



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Московский государственный технический университет имени
Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **«ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»**

КАФЕДРА **«ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ
ТЕХНОЛОГИИ»(ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ
НА ТЕМУ:**

**«Разработка базы данных для создания чат-бота на
платформе Telegram для управления событиями»**

Студент	<u>ИУ7-66Б</u> (Группа)	<u> </u> (Подпись,дата)	<u>Д.А.Мокеев</u> (И.О.Фамилия)
Руководитель курсового проекта		<u> </u> (Подпись,дата)	<u>О.В.Кузнецова</u> (И.О.Фамилия)

Москва, 2021

Реферат

Курсовой проект представляет собой телеграм-бота с возможностью записи, просмотра и управления событиями.

Ключевые слова: web-приложение, SQLight, PeeWee, Telegram, ORM.

Приложение реализовано на языке программирования Python.

Отчёт содержит 28 страниц, 12 рисунков, 13 источников.

Содержание

Введение	5
1 Аналитическая часть	6
1.1 Постановка задачи	6
1.2 Общие сведения о БД и СУБД	7
1.2.1 Основные функции СУБД	7
1.2.2 Классификация СУБД по модели данных	7
Иерархическая модель	7
Сетевая модель	8
Реляционная модель	8
1.3 Разбор различных СУБД	10
1.3.1 SQLite	10
1.3.2 MySQL	11
1.3.3 PostgreSQL	11
1.4 Анализ алгоритмов работы чат-бота	12
1.4.1 Long Polling — алгоритм опрашиваемых операций	12
1.4.2 Webhooks — алгоритм оповещения системы о событиях	13
1.5 Обоснование выбора алгоритма чат-бота	13
2 Конструкторская часть	14
2.1 Диаграмма вариантов использования	14
2.2 Конструирование базы данных	16
2.2.1 Модель системы	17
2.2.2 Структуры данных	17
2.2.3 Модель данных	18
3 Технологическая часть	20
3.1 Средства реализации	20
3.2 Использование объектно-реляционного отображения	20
3.3 Реализация доступа к данным	22
3.4 Описание интерфейса чат-бота	23
3.5 Вывод	24
Заключение	24
Список использованной литературы	27

Введение

Звонки по телефону поколение Z не любит больше всего. Чтобы ориентироваться на так называемых зумеров как на целевую аудиторию необходима выработка альтернативных способов коммуникации с целью поддержания связи с клиентами. Представим бар, проводящий большое количество мероприятий. Место популярное и на мероприятия приходит много человек, но для всех вместимости не хватает. Одно из возможных решений – регистрация на эти события. Для этого можно предложить много решений – звонки, сообщения в инстаграме, разработка сайта или чат-бот. Целью данного курсового проекта является разработка чат-бота на платформе мессенджера Telegram с подключением базы данных. Задачей чат-бота является осуществлять запись гостей на мероприятия. Все запросы пользователей должны обрабатываться с использованием базы данных.

Чат-бот должен предоставлять следующий функционал:

- Для клиентов – выбрать ближайшие доступные события и зарегистрироваться на них
- для администраторов - управлять событиями, а именно добавлять новые, изменять названия, даты и время проведения, описание и другое, отменять и возвращать к возможности записи.
- для обеих ролей - регистрироваться, управлять настройками своего аккаунта, получать справочную информацию.

При этом интерфейс должен быть спроектирован просто и понятно, но при этом поддерживать безболезненное добавление новых элементов для обоих ролей пользователей.

Для достижения цели поставлены следующие задачи:

1. Анализ существующих СУБД, выбор СУБД для этого проекта;
2. проектирование программного обеспечения;
3. описать структуру базы данных, включая объекты, из которых она состоит;
4. спроектировать приложение для доступа к БД;
5. разработать приложение, которое позволит добавлять и использовать данные;
6. тестирование приложения.

1. Аналитическая часть

В данном разделе будут рассмотрены постановка задачи, общие сведения о БД и СУБД, основные реляционные СУБД на рынке и основные алгоритмы чат-ботов.

1.1 Постановка задачи

В соответствии с техническим заданием на курсовой проект необходимо разработать приложение, позволяющее работать с событиями для гостей и администраторов. Для программного обеспечения выдвигаются следующие требования:

- Необходимо разработать чат-бота на платформе мессенджера Telegram с подключением базы данных, то есть, необходимо реализовать две составляющие: бота(приложение) и базу данных;
- задачей чат-бота является предоставления сервиса клиентам и сотрудникам бара. То есть, программа должна выполнять свою основную задачу, а также необходима корректная работа программы;
- все запросы должны обрабатываться с использованием базы данных. То есть, должны быть разработаны запросы к базе данных;
- чат-бот должен иметь следующие функции:
 - Для клиентов –
 - * просмотр событий на текущей неделе;
 - * запись на событие;
 - * отмена записи на событие;
 - Для администраторов –
 - * просмотр статистики по регистрациям;
 - * добавление новых событий;
 - * редактирование существующих событий;
 - Для обоих ролей –
 - * регистрацию нового пользователя;
 - * возможность смены имени.
- архитектура приложения и разработанный UI должны быть понятны и открыты к возможным будущим изменениям.

1.2 Общие сведения о БД и СУБД

База данных — совокупность взаимосвязанных данных некоторой предметной области, хранимых в памяти ЭВМ и организованных таким образом, что эти данные могут быть использованы для решения многих задач многими пользователями. [1].

Система управления базами данных (СУБД) – комплекс программно-языковых средств, позволяющих создать базы данных и управлять данными. СУБД – набор программ, позволяющий организовывать, контролировать и администрировать базы данных. Большинство сайтов не могут функционировать без базы данных, поэтому СУБД используется практически повсеместно. [2].

1.2.1 Основные функции СУБД

Основными функциями СУБД являются:

- управление данными во внешней памяти;
- управление данными в оперативной памяти с использованием дискового кэша;
- журнализация изменений, резервное копирование и восстановление базы данных после сбоев;
- поддержка языков БД.

1.2.2 Классификация СУБД по модели данных

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных[3].

Существует 3 основных типа моделей организации данных:

- иерархическая;
- сетевая;
- реляционная.

Иерархическая модель

Иерархическая модель - самая ранняя модель представления сложной структуры данных. Информация в иерархической базе организована по принципу древовидной структуры, в виде отношений «предок-потомок».

Каждая запись может иметь не более одной родительской записи и несколько подчиненных. Связи записей реализуются в виде физических указателей с одной записи на другую. Основной недостаток иерархической структуры базы данных — невозможность реализовать отношения «многие-к-многим», а также ситуации, когда запись имеет несколько предков.

Графически такую структуру можно изобразить в виде дерева, состоящего из объектов различных уровней. Верхний уровень занимает один объект, второй — объекты второго уровня и так далее.

Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка (объект более близкий к корню) к потомку (объект более низкого уровня), при этом возможно, чтобы объект-предок не имел потомков или имел их несколько, тогда как у объекта-потомка обязательно только один предок. Объекты, имеющие общего предка, называются близнецами[4].

На рисунке 1.1 представлена структура иерархической модели данных.

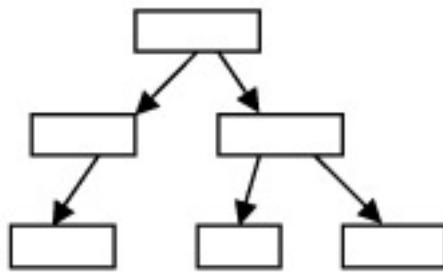


Рис. 1.1: Структура иерархической модели данных

Сетевая модель

В сетевой структуре любой элемент может быть связан с любым другим элементом, и каждый из элементов может являться входом в структуру. Данные в сетевой модели представлены в виде совокупностей записей, а связи — в виде наборов. Сетевая модель является обобщением иерархической модели.

Сетевую структуру также можно описать с помощью исходных и порожденных элементов: каждый элемент может иметь как несколько порожденных, так и несколько исходных элементов. В ней порожденные элементы располагаются ниже исходных. В простых сетевых структурах между парой элементов поддерживается отношение «один – ко – многим». Направление и характер связи между элементами не является очевидным, и поэтому направление связи должно быть указано[5].

Главным недостатком сетевой модели данных являются жесткость и высокая сложность схемы базы данных, построенной на основе этой модели. Так как логика процедуры выбора данных зависит от физической организации этих данных, эта модель не является полностью независимой от приложения.

На рисунке 1.2 представлена структура сетевой модели данных.

Реляционная модель

Реляционная модель базируется на теоретико-множественном понятии отношения. В математических дисциплинах существует понятие «отношение» (relation), физическим представлением которого является таблица.

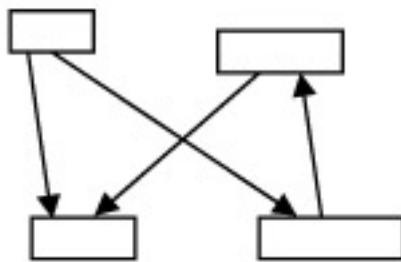


Рис. 1.2: Структура сетевой модели данных

Отсюда и произошло название модели – реляционная. Применительно к БД понятия «реляционная БД» и «табличная БД» являются синонимами. Реляционные базы получили наибольшее распространение в мире. Почти все продукты БД, созданные с конца 70-х годов, являются реляционными. В 1970 году появились работы, в которых обсуждались возможности применения различных табличных моделей данных. Наиболее значительной из них была статья сотрудника фирмы IBM Э.Кодда, где впервые был применен термин "реляционная модель данных". Проект System R был разработан в исследовательской лаборатории корпорации IBM. Этот проект был задуман с целью доказать практичесность реляционной модели.

Реляционная модель данных является совокупностью данных и состоит из набора двумерных таблиц. При табличной организации отсутствует иерархия элементов. Таблицы состоят из строк – записей и столбцов – полей. На пересечении строк и столбцов находятся конкретные значения. Для каждого поля определяется множество его значений. За счет возможности просмотра строк и столбцов в любом порядке достигается гибкость выбора подмножества элементов. Реляционная модель является удобной и наиболее широко используемой формой представления данных[6].

На рисунке 1.3 представлена структура реляционной модели данных.

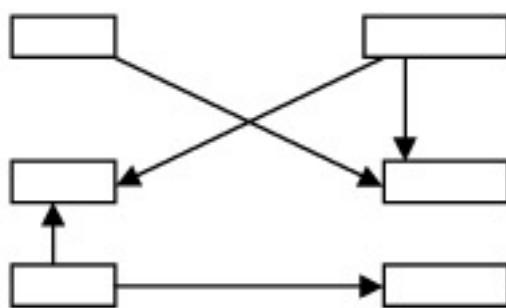


Рис. 1.3: Структура реляционной модели данных

В результате анализа моделей баз данных, в соответствии с поставленной задачей, оптимальным решением является использование реляционной модели базы данных, так как это позволит реализовать поставленные цели, не усложняя программную архитектуру.

1.3 Разбор различных СУБД

Для данной задачи подойдет использование реляционной модели данных. Реляционная модель состоит из трех частей: структурной, целостностной и манипуляционной. Структурная часть реляционной модели описывает, из каких объектов состоит реляционная модель. В целостностной части реляционной модели фиксируются два базовых требования целостности, которые должны выполняться для любых отношений в любых реляционных базах данных. Это целостность сущностей и ссылочная целостность. Манипуляционная часть реляционной модели описывает два эквивалентных способа манипулирования реляционными данными – реляционную алгебру и реляционное исчисление. [7] Таким образом, выбранная СУБД должна поддерживать реляционную модель данных. То есть, сущности должны храниться в таблицах (структурная часть), а также сущности разных таблиц должны иметь возможность быть связанными, то есть должна присутствовать возможность выстраивать отношения между сущностями при помощи внешних ключей (целостная часть). Также должна иметься возможность управлять данными, изменять их, производить поиск по данным и так далее (манипуляционная часть).

Для разработки была выбрана реляционная модель базы данных, поэтому рассмотрим три наиболее важные и популярные СУБД с открытым исходным кодом: SQLite, MySQL, PostgreSQL.

1.3.1 SQLite

- это встраиваемая файловая СУБД, которая не требует установки или настройки. Это, в свою очередь, означает, что приложение не запускается в рамках отдельного серверного процесса, который необходимо запускать, останавливать или настраивать. Такая безсерверная архитектура позволяет базе данных поддерживать кроссплатформенность. Полная база данных SQL содержится в одном файле на диске, и все операции чтения и записи осуществляются непосредственно в этот файл на диске. Поскольку данные напрямую записываются обратно в файл диска, SQLite поддерживает систему ACID защиты транзакций от сбоев выделения памяти и ошибок ввода-вывода диска, которые могут возникнуть в результате непредвиденных сбоев системы или сбоев питания. [8].

Преимущества:

- + Библиотека SQLite - одна из самых компактных библиотек СУБД, размер библиотеки может быть меньше 600 КБ.
- + База использует для работы единственный файл, что хорошо влияет на ее переносимость.
- + Легко настраивается и показывает хорошие результаты на веб-сайтах с низким и средним трафиком (100 тыс. Запросов в день).
- + Совместима с языком программирования Python.

Недостатки:

- Отсутствие встроенного шифрования данных, что стало стандартом для предотвращения наиболее распространенных хакерских атак в интернете.

- База не поддерживает систему учета пользователей, в то время как другие популярные СУБД поддерживают эту возможность.
- Только одна операция записи за транзакцию, что уменьшает производительность системы

1.3.2 MySQL

MySQL - одна из самых популярных и широко распространенных СУБД с открытым исходным кодом. В отличие от SQLite, она использует архитектуру клиент-сервер, которая состоит из многопоточного сервера SQL. Эта многопоточная природа MySQL обеспечивает большую производительность, поскольку потоки ядра могут легко использовать несколько процессоров. База данных написана на C и C ++ и поддерживает различные платформы. Она также придерживается системы ACID для обеспечения согласованности транзакций. [8]

Преимущества:

- + Обширный функционал и наличие корпоративных функций.
- + Важным отличием MySQL от SQLite является поддержка многопользовательских функциями MySQL. Это и наличие корпоративных функций, делает эту СУБД идеально подходящей для распределенных приложений.
- + MySQL имеет преимущество перед PostgreSQL, когда речь идет о пропускной способности и производительности. Она также намного проще в установке и использовании и более широко используется в сравнению с PostgreSQL.

Недостатки:

- Плохо проявляется на больших операциях SELECT и BULK INSERT.
- Уязвимости: метод обработки данных, применяемый в MySQL, делает эту СУБД немного менее надежной по сравнению с другими СУБД.
- Медленное развитие: хотя MySQL является продуктом с открытым исходным кодом, он очень медленно развивается.

1.3.3 PostgreSQL

PostgreSQL — это продвинутая открытая объектно-ориентированная СУБД. PostgreSQL — это СУБД с открытым исходным кодом, в которой особое внимание уделяется расширяемости и соответствуя стандартам. Как и MySQL, PostgreSQL использует модель данных клиент-сервер, а процесс сервера, который обрабатывает связь с клиентом, управляет файлами и операциями базы данных, называется процессом postgres. PostgreSQL обрабатывает параллельные клиентские сессии, «разветвляя» новый процесс для каждого соединения. Этот процесс отделен от основного процесса postgres, он создается и уничтожается во время жизненного цикла клиентского соединения. Написанный на C, Postgres также поддерживает ACID, функции и хранимые процедуры. В отличие от MySQL, PostgreSQL поддерживает материализованные представления (кэшированные представления), что ускоряет частый доступ к большим и активным таблицам. [8]

Преимущества:

- + Благодаря превосходной способности параллельной обработки PostgreSQL выходит на первое место (по сравнению с MySQL) при выполнении длинных SELECT.
- + Помимо встроенных функций, PostgreSQL поддерживает множество открытых сторонних инструментов для проектирования, управления данными и т. п. Одним из таких инструментов является веб-приложение pgAdmin.
- + База очень хорошо масштабируется и расширяется.
- + Работает быстро и надежно, база способна обрабатывать терабайты данных.
- + PostgreSQL всегда считался лучшим для аналитических процессов, таких как хранилище данных.

Недостатки:

- Механизм хранения PostgreSQL требует большой работы, он неоптимален в определенных случаях.
- PostgreSQL очень энергоемкий в сравнении с другими СУБД. Поскольку PostgreSQL разветвляет процесс для новых клиентских соединений, это может занять до 10 МБ на соединение. Эта модель может занимать много памяти при одновременном подключении клиента по сравнению с моделью MySQL для потокового подключения.
- Сложно устанавливать, много обновлений, не поддерживает кластеризованные индексы, что может оказаться огромное негативное влияние на производительность по сравнению с базами данных MySQL.

1.4 Анализ алгоритмов работы чат-бота

Чат-бот должен справляться со всеми задачами, описанными во введении посредством отправки вопросов, получения ответов на них и обработки полученной информации. Для обработки сообщений пользователей ботами на платформе мессенджера Telegram существует два вида алгоритмов: Webhooks — алгоритмы, работающие на основе механизмов оповещения системы о новых событиях и Long Polling — алгоритмы, работающие на основе опрашиваемых операций. Вне зависимости от используемого алгоритма все данные приходят в виде json-объектов, которые очень удобно обрабатывать. Данные алгоритмы будут рассмотрены ниже, после чего будет выбран наиболее подходящий для решения текущей задачи.

1.4.1 Long Polling — алгоритм опрашиваемых операций

Это самый распространенный и просто реализуемый алгоритм, который используется при написании чат-ботов на платформе Telegram. Суть этого алгоритма заключается в том, что бот периодически запрашивает обновления у Telegram, используя метод getUpdates [9], запрос должен отправляться не реже чем 1 раз в сутки, потому что дальше обновления не хранятся на серверах Telegram. В этом алгоритме важно сбалансировать скорость ответа бота и ресурсы, затрачиваемые на опрос серверов Telegram. Если слишком часто запрашивать обновления, скорость ответа будет высокой, но будет обрабатываться много бесполезных обновлений, не содержащих никакой новой информации. Если запрашивать обновления редко, будет очень большая задержка между отправкой сообщения пользователя и ответом бота.

1.4.2 Webhooks — алгоритм оповещения системы о событиях

В этом алгоритме нет необходимости для бота опрашивать серверы Telegram. Вместо этого сервера Telegram сообщают об обновлениях боту путем отправки в http-запросах структур данных, содержащих информацию о новых сообщениях. Webhook-и очень удобны, но также сложны в реализации. Для того, чтобы использовать этот алгоритм нужно иметь место на хостинге и зарегистрированный интернет домен. Кроме того, хостинг должен поддерживать протокол https и SSL-сертификат.

1.5 Обоснование выбора алгоритма чат-бота

Для данного курсового проекта алгоритм long polling будет более подходящим, потому как будет происходить обработка исключительно текстовых данных, а также для его использования нет необходимости регистрировать интернет домен, с этим алгоритмом бота можно развернуть со своего компьютера. В данном проекте основной задачей остается реализация базы данных, поэтому можно пренебречь скоростью обработки сообщений.

Вывод

В данном разделе были рассмотрены постановка задачи, общие сведения о БД и СУБД и алгоритм работы чат-ботов

2. Конструкторская часть

В данном разделе будет рассмотрено проектирование ПО, представлены диаграмма вариантов использования, ER диаграмма и диаграмма базы данных. Рассмотрена регистрация и аутентификация пользователей.

Необходимо создать модель работы системы, которая будет связывать приложение и базу данных. Все операции ввода-вывода связаны с обращением к базе данных. Обработка и получение сообщений – задача библиотеки и алгоритмов Telegram бота.

2.1 Диаграмма вариантов использования

Use-case диаграмма состоит из графической диаграммы, описывающей действующие лица и конкретные действия, которые выполняет пользователь при работе с системой. Данная диаграмма предназначена для определения функциональных требований. В системе есть три типа пользователей:

- Незарегистрированный пользователь(гость): пишет команду /start – его уникальный Telegram ID сохраняется в базу данных со статусом PREGISTRADED, на этом этапе пользователю предлагается ввести имя, остальные функции не доступны;
- зарегистрированный пользователь, гость: использует приложение для просмотра предстоящих событий и управления регистрацией на них, имеют тип пользователя GUEST и статус REGISTERED; переход в режим пользователя администратор осуществляется с помощью команды /admin и требует ввода пароля;
- зарегистрированный пользователь, администратор: использует приложение для редактирования и создания новых событий, просмотра статистики. Имеет тип пользователя ADMIN и статус REGISTERED.

На рисунке 2.1 представлена Use-Case диаграмма.



Рис. 2.1: Use-Case диаграмма

2.2 Конструирование базы данных

База данных включает в себя три таблицы, которые изображены на рисунке 2.2

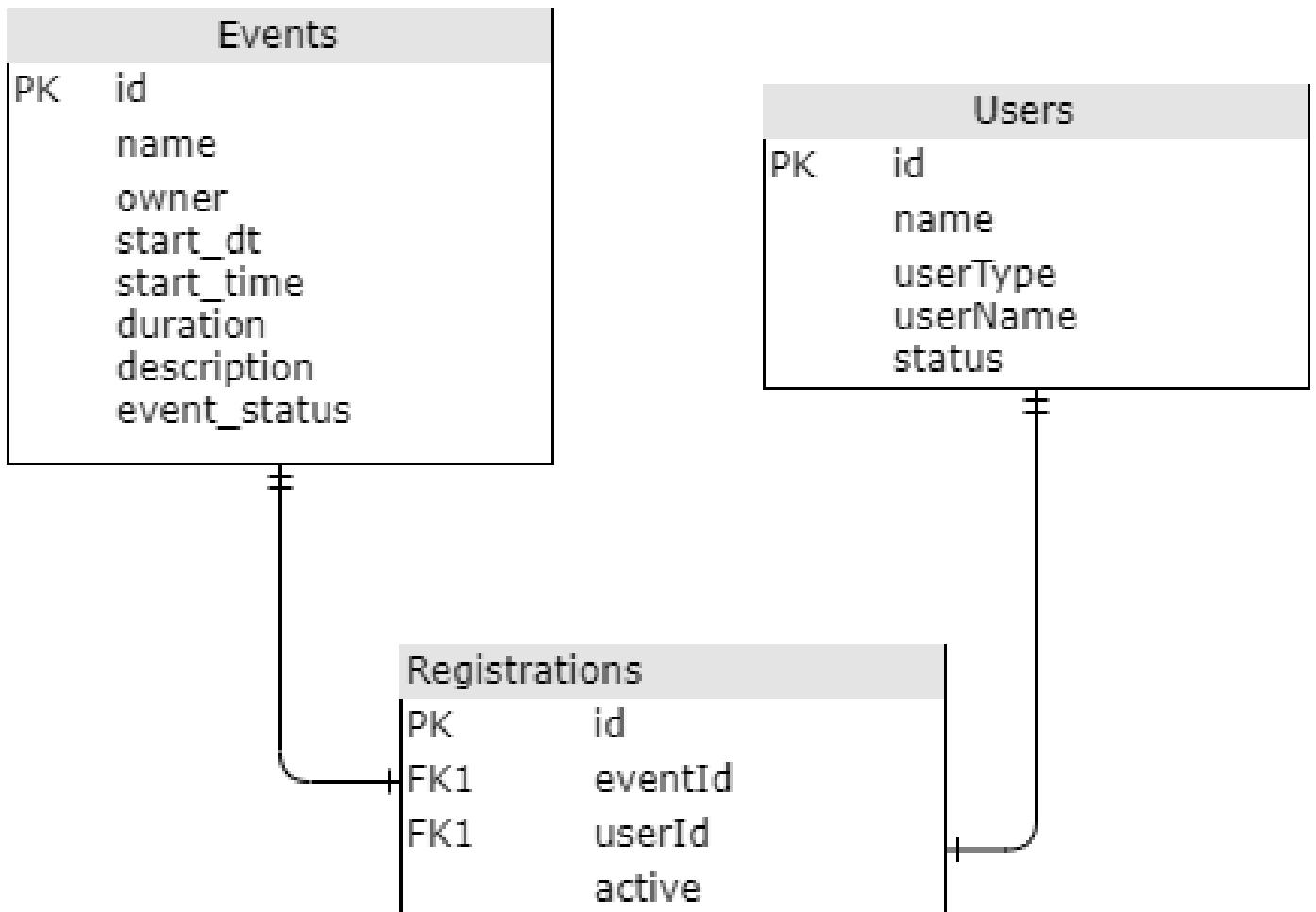


Рис. 2.2: Диаграмма таблиц базы данных

Таблица Users хранит информацию о пользователе и имеет следующие поля:

- id – идентификатор – primary key – в качестве него используется уникальный Telegram ID пользователя;
- name – имя гостя или администратора;
- userType – тип пользователя;
- userName – никнейм пользователя в Telegram (@nickname)
- status - статус пользователя (зарегистрирован, отключен и т.д.)

Таблица Events хранит информацию о событиях и имеет следующие поля:

- id – суррогатный primary key;
- name – название события;
- owner – никнейм владельца события;

- start_dt – дата проведения;
- start_time – время начала;
- duration – продолжительность;
- description – описание;
- event_status – статус (состоится, отменен и т.д.)

Таблица Registration хранит информацию о записях, тем самым связывая таблицы пользователей и событий, и имеет следующие поля:

- id – суррогатный primary key;
- eventId – идентификатор события – внешний ключ;
- userId – идентификатор пользователя – внешний ключ;
- status – статус регистрации (активна или отменена).

ER-диаграмма базы данных показана на рисунке 2.3. Было выделено три сущности. Сущность Appointment связана с сущностями Patient и Doctor связью один ко многим. Атрибуты сущностей также отображены на диаграмме.

2.2.1 Модель системы

Модель на вход получает сообщения от пользователя, они обрабатываются при помощи использования алгоритмов чат-бота, полученные данные, передаются в базу данных, данные, которые в последствии отправляются пользователю, тоже берутся из базы данных, приложение формирует ответное сообщение, оно отправляется пользователю при помощи библиотеки Telebot. Обращение к базе данных происходит каждый раз, когда требуется информация о врачах, пациентах или записях.

2.2.2 Структуры данных

В данном подразделе будут рассмотрены структуры данных, которые будут использованы в программе. В структуры входят:

- Объектно-реляционное отображение базы данных классами:
 - Класс пользователя User;
 - класс события Event;
 - класс записей Registration.
- Message_handlers – механизмы, позволяющие работать с сообщениями пользователя;
- callback_query_handlers – механизмы, позволяющие обрабатывать ввод пользователя с помощью inline buttons;
- классы библиотеки Telebot, содержащие информацию о сообщениях пользователя;
- классы клавиатур Keyboard, позволяющие выводить на экран пользователя кнопки с доступными вариантами ответов;

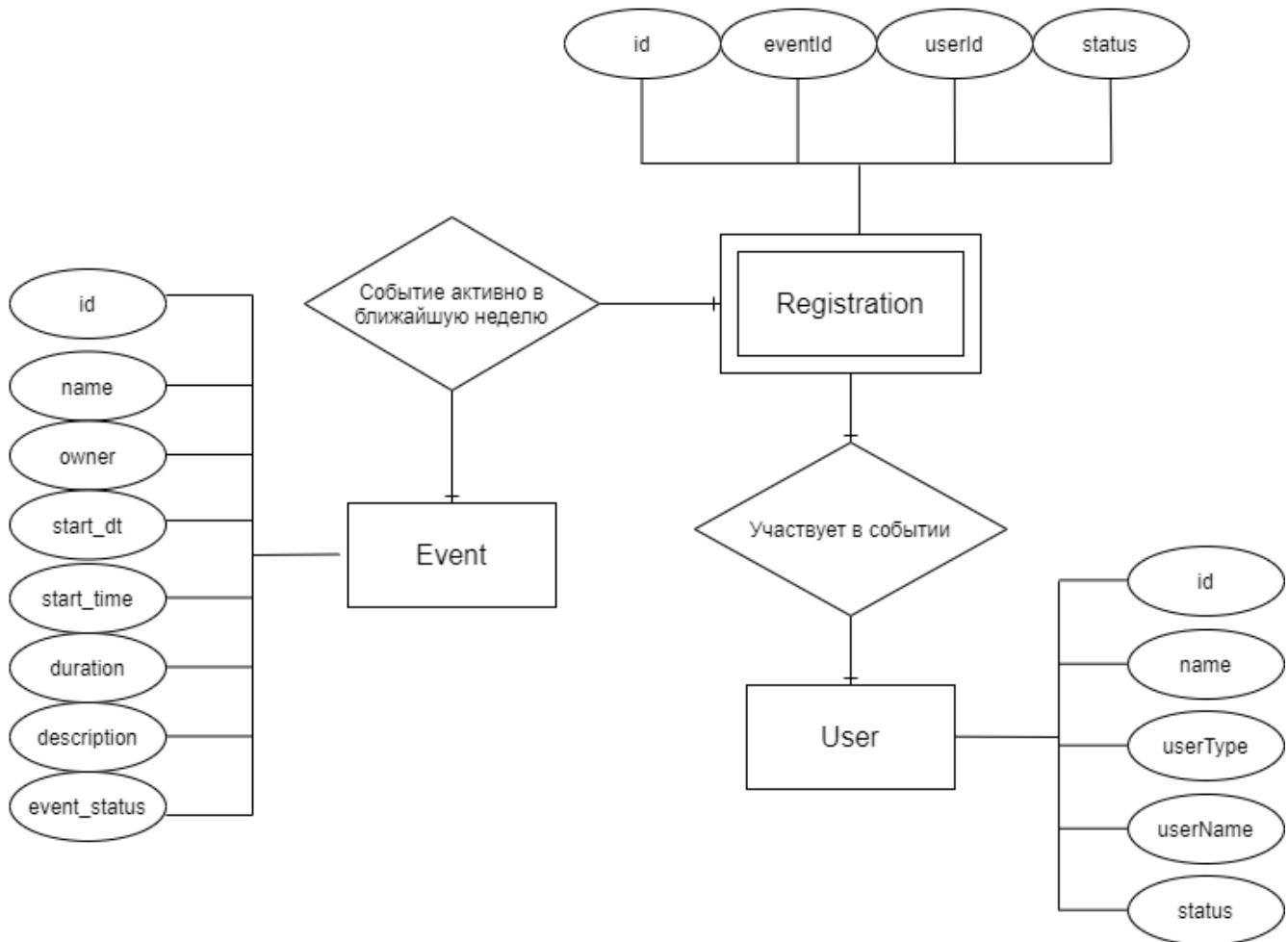


Рис. 2.3: ER-диаграмма модели

- классы констант такие как `UserStatus`, `UserType`, `EventStatus` и т.д.;
- классы `GuestChoiceButton` и `AdminChoiceButton` содержащие текстовые ответы на сообщения;
- статические параметры такие как `dateFormat`, `timeFormat`, `API-token` и т.д.

2.2.3 Модель данных

В данной работе будет использована технология ORM. ORM (англ. Object-Relational Mapping, рус. объектно-реляционное отображение) — технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». ORM позволяет удобно интегрировать модели в приложения с объектно-ориентированным стилем программирования.

Вывод

Были представлены диаграммы вариантов использования (так называемая User Case-диаграмма) и «сущность — связь», с помощью которых спроектирована база данных, рас-

смотрен атрибутный состав таблиц, архитектура приложения и модели данных, обеспечивающие связь приложение-база данных.

3. Технологическая часть

В данной части рассмотрены популярные СУБД и используемый фреймворк, приведены листинги классов для оформления таблиц базы данных, доступ к данным, а также рассмотрена интерфейс приложения.

3.1 Средства реализации

Для написания программного обеспечения для данного курсового проекта был использован язык Python 3.8. Этот язык поддерживает объектно-ориентированную модель разработки, что позволяет четко структурировать программу и легко модифицировать отдельные ее компоненты независимо от других. В качестве среды разработки был использован PyCharm 2020.1 (build 201.6668.115). Он обладает всем необходимым функционалом для написания, профилирования и отладки программ. Так как серверная программа не использует много ресурсов, достаточно одного потока для её работы.

3.2 Использование объектно-реляционного отображения

Для связывания серверной части и СУБД, было решено использовать объектно-реляционное отображение базы данных. Объектно-реляционное отображение — это технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Чтобы использовать этот метод работы с базой данных, было решено использовать классы и методы библиотеки Peewee. Для работы с таблицами базы данных было создано три класса — каждый класс являлся представлением таблицы на ООП. Классы представлены в листинге 3.2.1

```
1 class Events(Model):
2     id = AutoField(unique=True, null=False)
3     name = CharField()
4     owner = CharField()
5     start_dt = DateField(formats=[dateFormat])
6     start_time = TimeField(formats=[timeFormat])
7     duration = CharField()
8     description = CharField()
9     event_status = IntegerField(default=None)
10
11    class Meta:
12        db_table = 'events'
13        database = db
14
15    class Users(Model):
16        id = PrimaryKeyField(unique=True, null=False)
```

```

17     name = CharField(default="")
18     userType = IntegerField(-1)
19     userName = CharField(default="")
20     status = IntegerField(default=UserStatus.UNKNOWN)
21
22     class Meta:
23         db_table = 'users'
24         database = db
25
26
27 class Registrations(Model):
28     id = AutoField(unique=True, null=False)
29     eventId = ForeignKeyField(Events, backref='events')
30     userId = ForeignKeyField(Users, backref='users')
31     active = IntegerField()
32
33     class Meta:
34         db_table = 'registrations',
35         database = db

```

Листинг 3.2.1 – Классы пользователя, события и регистрации

Данные классы соответствуют описанной в конструкторском разделе базе данных. Класс Registrstion связан с классами Users и Events посредством внешних ключей (ForeignKeyField). Поля классов соответствуют полям базы данных из конструкторского раздела. Были выбраны типы данных, соответствующие данным, хранимым в этих полях.

Для обращения к базе данных были реализованы функции записи и получения данных. Добавление данных представлено в листинге 3.2.2.

```

1 def add_user(id, name="", userType=UserType.UNKNOWN, userName="", status=
              UserStatus.UNKNOWN):
2     Users.create(
3         id=id,
4         name=name,
5         userType=userType,
6         userName=userName,
7         status=status
8     )
9
10    return True
11
12 # true on success
13 def addRegistrationOnEvent(event_id, user_id) -> bool:
14     try:
15         Registrations.get(Registrations.eventId == event_id,
16                            Registrations.userId == user_id,
17                            Registrations.active == RegistrationStatus.ACTIVE
18                            )
19     except:
20         add_registration(event_id,
21                         user_id, RegistrationStatus.ACTIVE)
22         return True
23
24 def add_registration(eventId=0,
25                      userId=0, active=RegistrationStatus.INACTIVE):
26     app = Registrations.create(
27         eventId=eventId,
28

```

```

27         userId=userId,
28         active=active
29     )
30     app.save()
31     return True
32
33
34 def add_event(name="", owner="", start_dt="", start_time="",
35               duration="",
36               description="", event_status=EventStatus.UNKNOWN):
37     Events.create(
38         name=name,
39         owner=owner,
40         start_dt=start_dt,
41         start_time=start_time,
42         duration=duration,
43         description=description,
44         event_status=event_status
45     )
46
47     return True

```

Листинг 3.2.2 – Добавление записей

3.3 Реализация доступа к данным

В листинге 3.3.1 представлены функции запросов для получения данных из таблиц.

```

1 def getTodayEvents() -> str:
2     today = date.today().strftime(dateFormat)
3     return selectValidEventForTheDay(today)
4 def selectValidEventForTheDay(day):
5     todayEvents = Events.select()
6         .where(Events.start_dt==day,
7                 Events.event_status == EventStatus.
8                     GOINGTO)
9         .execute()
10    events = []
11    for line in todayEvents:
12        events.append(reply_event_markup.format(
13            line.name, line.start_dt,
14            line.start_time, line.duration,
15            line.owner, line.description))
16    return '\n'.join(events)
17
18 def selectEventNamesForTheDay(day, status=EventStatus.GOINGTO):
19     if status != EventStatus.UNKNOWN:
20         todayEvents = Events.select()
21             .where(Events.start_dt==day,
22                   Events.event_status== status).execute()
23     else:
24         todayEvents = Events.select()
25             .where(Events.start_dt == day)
26             .execute()
27    events = []
28    for line in todayEvents:

```

```

27     event = {"id": line.id,
28                "name": line.name,
29                "start_dt": line.start_dt,
30                "status": line.event_status,
31                "owner": line.owner,
32                "description": line.description,
33                "start_time": str(line.start_time)}
34     events.append(event)
35
36 return events
37
38 def selectUserActiveRegisteredEvents(id):
39     today = date.today()
40     query = (Registrations
41               .select(Registrations.id, Events.name, Events.id, Registrations
42                      .eventId)
43               .join(Events)
44               .where(Registrations.userId == id,
45                     Events.start_dt.year >= today.year,
46                     Events.start_dt.month >= today.month,
47                     Events.start_dt.day >= today.day,
48                     Registrations.active == RegistrationStatus.ACTIVE))
49
50     events = []
51     for line in query:
52         event = {"id": line.id,
53                   "name": line.eventId.name,
54                   "eventId": line.eventId.id
55         }
56
57         events.append(event)
58
return events

```

Листинг 3.3 – Классы пользователя, события и регистрации

3.4 Описание интерфейса чат-бота

Для начала работы пользователь должен дать команду /start (стандартное поведение для Telegram-бота), после чего ему предлагается ввести свое имя. Впоследствии имя будет использоваться для регистрации и может быть изменено программно. После введения имени пользователь получает доступ к блоку функций гостя (просмотр и регистрация на события). Переход пользователя к режиму администратора начинается после вызова пользователем команды /admin. Будет предложено ввести пароль (подразумевается, что он выдается сотрудникам) и в случае совпадения становятся доступны функции блока администратора — просмотр статистики, изменение и добавление новых событий. Оба типа пользователей имеют доступ к командам /menu — вызывает основное меню для соответствующего типа пользователя и /help — позволяет получить справочную информацию о боте и доступных командах.

На рисунке 3.1 представлена регистрация нового пользователя, основное и главное меню пользователя-гостя. На рисунке 3.2 показан интерфейс для просмотра пользователем событий - сегодня и в любой день текущей календарной недели и соответствующий ответ бота.

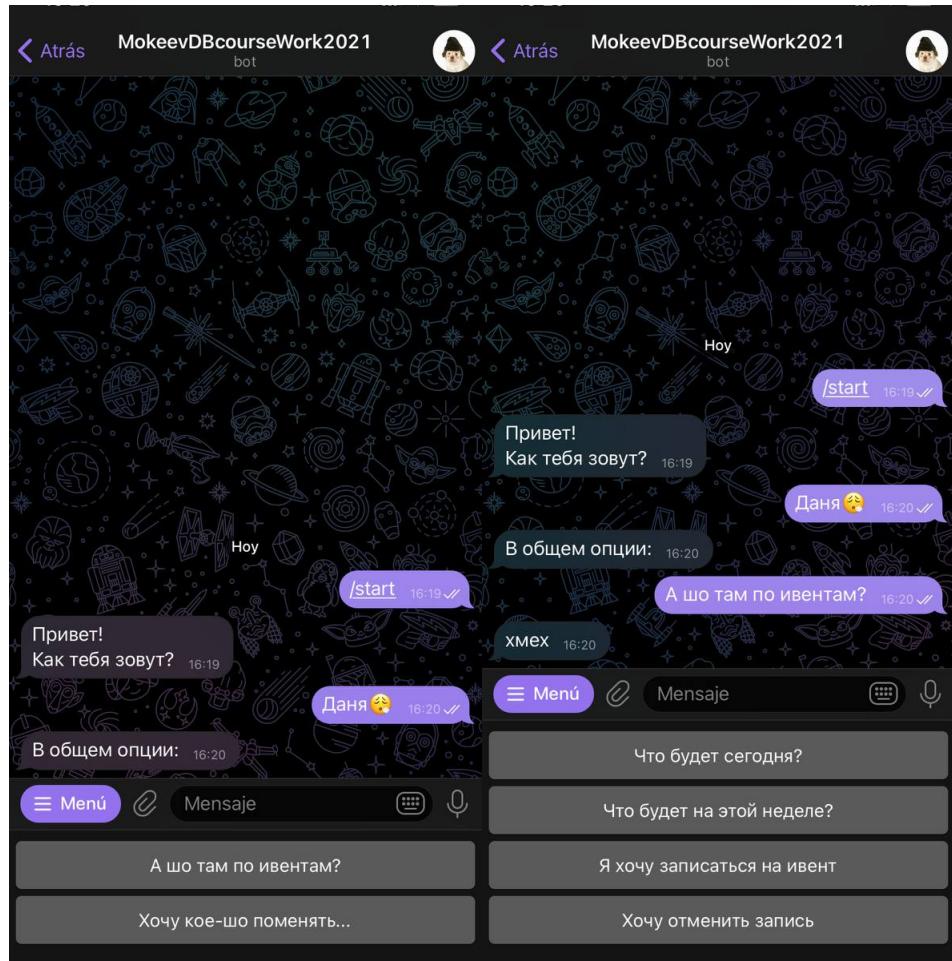


Рис. 3.1: Регистрация нового пользователя, основное и главное меню Пользователя-гостя

На рисунках 3.3 и 3.4 представлены интерфейс и ответы бота при регистрации и отмене регистрации пользователем.

На рисунке 3.5 показан интерфейс получения прав администратора и основное меню пользователя-администратора

3.5 Вывод

Были рассмотрены популярные СУБД и используемые алгоритмы работы чат-ботов, листинги реализованных классов для оформления таблиц базы данных. Был рассмотрен интерфейс приложения и его основные функции.

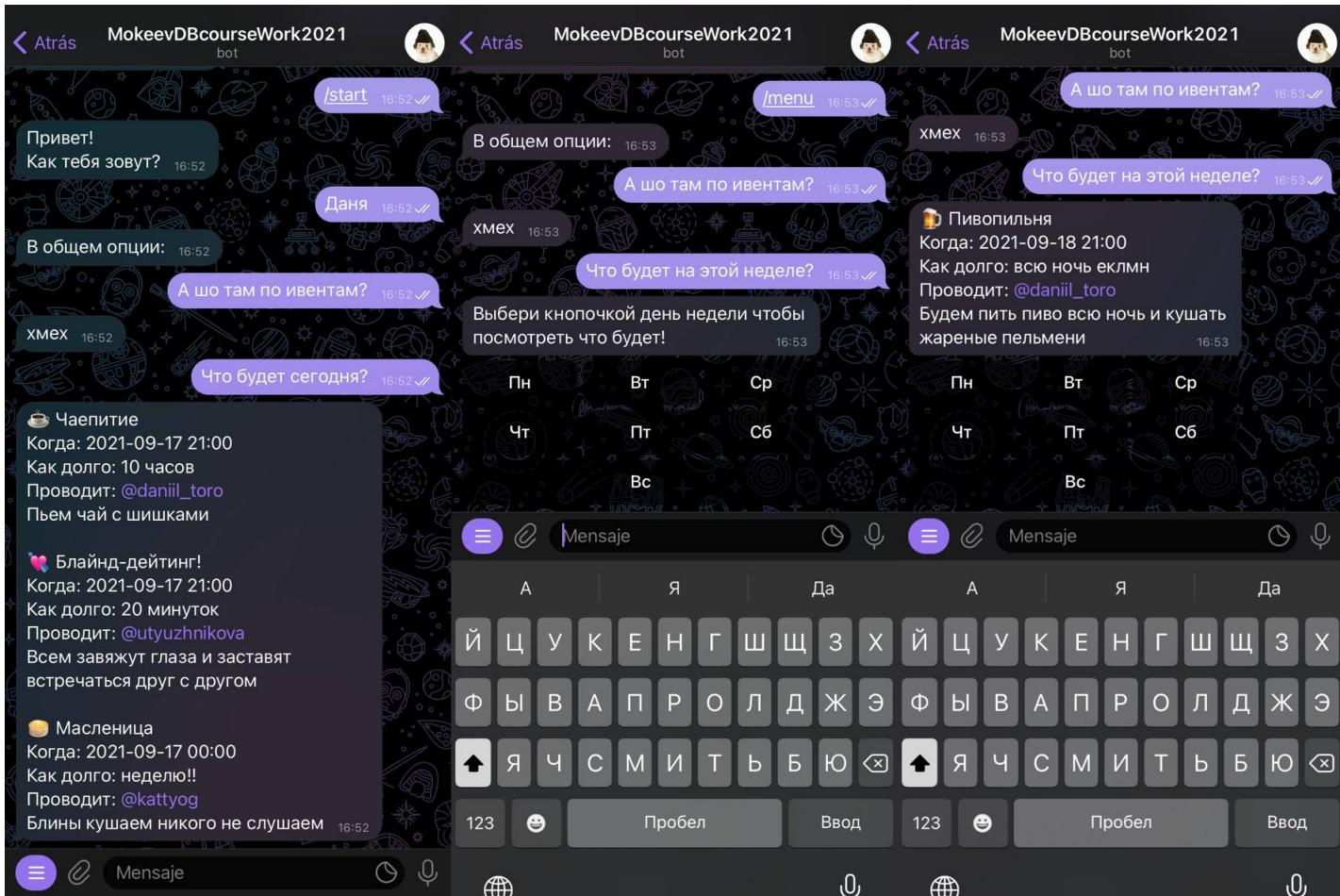


Рис. 3.2: Интерфейс для просмотра сегодняшнего события и событий недели

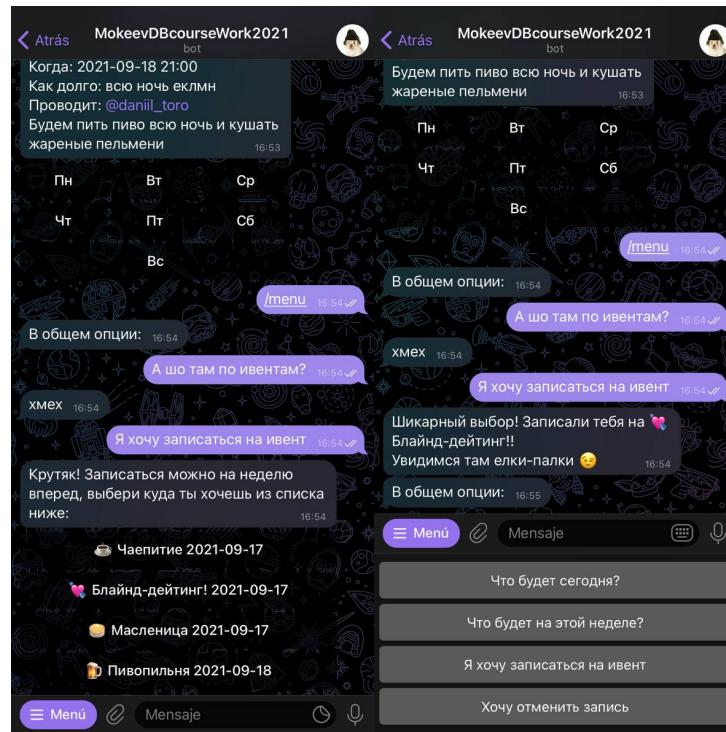


Рис. 3.3: Интерфейс регистрации на событие

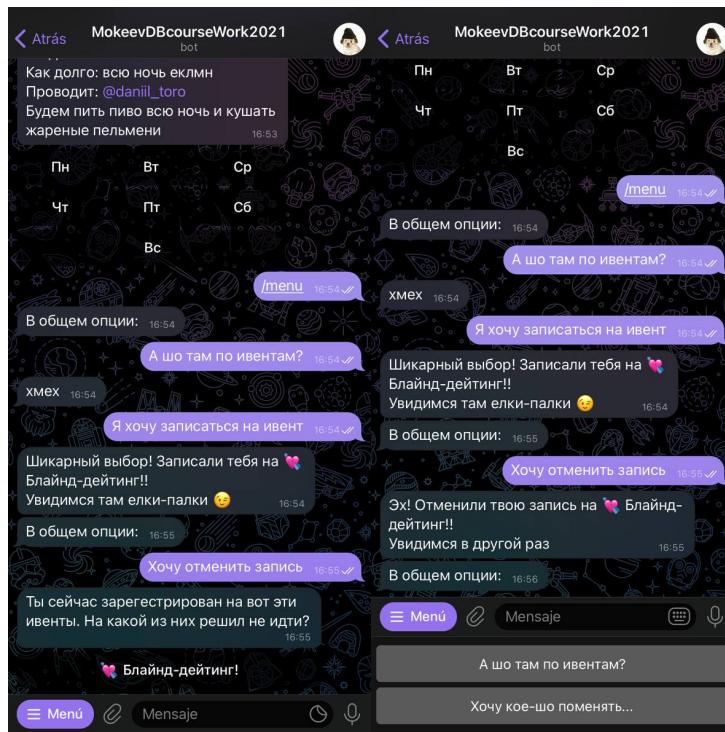


Рис. 3.4: Интерфейс отмены регистрации на событие

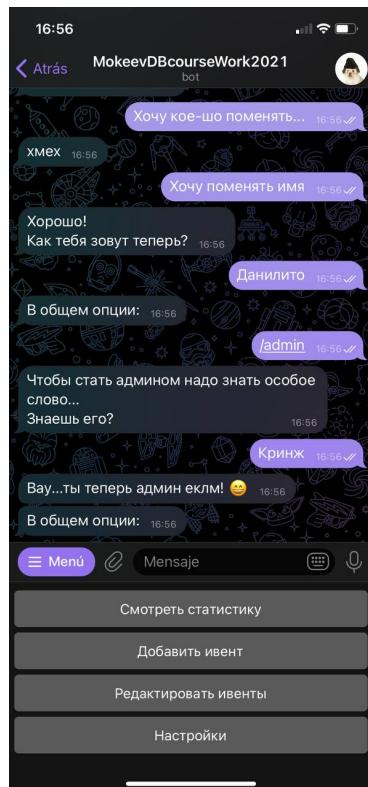


Рис. 3.5: Получение прав администратора и основное меню

Заключение

В результате выполнения курсового проекта была выполнена цель по разработке чат-бота на платформе мессенджера Telegram с подключением базы данных, задачей которого является обслуживание клиентов и сотрудников медицинских учреждений. Все запросы пользователей обрабатываются с использованием базы данных. Были выполнены задачи поставленные во введении. Был выполнен анализ СУБД и анализ алгоритмов работы чат-ботов на платформе Telegram. Было спроектировано программное обеспечение, созданы схемы и диаграммы. Была спроектирована архитектура базы данных. Выбранные алгоритмы и структуры были реализованы в полной мере. Были произведены тестирование и отладка программного обеспечения. В ходе данного курсового проекта были получены навыки создания чат-ботов на платформе Telegram, использования библиотеки Python Peewee для взаимодействия с СУБД SQLite.

В результате проделанной работы:

- формализована задача, определен необходимый функционал;
- проведён анализ инструментов, необходимых для проектирования и разработки приложения, в результате которого были выбраны СУБД SqlLight ORM Peewee;
- описана структуру базы данных, включая объекты, из которых она состоит;
- с помощью выбранных средств была создана платформа для управления событиями;

Цель работы достигнута, все задачи выполнены.

Список использованной литературы

- [1] Лекция № 3. Технологии баз данных — URL: <https://pandia.ru/text/78/475/50198.php> (дата обращения: 15.06.2021).
- [2] СУБД — URL: https://www.nic.ru/help/chto-takoe-subd_8580.html (дата обращения: 17.06.2021).
- [3] Дейт К. Дж. Введение в системы баз данных. — 8-е изд. — М.: «Вильямс», 2006.
- [4] Иерархическая СУБД [Электронный ресурс]. — Режим доступа: <https://www.tadviser.ru/index.php/> (Дата обращения: 18.06.2021)
- [5] Сетевые модели данных [Электронный ресурс]. — Режим доступа: https://studopedia.ru/220751setevie_modeli_dannih.html (Дата обращения: 18.06.2021)
- [6] Основные понятия реляционной модели данных [Электронный ресурс]. — Режим доступа: https://studopedia.ru/220752osnovnie_ponyatiya_relyatsionnoy_modeli_dannih.html (Дата обращения: 18.06.2021)
- [7] Гаврилова Юлия Михайловна, лекции из курса “Базы данных, ИУ7, 3 курс” — 2019 г.
- [8] Интернет-ресурс Logz.io — сравнение характеристик популярных СУБД [Электронный ресурс]. <https://logz.io/blog/relational-database-comparison/> (Дата обращения: 01.05.2020).
- [9] Документация библиотеки работы с Telegram API — Telebot [Электронный ресурс]. — Режим доступа: URL <https://core.telegram.org/bots/api> (Дата обращения: 01.05.2020).
- [10] Документация Python 3.8.3 Режим доступа: URL: <https://docs.python.org/3/> (Дата обращения: 01.05.2020).
- [11] Документация PyCharm [Электронный ресурс]. Режим доступа: URL <https://www.jetbrains.com/ru-ru/pycharm/documentation> (Дата обращения: 01.05.2020).
- [12] ORM Peewee [Электронный ресурс]. — Режим доступа: URL: <http://docs.peeweeorm.com/en/latest/index.html> (Дата обращения: 01.05.2020).
- [13] Документация СУБД SQLite [Электронный ресурс]. — Режим доступа: URL: <https://www.sqlite.org/index.html> (Дата обращения: 01.05.2020).