        currency='RUB',
        prices=[types.LabeledPrice('Пополнениебаланса',
int(value)*100)]
        )
invoice_dict = invoice.to_python()
        return chat_id, invoice_dict

async def checkout_process(pre_checkout_query:
PreCheckoutQuery):
        if pre_checkout_query.invoice_payload == "FAILURE_PAYMENT":
            await bot.answer_pre_checkout_query(
                pre_checkout_query.id,
                ok=False,
error_message="Ошибка при проведении платежа."
            )
        else:
            await
bot.answer_pre_checkout_query(pre_checkout_query.id, ok=True)

@dp.message_handler(content_types=ContentType.SUCCESSFUL_PAYMENT
)
async def process_successful_payment(message: Message):
successful_payment = message.successful_payment
id_payment = db.set_payment(message.from_user.id,
datetime.datetime.now().date(),
datetime.datetime.now().time(),successful_payment.total_amount /
100,successful_payment.currency,
successful_payment.provider_payment_charge_id,
successful_payment.invoice_payload )
db.update_balance(message.from_user.id,successful_payment.total_
amount / 100 )
        # send_email_payment(message.from_user.id)
        data = db.get_full_data(message.from_user.id)
        now = datetime.datetime.now()
        try:

```

```

Generate_invoice("test.html",f'invoice{id_payment}{message.from_
user.id}.html',id_payment, data[2], message.from_user.id,
data[5], "000012345",data[1],'4 цифрыкарты', f'{now.date()}
{now.time()} ', successful_payment.invoice_payload,
f'Пополнениебаланса. Кол-возанятий:
{math.floor(int(successful_payment.total_amount / 100) / 500)}
',
successful_payment.total_amount / 100, "Пополнение", data[8] )
except:
print(f"Чекномер {id_payment} несоздан. Ошибка!")

try:
Convert_and_send(f'invoice{id_payment}{message.from_user.id}.htm
l', f'invoice{id_payment}{message.from_user.id}.pdf',data[5] )
except:
print(f"Чек номер {id_payment} не отправлен. Ошибка!")

await bot.delete_message(message.chat.id, message.message_id-1)
await bot.delete_message(message.chat.id,
message.message_id)

keyboard_add = types.InlineKeyboardMarkup()
b1 = types.InlineKeyboardButton(text="Вернуться",
callback_data="add_money_back")
keyboard_add.add(b1)
await bot.send_message(message.chat.id, f"Вашаоплатанасумму
{int(successful_payment.total_amount / 100 )}
рублейпрошлауспешно. Чекбудетвысланна
email",reply_markup=keyboard_add)

```

Файл «*edit_lesson.py*»

```

from create_bot import dp,bot,db
from db import Database
from aiogram import types, Dispatcher
from aiogram.dispatcher import FSMContext
import datetime
import calendar
from aiogram import types
from aiogram.dispatcher.filters.state import State, StatesGroup
from invoice.generate_send import Generate_invoice,
Convert_and_send
import random
import math
from html import escape

async def les_menu(message: types.Message):

```

```

        await bot.delete_message(message.chat.id,
message.message_id-1) if message.text ==
"📅Управление расписанием" else None
        await bot.delete_message(message.chat.id,
message.message_id)

keyboard_rec = types.InlineKeyboardMarkup()
        b1 = types.InlineKeyboardButton(text="Занятия",
callback_data="view_lessons") #types.InlineKeyboardButton(text="А
вторизоваться", callback_data="log in")
        b2 = types.InlineKeyboardButton(text="Экзамены",
callback_data="view_exams")
keyboard_rec.add(b1,b2)

#PR сюда нужно цену занятия.
await message.answer('Вы перешли в меню управления расписанием!
Для продолжения выберите нужный пункт.\n\nЧтобы выйти в главное
меню введите команду /menu или нажмите на нее!',
reply_markup=keyboard_rec) #reply_markup=types.ReplyKeyboardRemov
e()

async def view_works(query: types.CallbackQuery):

    if "lessons" in query.data:
        await query.message.delete()

lessons_tr = db.get_lessons_tr(query.from_user.id)

        today = datetime.date.today()
weekday_num = lessons_tr[0]
days_ahead = weekday_num - today.weekday() + (7 if weekday_num<=
today.weekday() else 0)
lessons_tr = ((today + datetime.timedelta(days_ahead),) +
lessons_tr[1:],) if lessons_tr else ()

lessons_pr =
db.get_lessons_pr(query.from_user.id,datetime.datetime.now().dat
e().strftime('%Y-%m-%d'))
        lessons = lessons_tr + tuple(lessons_pr)
markup = types.InlineKeyboardMarkup(row_width=1)
        if len(lessons) !=0:
            lessons = sorted(lessons, key=lambda x: (x[0],
x[1]))
            for i in range(len(lessons)):
                callback =
f"work_{lessons[i][0]}_{lessons[i][1]}_{lessons[i][2]}_{lessons[
i][3]}_{lessons[i][4]}_{lessons[i][5]}"

```

```

        text = f"Практическоезанятие {lessons[i][0]}
{lessons[i][1]}" if lessons[i][4] == 1 else
f"Теоретическоезанятие {lessons[i][0]} {lessons[i][1]}"
markup.insert(types.InlineKeyboardButton(text,
callback_data=callback))
markup.insert(types.InlineKeyboardButton("Назад",
callback_data='view_back'))
        awaitquery.message.answer('Выберите занятие для
дальнейших действий. Иливернитесьназад.', reply_markup=markup)
    else:
markup.insert(types.InlineKeyboardButton("Назад",
callback_data='view_back'))
        awaitquery.message.answer('Занятий не обнаружено.
Вернитесьназад.', reply_markup=markup)

    if "exams" in query.data:
        await query.message.delete()
        exams = db.get_exams_student(query.from_user.id)
        markup = types.InlineKeyboardMarkup(row_width=1)
        if len(exams) !=0:

            for i in range(len(exams)):
                callback =
f"work_ex_{exams[i][0]}_{exams[i][1]}_{exams[i][2]}_{exams[i][3]}
}"

                text = f"Теоретическийэкзамен {exams[i][1]}" if
exams[i][0] == 3 else f"Практическийэкзамен {exams[i][1]}"
markup.insert(types.InlineKeyboardButton(text,
callback_data=callback))
markup.insert(types.InlineKeyboardButton("Назад",
callback_data='view_back'))
                awaitquery.message.answer('Выберите экзамен для
дальнейших действий. Иливернитесьназад.', reply_markup=markup)
            else:
markup.insert(types.InlineKeyboardButton("Назад",
callback_data='view_back'))
                awaitquery.message.answer('Записей не обнаружено.
Вернитесьназад.', reply_markup=markup)

        if "back" in query.data:
            await les_menu(query.message)

async def edit_works(query: types.CallbackQuery):

    if "ex" in query.data:
        lesson_type, date, name_inspector, id_exam =
query.data.split('_')[2:]
        markup = types.InlineKeyboardMarkup(row_width=1)
        rules = 'Вы не можете отменить эту запись.'

```

```

if datetime.date.today().strftime('%Y-%m-%d') < date:
    rules = 'Вы можете отменить эту запись.'
markup.insert(types.InlineKeyboardButton(text="Отменить запись",
callback_data=f"return_exam_{id_exam}"))
markup.insert(types.InlineKeyboardButton(text="Назад",
callback_data="view_back"))

    if lesson_type == '3':
        msg = f'Ваша запись: \nТеоретический экзамен \nДата
{date} \n \n Инспектор: \n {name_inspector} \n \n Место занятия: \n МРЭО
ГИБДД \n \n \n {rules}'
    elif lesson_type == '4':
        msg = f'Ваша запись: \n Практический экзамен \n Дата
{date} \n \n Инспектор: \n {name_inspector} \n \n Место занятия: \n Уточняй
те по телефону. \n \n \n {rules}'
    else:
        date, time, id_teacher, id_class, lesson_type,
id_practic = query.data.split('_')[1:]
        markup = types.InlineKeyboardMarkup(row_width=1)
        rules = 'Вы не можете отменить эту запись.'
if lesson_type == '1':
info_teacher, info_class = db.get_ihfoles(id_teacher, id_class)
    if datetime.date.today().strftime('%Y-%m-%d') <
date:
        rules = 'Вы можете отменить эту запись.'
markup.insert(types.InlineKeyboardButton(text="Отменить занятие",
callback_data=f"return_lesson_{id_practic}"))
        msg = f'Ваше занятие: \n Практическое занятие \n Дата
{date}      Время {time}
\n \n Инструктор: \n {info_teacher[0]} \n Телефон:
+{info_teacher[1]} \n email:      {info_teacher[2]} \n \n
Место занятия: \n {info_class[0]} {info_class[1]} \n \n \n {rules}'
    elif lesson_type == '2':
info_teacher, info_class = db.get_ihfoles(id_teacher, id_class)
        msg = f'Ваше занятие: \n Теоретическое занятие \n Дата
{date}      Время {time}
\n \n Инструктор: \n {info_teacher[0]} \n Телефон:
+{info_teacher[1]} \n email:      {info_teacher[2]} \n \n
Место занятия: \n {info_class[0]} каб. {info_class[1]} \n \n \n
{rules}'
markup.insert(types.InlineKeyboardButton(text="Назад",
callback_data="view_back"))
        await query.message.answer(msg, reply_markup=markup)

Generate_invoice("test.html", f'invoice_return{number}{query.from
_user.id}.html', number, data[2], query.from_user.id,
data[5], "000012345", data[1], '4 цифры карты', f'{now.date()}
{now.time()}', 'Возврат',
'Возврат за занятие.', 500, "Возврат", data[8] )
except:
print(f"Чек номер {number} не создан. Ошибка!")
try:

```

```
Convert_and_send(f'invoice_return{number}{query.from_user.id}.html', f'invoice_return{number}{query.from_user.id}.pdf', data[5] )
```

```
except:
print(f"Чек номер {number} не отправлен. Ошибка!")
```

```
await query.answer(text="Занятие успешно отменено. Деньги  
вернуться на Ваш баланс.", show_alert=True)
await les_menu(query.message)
```

Файл «*rec_lesson.py*»

```
from create_bot import dp, bot, db
from db import Database
from aiogram import types, Dispatcher
from aiogram.dispatcher import FSMContext
import datetime
import calendar
from aiogram import types
from aiogram.dispatcher.filters.state import State, StatesGroup
from invoice.generate_send import Generate_invoice,
Convert_and_send

class UserRec(StatesGroup):
    date = State()
    time = State()

#@dp.message_handler(lambda message: message.text ==
"🚗Записаться на вождение")
async def rec_menu(message: types.Message):

    await bot.delete_message(message.chat.id,
message.message_id-1) if message.text == "🚗Записаться на вождение"
    else None
    await bot.delete_message(message.chat.id,
message.message_id)

keyboard_rec = types.InlineKeyboardMarkup(row_width=1)
    b1 = types.InlineKeyboardButton(text="Занятие",
callback_data="lesson") #types.InlineKeyboardButton(text="Авториз
оваться", callback_data="log in")
    b2 = types.InlineKeyboardButton(text="Теоретический экзамен",
callback_data="t_exam")
    b3 = types.InlineKeyboardButton(text="Практический экзамен",
callback_data="p_exam")
```

```
keyboard_rec.add(b1,b2,b3)
```

```
#PR сюда нужно цену занятия.
awaitmessage.answer('Вы перешли в меню записи! Выберите на что
хотите записаться.\n\n Помните, что запись на экзамены
формируется только после прохождения всех теоретических и
парктических занятий предусмотренных программой обучения.\n\nДни
для экзамена формирует МРЭО ГИБДД самостоятельно, обычно 2 дня в
неделю. Нажмите на нужный экзамен для просмотра этих дней.
\n\n\nОплата экзаменов происходит в МРЭО ГИБДД. Не забудьте мед.
справку, паспорт и сертификат об окончании Автошколы.\n\nЧтобы
выйти в главное меню введите команду /menu или нажмите на нее!',
reply_markup=keyboard_rec)#reply_markup=types.ReplyKeyboardRemov
e()
```

```
async def rec(callback: types.CallbackQuery, state: FSMContext):
    lesson = False
    t_exam = False
    p_exam = False
```

```
        if callback.data == "lesson":
            lessons =
db.get_lessons_pr(callback.from_user.id,datetime.datetime.now().
date()).strftime('%Y-%m-%d'),1)
            if len(lessons)<5:

                lesson = True
                await callback.message.delete()
                await bot.answer_callback_query(callback.id)
                await show_calendar(lesson, t_exam, p_exam,
callback.message.chat.id)
            else:
awaitcallback.answer(text="Вы не можете записаться больше чем на
5 занятий одновременно.",show_alert=True)
await rec_menu(callback.message)
```

```
elifcallback.data == "t_exam":
    t_exam = True
        await callback.message.delete()
        await bot.answer_callback_query(callback.id)
        await exams(lesson, t_exam, p_exam,
callback.message.chat.id)
```

```
elifcallback.data == "p_exam":
    p_exam = True
        await callback.message.delete()
```

```

        await bot.answer_callback_query(callback.id)
        await exams(lesson, t_exam, p_exam,
callback.message.chat.id)

elif callback.data == "back_to_rec":
    await rec_menu(callback.message)

async def exams(lesson, t_exam, p_exam, chat_id: int):
    if t_exam:
        date_array = db.show_date_exam(3, datetime.date.today())
        markup = types.InlineKeyboardMarkup(row_width=1)
        for i in range(len(date_array)):
            count_st = db.show_count_slots(date_array[i][0])
            date_str = date_array[i][1].strftime("%Y-%m-%d")
            callback_data = f"rec- t_exam -{date_str}"
            button_text = f"{date_str}, Осталось {date_array[i][2]-
len(count_st)} из {date_array[i][2]} мест"
            markup.insert(types.InlineKeyboardButton(button_text,
callback_data=callback_data))

        markup.insert(types.InlineKeyboardButton("Назад",
callback_data="back_to_rec"))
        await bot.send_message(chat_id,
"Выберите день теоретического экзамена. Нажимая на выбранный день,
вы подтверждаете свою запись на этот день. Отменить эту запись
можно будет в главном меню.", reply_markup=markup)
    if p_exam:
        date_array = db.show_date_exam(4, datetime.date.today())
        markup = types.InlineKeyboardMarkup(row_width=1)
        for i in range(len(date_array)):
            count_st = db.show_count_slots(date_array[i][0])
            date_str = date_array[i][1].strftime("%Y-%m-%d")
            callback_data = f"rec- p_exam -{date_str}"
            button_text = f"{date_str}, Осталось {date_array[i][2]-
len(count_st)} из {date_array[i][2]} мест"
            markup.insert(types.InlineKeyboardButton(button_text,
callback_data=callback_data))

        markup.insert(types.InlineKeyboardButton("Назад",
callback_data="back_to_rec"))
        await bot.send_message(chat_id, "Выберите день
практического экзамена. Нажимая на выбранный день, вы
подтверждаете свою запись на этот день. Отменить эту запись
можно будет в главном меню.", reply_markup=markup)

async def show_calendar(lesson, t_exam, p_exam, chat_id: int):
    markup = create_calendar(lesson, t_exam, p_exam)
    await bot.send_message(chat_id, "Выберите дату",
reply_markup=markup)

```

```

def create_calendar(lesson, t_exam, p_exam, year=None,
month=None):
    now = datetime.datetime.now()
    if year is None:
        year = now.year
    if month is None:
        month = now.month
    markup = types.InlineKeyboardMarkup(row_width=7)
    month_names = ("Январь", "Февраль", "Март", "Апрель", "Май",
"Июнь",
                    "Июль", "Август", "Сентябрь", "Октябрь",
"Ноябрь", "Декабрь")
    # Первая строка - месяц и год
    row = [types.InlineKeyboardButton("◀",
callback_data=f"calendar-prev-month-{year}-{month}-{lesson}-{t_exam}-{p_exam}"),
types.InlineKeyboardButton(month_names[month - 1] + " " +
str(year), callback_data="ignore"),
types.InlineKeyboardButton("▶", callback_data=f"calendar-next-month-{year}-{month}-{lesson}-{t_exam}-{p_exam}")]
    markup.row(*row)
    # Вторая строка - дни недели
    week_days = ("Пн", "Вт", "Ср", "Чт", "Пт", "Сб", "Вс")
    row = [types.InlineKeyboardButton(day, callback_data="ignore")
for day in week_days]
    markup.row(*row)
    # Остальные строки - дни месяца
    month_calendar = calendar.monthcalendar(year, month)
    for week in month_calendar:
        row = []
        for day in week:
            if day == 0:
row.append(types.InlineKeyboardButton(" ",
callback_data="ignore"))
            else:
row.append(types.InlineKeyboardButton(str(day),
callback_data=f"calendar-day lesson -{month}-{day}"))

    markup.row(*row)
    return markup

async def calendar_prev_month_callback(query):
    lesson, t_exam, p_exam = query.data.split('-')[5:8]
    year, month = map(int, query.data.split('-')[3:5])
    prev_month = month - 1
    prev_year = year
    if prev_month == 0:

```

```

prev_month = 12
prev_year -= 1
    await
bot.edit_message_reply_markup(chat_id=query.message.chat.id,
message_id=query.message.message_id,
reply_markup=create_calendar(lesson, t_exam, p_exam,prev_year,
prev_month))

async def calendar_next_month_callback(query):
    lesson, t_exam, p_exam = query.data.split('-')[5:8]
    year, month = map(int, query.data.split('-')[3:5])
    next_month = month + 1
    next_year = year
    if next_month == 13:
        next_month = 1
        next_year += 1
    await
bot.edit_message_reply_markup(chat_id=query.message.chat.id,
message_id=query.message.message_id,
reply_markup=create_calendar(lesson, t_exam, p_exam,next_year,
next_month))

async def show_time_picker(user_id: int, message_id: int,
chat_id: int, selected_day: int,selected_month:int):
    await bot.delete_message(chat_id,message_id)
    time_array = db.check_table(user_id,
selected_month,selected_day)
    for i in range(len(time_array)):
        time_array[i] = time_array[i][0].strftime("%H:%M")
    # Показываем пользователю выбор времени
    now = datetime.datetime.now()
    markup = types.InlineKeyboardMarkup(row_width=2)
    for hour in range(8, 19):
        if hour != 12:
            time_str = f"{hour:02}:00"
            callback_data = f"lesson-time-2023-{selected_month}-{
selected_day}-{time_str}"
            if time_str not in time_array and
datetime.datetime.strptime(callback_data, "lesson-time-%Y-%m-%d-
%H:%M") >datetime.datetime.combine(now.date(), now.time()) and
datetime.datetime.strptime(callback_data, "lesson-time-%Y-%m-%d-
%H:%M").weekday() in [1,2,3,4]:
                button_text = time_str
                markup.insert(types.InlineKeyboardButton(button_text,
callback_data=callback_data))
            else:
                button_text = time_str
                button_text = f"{time_str} (занято)" if time_str in time_array
                else button_text

```

```

button_text = f"{time_str} (уже прошло)" if
datetime.datetime.strptime(callback_data, "lesson-time-%Y-%m-%d-
%H:%M") < datetime.datetime.combine(now.date(), now.time()) else
button_text
button_text = f"{time_str} (выходной)" if
datetime.datetime.strptime(callback_data, "lesson-time-%Y-%m-%d-
%H:%M").weekday() in [5,6] else button_text

callback_data = callback_data + " занято" if "(занято)" in
button_text else callback_data
callback_data = callback_data + " выходной" if "(выходной)" in
button_text else callback_data
callback_data = callback_data + " прошло" if "прошло" in
button_text else callback_data
markup.insert(types.InlineKeyboardButton(button_text,
callback_data=callback_data))
    else:
        pass
markup.insert(types.InlineKeyboardButton("Назад",
callback_data="back_to_rec"))
    await bot.send_message(chat_id, "Выберите время",
reply_markup=markup)

async def process_lesson_callback(callback_data:
types.CallbackQuery):
    if "прошло" in callback_data.data:
        keyboard = types.InlineKeyboardMarkup()

        b2 = types.InlineKeyboardButton(text="Назад",
callback_data="back_to_rec")
        selected_year_str, selected_month_str, selected_day_str,
selected_time_str = callback_data.data.split('-')[2:]
        keyboard.add(b2)
        await callback_data.message.delete()

        await callback_data.message.answer
('ВЫ ВЫБРАЛИ ДАТУ КОТОРАЯ ПРОШЛА. Вернитесь назад!',
reply_markup=keyboard)
        if "выходной" in callback_data.data:
            keyboard = types.InlineKeyboardMarkup()

            b2 = types.InlineKeyboardButton(text="Назад",
callback_data="back_to_rec")
            selected_year_str, selected_month_str, selected_day_str,
selected_time_str = callback_data.data.split('-')[2:]
            keyboard.add(b2)
            await callback_data.message.delete()

            await callback_data.message.answer
('ВЫ ВЫБРАЛИ ВЫХОДНОЙ ДЕНЬ. Вернитесь назад!',
reply_markup=keyboard)

```

```

        if "занято" in callback_data.data:
            keyboard = types.InlineKeyboardMarkup()

            b2 = types.InlineKeyboardButton(text="Назад",
callback_data="back_to_rec")
selected_year_str,selected_month_str,selected_day_str,
selected_time_str = callback_data.data.split('-')[2:]
keyboard.add(b2)
            await callback_data.message.delete()

            awaitcallback_data.message.answer ('Эти дата и время уже
заняты. Вернитесьназад!', reply_markup=keyboard)

        else:

selected_year_str,selected_month_str,selected_day_str,
selected_time_str = callback_data.data.split('-')[2:]
            keyboard = types.InlineKeyboardMarkup()
            b1 = types.InlineKeyboardButton(text="Записаться",
callback_data=f"rec- lesson -{selected_year_str}-
{selected_month_str}-{selected_day_str}-{selected_time_str}")
            b2 = types.InlineKeyboardButton(text="Назад",
callback_data="back_to_rec")

keyboard.add(b1,b2)

            await callback_data.message.delete()

            await callback_data.message.answer (f'Вашазапись:Год
{selected_year_str} День {selected_day_str},Месяц
{selected_month_str}, Время {selected_time_str}',
reply_markup=keyboard)

async def process_calendar_day(callback_query:
types.CallbackQuery):
selected_month,selected_day = callback_query.data.split('-')[2:]
    await
show_time_picker(callback_query.from_user.id,callback_query.mess
age.message_id,callback_query.message.chat.id,
selected_day,selected_month)

async def recover(callback_query: types.CallbackQuery):
    if "lesson" in callback_query.data:
selected_year_str,selected_month_str,selected_day_str,
selected_time_str = callback_query.data.split('-')[2:]
        try:

```

```

db.update_balance(callback_query.from_user.id,-500 ) # PR
суданужноценузантия
db.rec_lesson(callback_query.from_user.id,
f"{selected_year_str}-{selected_month_str}-
{selected_day_str}",f"{selected_time_str}")
id_payment =
db.set_payment(callback_query.from_user.id,datetime.datetime.now
().date(), datetime.datetime.now().time(),500,"RUB",
"id_pay_lesson","pay_lesson")

```

```

data = db.get_full_data(callback_query.from_user.id)
now = datetime.datetime.now()
try:

```

```

Generate_invoice("test.html",f'invoice{id_payment}{callback_quer
y.from_user.id}.html',id_payment, data[2],
callback_query.from_user.id,
data[5], "000012345",data[1], '4 цифрыкарты', f'{now.date()}
{now.time()}', "Оплата",
'Оплата 1 практического занятия',500, "Оплата", data[8] )
except:
print(f"Чекномер {id_payment} несоздан. Ошибка!")

```

```

try:

```

```

Convert_and_send(f'invoice{id_payment}{callback_query.from_user.
id}.html',
f'invoice{id_payment}{callback_query.from_user.id}.pdf',data[5]
)

```

```

except:
print(f"Чек номер {id_payment} не отправлен. Ошибка!")

```

```

awaitcallback_query.answer(text="Запись на занятие прошла
успешно! Чек об оплате придет на ваш email")
awaitrec_menu(callback_query.message)
except:

```

```

awaitcallback_query.answer(text="Записатьсянеудалось. Возможно у
Вас не хватает средств на балансе или это время было
занято",show_alert=True)

```

```

if "t_exam" in callback_query.data:
selected_year_str,selected_month_str,selected_day_str =
callback_query.data.split('-')[2:]
rule =
db.check_rul_examen(callback_query.from_user.id,0)
if rule[0]:
try:

```

```

db.rec_exam( callback_query.from_user.id,f"{selected_year_str}-{
selected_month_str}-{selected_day_str}", 3)
        awaitcallback_query.answer(text="Запись на
теоретический экзамен прошла успешно!")
await rec_menu(callback_query.message)
        except:

awaitcallback_query.answer(text="Записатьсянеудалось. Возможно
вы не прошли все занятия или уже были записаны на эту дату. Вы
будете возвращены в меню записи.",show_alert=True)
awaitrec_menu(callback_query.message)
else:
awaitcallback_query.answer(text="Записаться не удалось. Возможно
вы не прошли все занятия или уже были записаны на эту дату. Вы
будете возвращены в меню записи.",show_alert=True)
awaitrec_menu(callback_query.message)

if "p_exam" in callback_query.data:
selected_year_str,selected_month_str,selected_day_str =
callback_query.data.split('-')[2:]
        rule =
db.check_rul_examen(callback_query.from_user.id,0)
        if rule[0]:
                try:

db.rec_exam( callback_query.from_user.id,f"{selected_year_str}-{
selected_month_str}-{selected_day_str}", 4)
                awaitcallback_query.answer(text="Запись на
практический экзамен прошла успешно!")
await rec_menu(callback_query.message)

                except:

awaitcallback_query.answer(text="Записатьсянеудалось. Возможно
вы не прошли все занятия или уже были записаны на эту дату. Вы
будете возвращены в меню записи.",show_alert=True)
awaitrec_menu(callback_query.message)
else:
awaitcallback_query.answer(text="Записаться не удалось. Возможно
вы не прошли все занятия или уже были записаны на эту дату. Вы
будете возвращены в меню записи.",show_alert=True)
awaitrec_menu(callback_query.message)

```

Файл «generate_send.py»

```

from bs4 import BeautifulSoup
import pdfkit
import yagmail

class Generate_invoice:

```

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		140

```

    def __init__(self, input_file_path, output_file_path,
a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13):
self.input_file_path = input_file_path
self.output_file_path = output_file_path
self.replacements = {'id_payment': f'{a1}', 'st_name': f'{a2}',
'user_id': f'{a3}', 'email': f'{a4}', 'INN': f'{a5}',
'number_phone': f'+{a6}',
                    'card': f'{a7}', 'datetime': f'{a8}',
'name_pay': f'{a9}', 'opis_pay': f'{a10}', 'Psum': f'₽{a11}',
'name_transaction': f'{a12}',
                    'Pbalance': f'₽{a13}'}
self.generate_html()

    def generate_html(self):
        with open(self.input_file_path, 'r', encoding='utf-8')
as file:
html_code = file.read()
        soup = BeautifulSoup(html_code, 'html.parser')
# Заменить значения атрибутов на новые значения
for element in soup.find_all():
        for key, value in self.replacements.items():
# Ищем совпадение внутри атрибутов элемента
if key in element.attrs:
element.attrs[key] = value

        # Ищем совпадение внутри текста элемента
elif element.string and key in element.string:
        # Заменяем совпадение в тексте элемента
element.string.replace_with(element.string.replace(key, value))
# Ищем совпадение внутри текста элемента, если он содержит
переносы строк
elif element.name == 'br' and key in str(element.next_sibling):
# Заменяем совпадение в тексте, находящемся после тега br
element.next_sibling.replace_with(str(element.next_sibling).repl
ace(key, value))
elif element.name == 'br' and key in str(element.next_sibling):
# Заменяем совпадение в тексте, находящемся после тега br
element.next_sibling.replace_with(str(element.next_sibling).repl
ace(key, value))

# Сохранить измененный HTML-код в новом файле
with open(self.output_file_path, 'w', encoding='utf-8') as
new_file:
new_file.write(str(soup))

class Convert_and_send:
    def __init__(self, html_path, pdf_path, to_address):
self.html_path = html_path
self.pdf_path = pdf_path
self.to_address = to_address
self.convert_to_pdf()

```

```

def convert_to_pdf(self):
    try:
pdfkit.from_file(self.html_path, self.pdf_path)
    except Exception as e:
print(f"An error occurred during PDF conversion: {e}")

self.send_email()

def send_email(self):
# создаем объект yagmail.SMTP для отправки писем
yag = yagmail.SMTP('autoschool058@mail.ru',
'cdjMzQQ96qt6fHnMbxss', host='smtp.mail.ru', port=465,
smtp_starttls=False, smtp_ssl=True)

# отправляем сообщение
try:
yag.send(
to=self.to_address,
subject='Кассовый чек',
attachments=self.pdf_path,
contents='Благодарим за транзакцию. Ваш электронный чек.'
)
    except Exception as e:
print(f"An error occurred during email sending: {e}")
    return False
    finally:
yag.close()

```

ПриложениеВ

(Обязательное)

Стартап-проект информационная система «Автошкола» на основе телеграм-

бота

Презентация

					ПГУ1.090301.39.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		143