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 _____

КАФЕДРА _____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Сервис автоматизации регистрации участников на мероприятия.

Студент _____ _____ _____
(Группа) (Подпись, дата)
(И.О.Фамилия)

Руководитель курсовой работы _____ _____
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

Москва, 2024 г.

Содержание

ВВЕДЕНИЕ..... 4

1 Обзор предметной области..... 5

1.1 Обзор классических способов автоматизации регистрации на мероприятия..... 5

1.2 Преимущества telegram-бота для автоматизации регистрации..... 6

1.3 Моделирование данных с использованием модели «Сущность связь»..... 7

1.3.1 Требования к сервису регистрации..... 7

1.3.2 Модель «Сущность-связь»..... 8

2 Проектирование приложения..... 13

2.1 Разработка реляционной модели..... 13

2.2 Обеспечение правил минимальной кардинальности...... **23**

2.3 Разработка архитектуры приложения..... 29

2.3.1 Сервис авторизации..... 29

2.3.2 Сервис работы с ботами..... 30

3 Разработка приложения..... 32

3.1 Выбор технологических средств..... 32

3.2 Ключевые типы данных и функции..... 32

3.2.1 Сервис авторизации..... 32

3.2.2 Сервис работы с ботами..... 34

3.2.3 Интерфейс организатора..... 37

3.3 Руководство организатора..... 38

3.4 Руководство администратора..... 39

4 Тестирование..... 41

4.1 Разработка тестов 41

4.2 Результаты тестирования 42

5 Заключение 44

6 Список использованных источников 45

7 Приложение А 46

ВВЕДЕНИЕ

В условиях активной общественной жизни и большого числа мероприятий, проводимых в высших учебных заведениях, автоматизация регистрации участников становится важной частью эффективного управления событиями. В МГТУ имени Н. Э. Баумана ежегодно проводится более 500 мероприятий, и важно обеспечить оперативную регистрацию для студентов, преподавателей и гостей университета. Это позволяет существенно сократить временные затраты и повысить удобство для всех участников.

Автоматизация процесса регистрации предоставляет организаторам дополнительные возможности для управления массовыми мероприятиями. Важно иметь инструменты для эффективной обработки и систематизации данных о зарегистрированных участниках, что особенно актуально при большом объеме проводимых событий.

Таким образом, автоматизация регистрации в МГТУ имени Баумана становится необходимым инструментом для успешной организации мероприятий и повышения уровня удовлетворенности как участников, так и организаторов.

Целью работы является разработка системы автоматической регистрации участников мероприятий с использованием Telegram-ботов, позволяющей организаторам эффективно управлять событиями, минимизируя временные затраты на обработку данных и обеспечивая удобный интерфейс для пользователей.

1 Обзор предметной области

1.1 Обзор классических способов автоматизации регистрации на мероприятия

Автоматизация регистрации участников на мероприятия — ключевой элемент эффективного управления событиями, особенно в условиях большого количества мероприятий, как в случае с МГТУ им. Н. Э. Баумана. Существуют три основных подхода к автоматизации регистрации:

- Google или Yandex формы;
- Веб-сайты;
- Telegram-боты.

Google или Yandex формы обеспечивают простую настройку и быстрое создание регистрации, требуя лишь список вопросов для участников. Однако они не обладают высокой адаптивностью и ограничены в функциональности, что затрудняет их использование для различных категорий пользователей. Организация рассылок с помощью форм требует ручного написания сообщений, а внесение изменений не сопряжено с высокой трудоёмкостью. Эти формы являются удобным вариантом для небольших мероприятий, где функциональные требования ограничены.

Веб-сайты представляют более сложный инструмент для автоматизации регистрации. Они предоставляют гибкий и адаптивный интерфейс, который можно настраивать для различных групп участников. Организация рассылок на веб-сайтах не представляет особой сложности, если данный функционал интегрирован при разработке. Однако создание сайта и его дальнейшее обслуживание требуют значительных временных и технических ресурсов. Внесение изменений на веб-сайтах требует привлечения разработчиков, что может потребовать дополнительных временных затрат.

Telegram-боты проявляют себя как удобное решение для организаторов мероприятий, интегрируясь с платформой обмена сообщениями и предоставляя возможность автоматизации процессов. Хотя создание и поддержка Telegram-бота требуют значительных ресурсов на этапе разработки, его использование позволяет сократить время на регистрацию и

обработку данных в дальнейшем. Внесение изменений также может быть трудоёмким, так как требует модификации исходного кода. Однако высокая адаптивность и интеграция с внешними системами, такими как базы данных или API, делают Telegram-боты предпочтительным выбором для мероприятий с разными требованиями.

1.2 Преимущества telegram-бота для автоматизации регистрации

Автоматизация регистрации с использованием Telegram-бота представляет собой перспективное решение для организаторов массовых мероприятий. 31Одним из ключевых преимуществ данного подхода является интеграция бота с популярной платформой обмена сообщениями, что значительно упрощает процесс взаимодействия пользователей с системой. Благодаря этому, отпадает необходимость в интеграции дополнительного программного обеспечения или посещении внешних веб-ресурсов, что в свою очередь повышает доступность и удобство регистрации для участников.

Telegram-боты обладают высокой степенью адаптивности, предоставляя организаторам широкие возможности по настройке сбора данных в соответствии с конкретными требованиями мероприятия. Данный инструмент позволяет не только автоматизировать процесс регистрации, но и упрощает организацию рассылок, что способствует значительному снижению трудозатрат на коммуникации с участниками. Немаловажным преимуществом является возможность интеграции Telegram-бота с внешними системами, такими как базы данных или сторонние API, что облегчает хранение и обработку информации, необходимой для успешной организации мероприятий.

Несмотря на то, что создание Telegram-бота требует первоначальных затрат ресурсов, данный метод позволяет существенно сократить временные и трудовые затраты на последующих этапах регистрации и обработки данных. Боты также обладают высокой масштабируемостью, что делает их универсальным инструментом для регистрации на мероприятия различного масштаба. Благодаря адаптивности бота, он может эффективно работать с

различными категориями пользователей, предлагая гибкие решения для различных форматов мероприятий.

Таким образом, Telegram-боты представляют собой мощный инструмент для автоматизации регистрации на мероприятия. Успешная реализация и модификация таких решений позволит создать универсальный сервис, обеспечивающий высокую гибкость, удобство и степень автоматизации, что повышает эффективность проведения мероприятий и удовлетворенность как участников, так и организаторов.

1.3 Моделирование данных с использованием модели «Сущность связь»

1.3.1 Требования к сервису регистрации

На основании существующих мероприятий можно выделить две основные сущности любого мероприятия:

- участник мероприятия;
- организатор мероприятия.

С точки зрения участника сервис должен представлять telegram бот-анкету, содержащую конечный набор вопросов, характеризующих участника.

Организатору, в свою очередь, необходимо предоставить следующие возможности для проведения регистрации:

- генерация telegram-бота по заранее описанной схеме;
- создавать несколько telegram-ботов;
- получать доступ к ранее созданным telegram-ботам;
- в произвольный момент времени включить или выключить ранее созданный telegram бот;
- проведение рассылок зарегистрированным участникам мероприятия.

При описании схемы telegram-бота организатору необходимо предоставить следующие возможности:

- создание произвольного количества элементов регистрации одного из следующих типов: «Сообщение», «Выбор», «Вопрос»;

- создание ацикличного ветвления между любыми элементами регистрации;
- создание произвольного количества точек входа в telegram-бот;
- прикрепление произвольного текстового сообщения, присылаемого участник в telegram, к каждому элементу регистрации;
- создание произвольного количества кнопок, содержащих любой текст для каждого элемента регистрации типа «Выбор».

1.3.2 Модель «Сущность-связь»

В ER модель включено восемь сущностей:

1) user — сущность, характеризующая пользователя, который создаёт бота, т.е. представителя студенческой организации, проводящей мероприятие. Идентификатор сущности «email», представляющий собой адрес электронной почты владельца, атрибуты сущности:

- passhash – пароль в хэшированном представлении;
- created_at – дата создания аккаунта;
- updated_at – дата обновления аккаунта.

2) bot — сущность, характеризующая генерируемого telegram бота с идентификатором «token», представляющим собой уникальный идентификатор, предоставляемый Telegram Bot API и атрибутами:

- name – название бота;
- status – состояние бота, представляет собой элемент перечисления: “Включен”, “Выключен”, “Ошибка запуска”.
- created_at – дата создания;
- updated_at – дата обновления.

3) block — сущность, представляющая элемент регистрации с идентификатором state, представляющим собой порядковый номер элемента регистрации в боте. Атрибуты сущности:

- type – атрибут, определяющий тип элемента регистрации - элемент перечисления из трёх элементов: “Вопрос”, “Сообщение”, “Выбор”;

- next_state – указатель на следующий элемент регистрации, не пустой в случае непустого списка кнопок у элемента регистрации;
- title – название элемента регистрации;
- text – текст сообщения, выводимого конечному пользователю в Telegram.

4) option — сущность, представляющая собой кнопку для элемента регистрации с идентификатором option_id, представляющий собой суррогатный ключ, с атрибутами:

- text - значение, которое будет считаться ответом конечного пользователя после нажатия на кнопку также является текстом, находящимся на кнопке;
- next - указатель на элемент регистрации, который следует после нажатия на данную кнопку.

5) answer — сущность, представляющая собой ответ конечного пользователя на вопрос регистрации с идентификатором answer_id, представляющим собой суррогатный ключ и единственным атрибутом:

- text - текст ответа на вопрос;

6) participant — сущность, представляющая собой конечного участника мероприятия с идентификатором “user_id”, представляющий собой уникальный идентификатор, предоставляемый telegram API.

7) entry_point – сущность, представляющая собой точки входа в telegram бот - абстракции на событие «Telegram-бот самостоятельно пишет сообщение конечному участнику мероприятия». Идентификатор сущности key – уникальный текстовый ключ. Атрибут сущности:

- state – указатель на элемент регистрации, выведенный пользователю.

8) mailing - сущность, представляющая собой рассылку внутри telegram бота с идентификатором mailing_id, представляющим собой суррогатным ключ и атрибутами:

- name – название рассылки;

– required_state – условие для получения пользователем рассылки, представляет собой указатель на элемент регистрации на который пользователь обязан дать ответ для получения рассылки.

Между описанными сущностями были построены связи, согласующиеся с особенностями предметной области, описанных в пункте 1.3.1.

Организатор мероприятия, представляемый сущностью users, может создавать произвольное количество ботов для регистрации, но может и не создавать их вовсе. В то же время бот не может существовать без привязки к организатору. Таким образом, между сущностями users и bots образуется связь "один ко многим", а минимальные кардинальные числа равны 1 и 0 соответственно.

Каждый telegram-бот можно однозначно представить в виде непустого набора элементов регистрации, представленных сущностью blocks. Блоки являются обязательными для существования бота, так как его функциональность заключается во взаимодействии через блоки регистрации. Каждый блок, в свою очередь, всегда принадлежит какому-то боту. Между сущностями bots и blocks образуется связь "один ко многим" с минимальными кардинальными числами 1 и 1 соответственно.

Каждый элемент регистрации, представленный сущностью blocks, может содержать несколько кнопок для взаимодействия с пользователем, которые представлены сущностью options. Однако блок может не иметь кнопок, если это, например, информационное сообщение, а каждая кнопка должна принадлежать какому-то блоку, поскольку ее назначение связано с выбором в контексте элемента регистрации. Между сущностями blocks и options образуется связь "один ко многим" с минимальными кардинальными числами 1 и 0 соответственно.

Элементы регистрации связаны друг с другом последовательно: каждый блок ведет к ровно одному следующему блоку или является конечным. Ветвление реализуется через сущность options. Таким образом, между элементами регистрации (сущность blocks) существует рекурсивная связь

"один к одному" с минимальными кардинальными числами 1 и 0 соответственно.

Каждый участник мероприятия, представленный сущностью participants, может давать несколько ответов на вопросы регистрации, представленных сущностью answers. Ответы не могут существовать без участника, так как они привязаны к его действиям. В то же время участник может существовать без данных ответов, если он, например, еще не завершил регистрацию. Таким образом, между сущностями participants и answers образуется связь "один ко многим" с минимальными кардинальными числами 1 и 0 соответственно.

Каждый участник мероприятия, представленный сущностью participants имеет в каждый момент времени может работать ровно с одним элементом регистрации, либо пройти все элементы регистрации конкретного бота (завершить регистрацию). Таким образом между сущностями participants и blocks образуется связь "один к одному" с минимальными кардинальными числами 1 и 0 соответственно.

Каждый блок регистрации, представленный сущностью blocks, может существовать без ответа, но, если ответ существует, он всегда привязан к какому-то блоку. Таким образом, между сущностями blocks и answers образуется связь "один ко многим" (один блок ко многим ответам) с минимальными кардинальными числами 1 и 0 соответственно.

Каждый telegram-бот должен иметь хотя бы одну точку входа, как минимум пользовательскую команду "/start", которая инициализирует взаимодействие. Точки входа всегда связаны с ботом, так как их функциональность заключается во взаимодействии через бот. Таким образом, между сущностями bots и entry_points существует связь "один ко многим" с минимальными кардинальными числами 1 и 1 соответственно.

Точка входа, представленная сущностью entry_points, может быть связана с одной конкретной рассылкой, которая инициируется в этой точке. Рассылка не может существовать без привязки к точке входа, а точка входа всегда ведет к одной рассылке. Между сущностями entry_points и mailings образуется связь

"один к одному" с минимальными кардинальными числами 1 и 1 соответственно.

Каждая кнопка, представленная сущностью options, может вести к ровно одному следующему элементу регистрации. Кнопка завершает взаимодействие или переводит пользователя к следующему блоку. Таким образом, между сущностями options и blocks существует связь "многие к одному" с минимальными кардинальными числами 0 и 1 соответственно.

Каждой рассылке, представленной сущностью mailings, соответствует ровно одно условие прохождения рассылки, отсутствие условия обозначает, что рассылку должны получить все пользователи бота. Таким образом, между сущностями mailings и blocks образуется связь "один к одному" с минимальными кардинальными числами 1 и 0 соответственно.

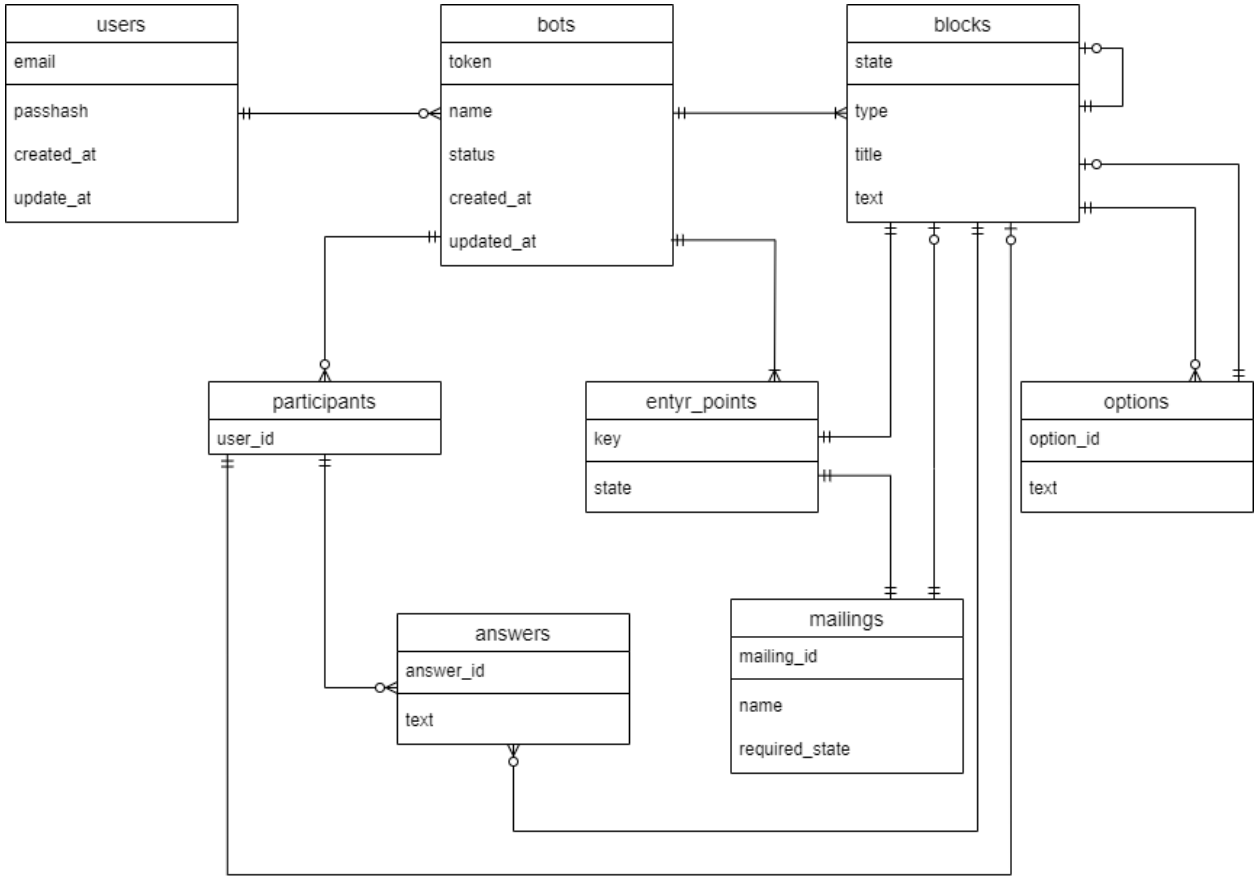

Рисунок 1. Модель «Сущность-связь»

2 Проектирование приложения

2.1 Разработка реляционной модели

Для проектирования реляционной модели использовалась модель «сущность-связь», созданная в рамках, раздела 1.3. Она представлена на рисунке 1.

В соответствии с правилами преобразования, была получена реляционная модель, представленная на рисунке 2.

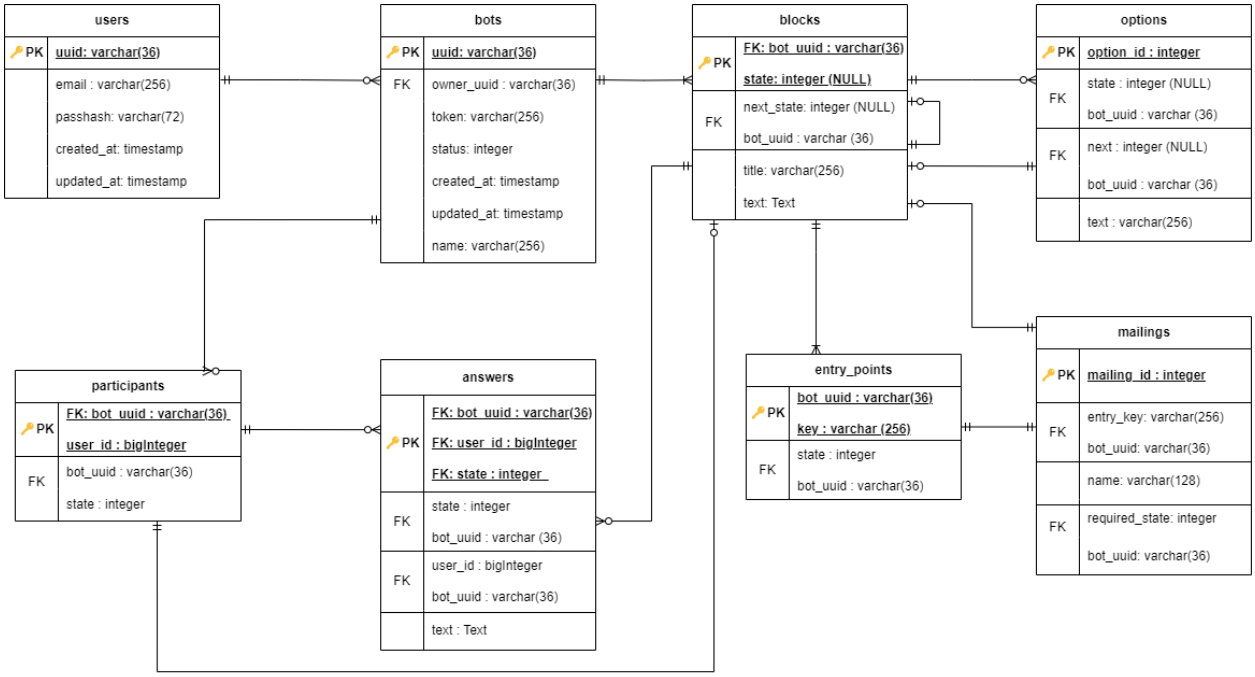

Рисунок 2. Реляционная модель

Были спроектированы таблицы, описывающие отношения реляционной модели:

Таблица 1. Отношение «users»

Column name	**Type**	**Key**	**Null status**	**Remarks**
uuid	Varchar(36)	PK	Not null	Суррогатный строковый ключ

email	Varchar(256)	Candidate Key	Not null	Уникальный ключ (AK 1.1)
password_hash	Varchar(256)	No	Not null	Пароль в хэшированном представлении
created_at	TimeStamp	No	Not Null	Время создания аккаунта
updated_at	TimeStamp	No	Not Null	Время обновления аккаунта, по умолчанию равняется времени создания.

Таблица 2. Отношение «bots»

Column name	**Type**	**Key**	**Null status**	**Remarks**
uuid	Varchar(36)	PK	Not null	Суррогатный строковый ключ
owner_uuid	Varchar(36)	FK	Not null	Внешний ключ, указывающий

				на создателя бота.
name	Varchar(256)	No	Not null	Название бота
token	Varchar(256)	Candidate key	Not Null	Уникальный ключ (AK1.1), связывает бота с telegram API
created_at	TimeStamp	No	Not Null	Время создания аккаунта
updated_at	TimeStamp	No	Not Null	Время обновления аккаунта, по умолчанию равняется времени создания.

Таблица 3. Отношение «particpants»

Column name	**Type**	**Key**	**Null status**	**Remarks**
user_id	BigInteger	PK	Not null	Идентификатор пользователя, предоставляемый

				telegram. Часть составного первичного ключа
bot_uuid	VarChar(36)	FK	Not null	Внешний ключ, указывающий на бот, с которым работает участник. Часть составного первичного ключа. Так же является частью внешнего составного ключа отношением blocks
state	integer	No	Not null	Указатель на текущий элемент регистрации пользователя. Является частью составного внешнего ключа отношения blocks

Таблица 4. Отношение «blocks»

Column name	**Type**	**Key**	**Null status**	**Remarks**
State	integer	PK	Null	Порядковый номер блока в боте. Часть составного первичного ключа.
bot_uuid	VarChar(36)	FK	Not null	Внешний ключ – ссылка на бот. Часть составного первичного ключа.
text	Text	No	Not null	Текстовое сообщение элемента регистрации
Title	VarChar(256)	No	Not null	Название элемента регистрации
Next_state	integer	FK	Null	Внешний ключ, образующий рекурсивную связь. Последовательность ссылок блоков друг на друга. NULL в этом контексте означает, что данный блок является одним из

				множества конечных блоков.
type	integer	No	Not null	Элемент перечисления: «Вопрос», «Сообщение», «Выбор».

Таблица 5. Отношение «mailings»

Column name	**Type**	**Key**	**Null status**	**Remarks**
Mailing_id	integer	PK	Not null	Суррогатный ключ
Bot_uuid	VarChar(36)	FK	Not null	Часть составного ключа отношения entry_points.
entry_key	VarChar(128)	FK	Not null	Часть составного внешнего ключа отношения entry_points.
required_state	integer	FK	NULL	Состояние после прохождения,

				которого пользователь может получить сообщение. Часть составного внешнего ключа отношения blocks.

Таблица 6. Отношение «entry_points»

Column name	**Type**	**Key**	**Null status**	**Remarks**
Bot_uuid	VarChar(36)	FK	Not null	Внешний ключ отношения bots. Часть составного первичного ключа.
key	Varchar(256)	PK	Not null	Текст команды для входа в бот. Часть

				составного первичного ключа.
state	integer	FK	Not null	Внешний ключ, ссылка на блок, в который ведёт команда входа. Часть составного внешнего ключа отношения blocks.

Таблица 7. Отношение «options»

Column name	**Type**	**Key**	**Null status**	**Remarks**
Option_id	integer	PK	Not null	Суррогатный ключ.
Bot_uuid	VarChar(36)	FK	Not null	Часть составного внешнего ключа отношения blocks.

state	Integer	FK	Not null	Ссылка на блок, к которому прикреплена данная кнопка. Часть составного внешнего ключа отношения blocks.
text	VarChar(128)	No	Not null	Текст, указанный на данной кнопке.
next	integer	FK	Null	Указывает на блок, к которому необходимо перейти после нажатия на кнопку. Null означает, что данная кнопка ведёт к завершению взаимодействия с пользователем. Часть внешнего ключа

				отношения blocks.

Таблица 8. Отношение «answers»

Column name	**Type**	**Key**	**Null status**	**Remarks**
Bot_uuid	VarChar(36)	FK	Not null	Внешний ключ, ссылка на бот, который содержит в себе ответ. Часть составного первичного ключа.
User_id	BigInteger	FK	Not null	Внешний ключ, ссылка на участника мероприятия, который отвечает на вопрос. Часть составного первичного ключа.
state	Integer	FK	Not null	Внешний ключ, ссылка на элемент

				регистрации, на который пользователю необходимо дать ответ. Часть составного первичного ключа.
text	Text	No	Not null	Текстовое представление ответа пользователя.

2.2 Обеспечение правил минимальной кардинальности.

Таблица 9. Обеспечение минимальной кардинальности связи отношений bot_owner и bots

Операция	Родительская таблица (users)	Дочерняя таблица (bots)
Вставка	Без ограничения	Подбор родительской записи
Изменение первичного или внешнего ключа	Запрещено	Запрещено
Удаление	Каскадное удаление	Без ограничения

Таблица 10. Обеспечение минимальной кардинальности связи отношений bots и blocks

Операция	Родительская таблица (bots)	Дочерняя таблица (blocks)
Вставка	Без ограничения	Подбор родительской записи
Изменение первичного или внешнего ключа	Запрещено	Запрещено
Удаление	Каскадное удаление	Без ограничения, если удаление не нарушает достижимости элементов регистрации и у бота остаётся хотя бы один элемент регистрации.

Таблица 11. Обеспечение минимальной кардинальности связи отношений bots и participants

Операция	Родительская таблица (bots)	Дочерняя таблица (participants)
Вставка	Без ограничения	Подбор родительской записи
Изменение первичного или внешнего ключа	Запрещено	Запрещено
Удаление	Каскадное удаление	Без ограничения

Таблица 12. Обеспечение минимальной кардинальности связи отношений blocks и options

Операция	Родительская таблица (blocks)	Дочерняя таблица (options)
Вставка	Без ограничения	Подбор родительской записи
Изменение первичного или внешнего ключа	Запрещено	Запрещено
Удаление	Каскадное удаление	Без ограничения, если при удалении в элементе регистрации остаётся хотя бы одна кнопка.

Таблица 13. Обеспечение минимальной кардинальности связи отношений blocks и blocks

Операция	Родительская таблица (blocks)	Дочерняя таблица (blocks)
Вставка	Без ограничения	Подбор родительской записи
Изменение первичного или внешнего ключа	Запрещено	Запрещено
Удаление	Каскадное удаление	Без ограничения

Таблица 14. Обеспечение минимальной кардинальности связи отношений blocks и mailings

Операция	Родительская таблица (bots)	Дочерняя таблица (*mailings*)
Вставка	Без ограничения	Подбор родительской записи
Изменение первичного или внешнего ключа	Запрещено	Запрещено
Удаление	Каскадное удаление	Без ограничения

Таблица 15. Обеспечение минимальной кардинальности связи отношений blocks и entry_points

Операция	Родительская таблица (blocks)	Дочерняя таблица (entry_points)
Вставка	Без ограничения	Подбор родительской записи
Изменение первичного или внешнего ключа	Запрещено	Запрещено
Удаление	Каскадное удаление	Разрешено, если у бота остаётся хотя бы одна точка входа.

Таблица 16. Обеспечение минимальной кардинальности связи отношений blocks и participants

Операция	Родительская таблица (blocks)	Дочерняя таблица (participants)

Вставка	Без ограничения	Подбор родительской записи
Изменение первичного или внешнего ключа	Запрещено	Запрещено
Удаление	Запрещено	Без ограничения

Таблица 17. Обеспечение минимальной кардинальности связи отношений blocks и answers

Операция	Родительская таблица (blocks)	Дочерняя таблица (answers)
Вставка	Без ограничения	Подбор родительской записи
Изменение первичного или внешнего ключа	Запрещено	Запрещено
Удаление	Каскадное удаление	Без ограничения

Таблица 18. Обеспечение минимальной кардинальности связи отношений participants и answers

Операция	Родительская таблица (particpants)	Дочерняя таблица (answers)
Вставка	Без ограничения	Подбор родительской записи
Изменение первичного или внешнего ключа	Запрещено	Запрещено
Удаление	Каскадное удаление	Без ограничения

Таблица 19. Обеспечение минимальной кардинальности связи отношений entry_points и mailings

Операция	Родительская таблица (entry_points)	Дочерняя таблица (mailings)
Вставка	Без ограничения	Подбор родительской записи
Изменение первичного или внешнего ключа	Запрещено	Запрещено
Удаление	Без ограничения	Без ограничения

Таблица 20. Обеспечение минимальной кардинальности связи отношений options и blocks

Операция	Родительская таблица (options)	Дочерняя таблица (blocks)
Вставка	Разрешено, если значение внешнего ключа соотноситься с сущностью blocks в рамках одного бота.	Подбор родительской записи или NULL.
Изменение первичного или внешнего ключа	Запрещено	Запрещено
Удаление	Каскадное удаление, если элемент регистрации не ведёт ни одна другая ветвь внутри бота.	Без ограничения

2.3 Разработка архитектуры приложения

В разрабатываемом сервисе можно выделить три основные компонента:

1. Сервис авторизации;
2. Сервис работы с ботами;
3. Интерфейс организатора.

На рисунке 3 представлена схема архитектуры приложения:

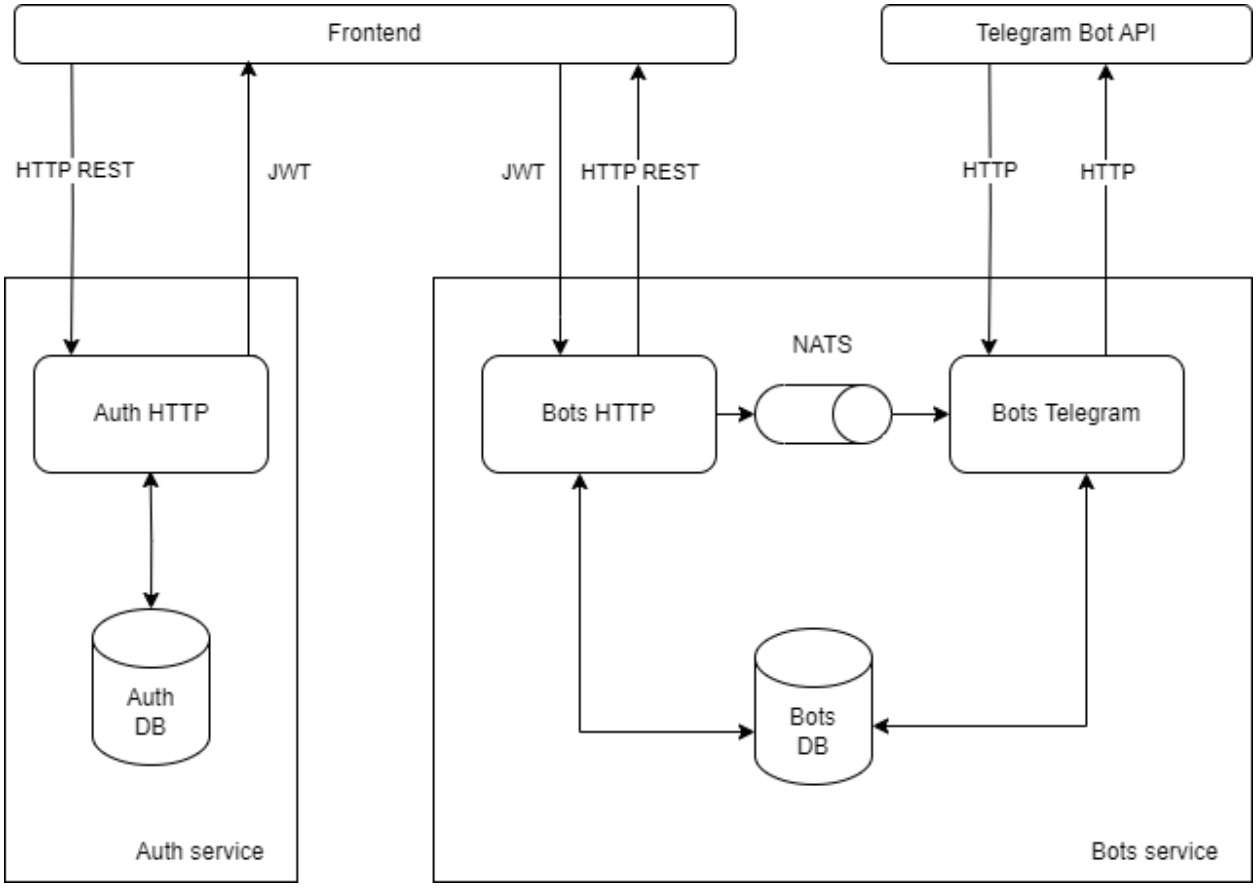

Рисунок 3 - Архитектура приложения.

2.3.1 Сервис авторизации

Сервис авторизации играет ключевую роль в обеспечении безопасности системы и управления доступом организаторов к сервису. Он включает собственную базу данных с таблицей "users", которая хранит информацию об организаторах мероприятий (описана в разделе 1.3.2). Основная задача сервиса заключается в проверке существующих организаторов или регистрации новых.

Процесс авторизации основан на проверке учетных данных организатора. Если организатор уже зарегистрирован, сервис выполняет поиск по базе данных, чтобы подтвердить его права доступа. Если же организатор новый, то сервис осуществляет регистрацию, записывая необходимые данные в таблицу. После успешной аутентификации организатор получает JWT (JSON Web Token) — токен, который используется для дальнейшего взаимодействия с системой. JWT-токен содержит закодированную информацию об организаторе и его правах доступа, что позволяет передавать его с каждым запросом для идентификации пользователя без необходимости повторной аутентификации. Этот подход обеспечивает безопасность и удобство работы с системой.

2.3.2 Сервис работы с ботами

Сервис работы с ботами является центральным компонентом системы, обеспечивающим управление и взаимодействие с Telegram-ботами, которые выполняют процесс регистрации участников. Данный сервис выполняет несколько ключевых функций:

- Прием запросов от организаторов;
- Взаимодействие с Telegram API;
- Функция сервера для ботов.

После авторизации через JWT организаторы могут отправлять HTTP-запросы к сервису для управления ботами. Эти запросы могут включать действия по созданию ботов, управлению их состояниями, настройке сценариев регистрации и мониторингу активности участников. Каждый запрос проверяется на наличие корректного JWT-токена, что обеспечивает безопасность и предотвращает несанкционированный доступ к боту.

Сервис работает в качестве посредника между организаторами и Telegram. Он отправляет HTTP-запросы к открытому API Telegram Bot, обрабатывая команды, полученные от организаторов, и отправляя их в Telegram. Этот процесс включает отправку сообщений участникам, получение их ответов и передачу данных обратно в систему.

Сервис не только управляет ботами, но и выполняет роль сервера для активных Telegram-ботов. Это означает, что все взаимодействия между участниками мероприятий и ботами проходят через этот сервис. Он обрабатывает команды, полученные от пользователей через Telegram, и обеспечивает их корректную обработку в контексте регистрационного сценария.

3 Разработка приложения

3.1 Выбор технологических средств

Для разработки технического решения был выбран язык программирования «Go» версии 1.22. Данный язык представляет собой распространённый инструмент для разработки высоконагруженных серверных приложений, обеспечивая высокий уровень производительности и предоставляя разработчикам широкий ряд возможностей и сторонних библиотек.

Для разработки хранилища данных использовалась СУБД с открытым исходным кодом PostgreSQL. Данная СУБД отличается высокой производительностью и гибкостью, что делает ее привлекательной для серверных приложений с высокой нагрузкой и требующими надежного управления данными.

Для разработки интерфейса организатора был выбран язык C# на платформе «.NET 8.0», которая является мощным инструментом для разработки высоконагруженных серверных приложений, обеспечивая отличную производительность и высокую масштабируемость. Платформа **.NET 8.0** предлагает множество встроенных возможностей для асинхронной обработки, параллельного выполнения задач и управления потоками, что позволяет эффективно работать с большими объемами данных и высокой нагрузкой.

3.2 Ключевые типы данных и функции

3.2.1 Сервис авторизации

Точкой входа в приложение является файл “http.go”, находящийся в директории “cmd/http” и запускающий сервер HTTP по указанному в переменных среды порту. Сервер представляет собой реализацию паттерна REST (Representational State Transfer) API, реализующее три HTTP метода:

- **Register User** – POST метод, осуществляющий регистрацию нового организатора. Входными данными метода является JSON-объект, представляющий организатора. В случае успешной регистрации возвращает код ответа 201 «Created», в случае ошибки возвращает код ответа 400 «Bad Request».
- **Login user** – POST метод, осуществляющий авторизацию организатора в сервисе. Входными данными является JSON-объект, содержащий аутентификационные данные пользователя: «email» и «password». В случае успешной авторизации, возвращает код ответа 200 «OK» и JSON-объект, содержащий JWT токен авторизации. В случае ошибки во входных данных, возвращает код ответа 401 «Unauthorized».
- **Get user** – GET метод для получения информации пользователя, метод принимает на вход один параметр заголовка – uuid организатора, В случае успешного поиска организатора, возвращает код ответа 200 «OK» и JSON-объект, содержащий не конфиденциальную информацию пользователя. В случае ошибки поиска, возвращает код ответа 400 «Not Found».

В рамках реализации сервиса авторизации были выделены следующие ключевые функции:

- **NewAccessToken –** создаёт новый JWT токен для авторизации пользователя, принимает на вход uuid пользователя для последующей авторизации и время истечения токена, после окончания которого пользователю необходимо будет снова авторизироваться.
- **ParseAccessToken –** функция проверяющая валидность введённого токена, на вход принимает JWT токен, в случае успешной авторизации возвращает декодированную информацию о пользовательском uuid.
- **NewUser –** функция создает нового пользователя, проверяя, что UUID, email и пароль не пусты. Если проверка пройдена, она генерирует хэш пароля с помощью функции createPasshash. В случае ошибки

возвращается объект ошибки, иначе создается и возвращается объект пользователя с временными метками создания и обновления.

3.2.2 Сервис работы с ботами

Сервис работы с ботами представляет собой одновременно и HTTP сервер для получения запросов от организаторов и клиент, работающий с Telegram Bot API по протоколу HTTP.

HTTP сервер реализует паттерн REST API, для доступа к которому в заголовке отправляемого запроса с тегом «Authorization» необходимо указать, полученный в сервисе авторизации JWT токен, в случае не указанного токена или некорректного токена, будет получен код ответа 401 «Unauthorized». Реализует семь HTTP методов:

- **Create Bot –** POST метод, отправляющий запрос на генерацию нового бота. На вход принимает JSON-объект, содержащий полную, заранее описанную схему бота. В случае успешного создания бота, будет получен код ответа 201 «Created». В случае ошибки при описании схемы, будет получен код возврата 400 «Bad request».
- **Start Bot –** POST метод, запускающий ранее созданного бота, на вход принимает один параметр заголовка – uuid бота. В случае существования бота, по указанному uuid будет получен код ответа 200 «ОК», в противном случае, будет получен код ответа 404 «Not Found».
- **Stop Bot -** POST метод, выключающий ранее созданного бота, на вход принимает один параметр заголовка – uuid бота. В случае существования бота, по указанному uuid будет получен код ответа 200 «ОК», в противном случае, будет получен код ответа 404 «Not Found».
- **Start Mailings –** POST метод, запускающий ранее созданную рассылку. Аргументами заголовка запроса выступают uuid бота и ключ рассылки. В случае успешного запуска рассылки будет получен код ответа 200 «ОК», в противном случае будет получен код ответа 404 «Not Found».
- **Get Bots –** GET метод, получающий список ботов конкретного организатора. Не принимает входных параметров, возвращает список

из полных схем, всех созданных организатором ботов с кодом ответа 200 «OK».

– **Get Bot –** GET метод, возвращающий одного бота, на вход принимает один параметр заголовка запроса – uuid бота. В случае успешного поиска бота, возвращает код ответа 200 «OK» и JSON-объект, содержащий полную схему бота, в противном случае, возвращает код ответа 404 «Not Found».
– **Get Answers –** GET метод, возвращающий список ответов пользователей. На вход принимает единственный параметр заголовка запроса - uuid бота. В случае успешного поиска бота, возвращает код ответа 200 «OK» и ответы пользователей в формате «.csv», в противном случае, возвращает код ответа 404 «Not Found».

На рисунке 4 представлена UML (Unified Modeling Language) диаграмма сервиса работы с ботами. На ней представлены основные типы данных и их связи между собой.

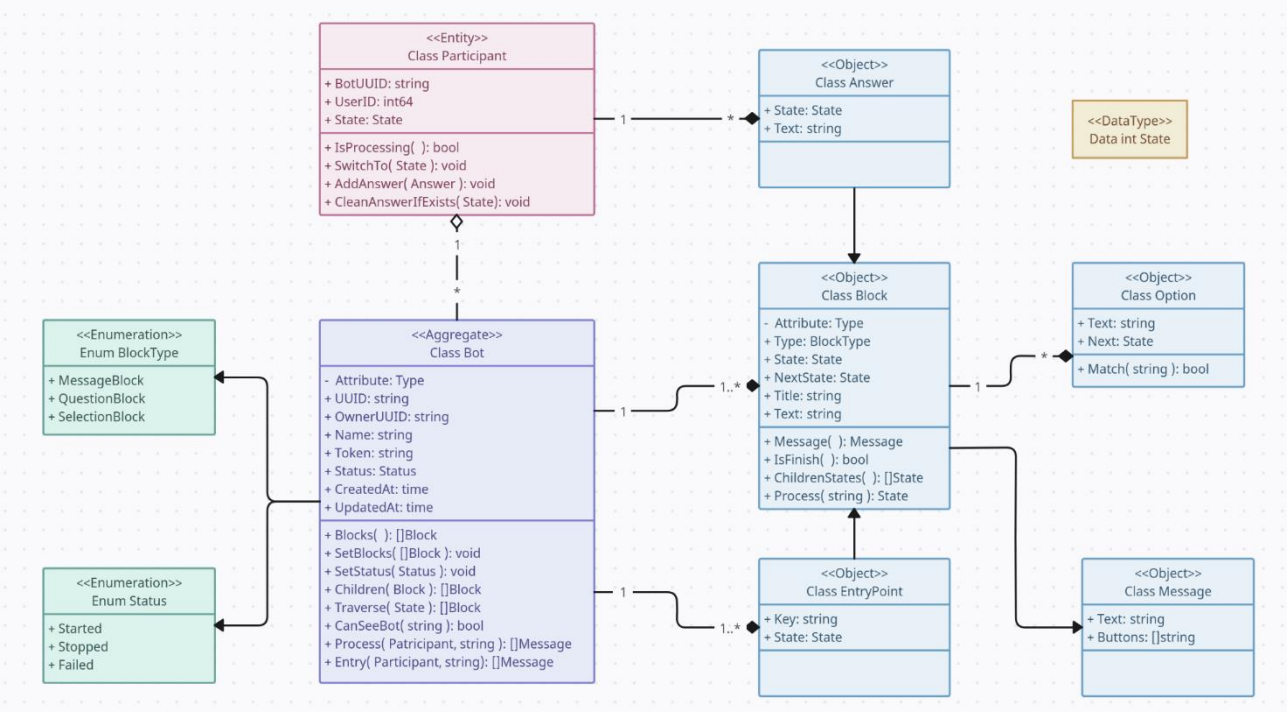

Рисунок 4. UML диаграмма сервиса работы с ботами

В рамках реализации сервиса авторизации были выделены следующие ключевые функции:

- **newTelegramBot –** функция, отвечающая за соединение описанной ранее схемы бота и TelegramAPI, путём предоставления нового экземпляра клиента API.
- **Start –** инициализирует канал обновлений Telegram с помощью GetUpdatesChan, чтобы получать новые сообщения. Затем она обновляет статус бота до "started" с помощью команды UpdateStatus. В фоне запускается функция run, которая обрабатывает входящие обновления.
- **SendMessage -** создает сообщение с переданным текстом и, при необходимости, добавляет кнопки. Если кнопки указаны, она создает клавиатуру с ними, иначе удаляет клавиатуру. Затем отправляет сообщение пользователю и обрабатывает возможные ошибки.
- **NewBot – создает** и инициализирует новый объект бота. Она принимает данные, такие как uuid, ownerUUID, наборы entryPoints, blocks, mailings, name и token. Прежде чем создать объект, функция выполняет валидацию значений и преобразует входные данные в нужный формат. Если один из входных параметров не прошел проверку, функция возвращает ошибку, предотвращая создание невалидного бота.
- **Traverse –** выполняет обход по всем блокам бота, начиная с указанного состояния. Она создает вершины блоков и использует рекурсивную функцию traverseRecursive, чтобы пройти через все возможные состояния и собрать блоки в массив. Результат — это список блоков, которые можно обработать или использовать для визуализации текущей последовательности работы бота.

3.2.3 Интерфейс организатора

В качестве интерфейса организатора реализован telegram-бот, обращающийся к API сервисов авторизации и работы с ботами посредством протокола HTTP.

Можно выделить ключевые статические функции:

- **HandleCallBack** – функция, отвечающая за обработку кнопочных команд от пользователя-организатора. Функция выполняется асинхронно, принимает в качестве аргумента объект типа “CallbackQuery”, представляющий собой событие нажатие кнопки в Telegram;
- **HandleMessage** – функция, отвечающая за обработку текстовых команд от пользователя-организатора. Функция выполняется асинхронно и принимает в качестве аргумента объект типа “Message”, представляющий собой сообщение в Telegram.

Для взаимодействия с API сервисов авторизации и сервиса работы с ботами используется статический класс RequestSender, который реализует восемь асинхронных функций-запросов:

- **SendRegisterRequest** — Отправляет HTTP-запрос для регистрации нового пользователя через REST API;
- **SendLoginRequest** — Отправляет HTTP-запрос для авторизации пользователя по его учетным данным;
- **SendCreateBotRequest** — Отправляет запрос на создание нового Telegram-бота, используя переданный JSON;
- **GetUserBots** — Получает список всех ботов, связанных с текущим пользователем, через GET-запрос;
- **GetBot** — Получает информацию о конкретном боте по его UUID;
- **TurnON** — Отправляет запрос для запуска Telegram-бота по его UUID;

- **TurnOFF** — Отправляет запрос для остановки Telegram-бота по его UUID;
- **GET_CSV** — Загружает ответы пользователей в формате CSV для определенного бота.

3.3 Руководство организатора

Для управления сервисом от лица организатора разработан telegram-bot, содержащий пять основных команды:

- auth – авторизация в сервисе;
- registration – регистрация в сервисе.
- newbot – создание telegram-бота по JSON-схеме;
- mybots – получение списка ранее созданных ботов;
- start – инициализация работы интерфейса организатора.

Команды авторизации и регистрации последовательно запрашивают у организатора электронную почту и пароль и отправляют сообщение об успешной или ошибочной регистрации или авторизации соответственно.

Команда для создания нового telegram-бота запрашивает у пользователя JSON файл, содержащий описание схемы генерируемого бота. Возвращает сообщение об успешном создании бота или ошибке.

При переходе к списку ботов, организатор может выбрать одного из них и получить следующие возможности взаимодействия с ним:

- включить\выключить бота в зависимости от его текущего состояния;
- загрузить ответы участников в формате «.csv»;
- получить короткую информацию о боте: название, дату создание и текущее состояние.

3.4 Руководство администратора

Для развёртывания проекта необходимо по отдельности запустить сервисы работы с ботами, сервис авторизации и интерфейс организатора.

Для корректной работы сервисов необходимо в UNIX систему (WSL в случае Windows NT) установить и развернуть «Docker» — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации. Для этого в терминале необходимо выполнить следующие команды:

- 'for pkg in docker.io docker-doc docker-compose docker-compose-v2 podman-docker containerd runc; do sudo apt-get remove $pkg; done' ;
- 'sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin'.

Так же для запуска программных решений необходимы компиляторы языков «go» и «C#» для этого необходимо выполнить следующие команды:

- 'sudo apt install golang-go'
- 'sudo apt-get install -y dotnet-sdk-8.0'

Для запуска сервиса работы с ботами необходимо в корневой директории сервиса создать файл «.env», содержащий переменные среды, необходимые для корректной работы приложения. Список необходимых переменных среды содержится в файле «.env.example». Следующим шагом необходимо инициализировать и запусть контейнер Docker:

- 'docker compose -f deployment/docker-compose.prod.yaml up -d'

Затем рекомендуется проверить работоспособность проекта путём запуска автоматических тестов:

- 'go test -short ./internal/domain/bots/...', '…' необходимо заменить на название запускаемого теста.

Для запуска сервиса авторизации в корневой директории проекта также необходимо создать файл «.env », содержащий переменные среды, необходимые для корректной работы приложения. Список необходимых

переменных среды содержится в файле «.env.example». Следующим шагом необходимо инициализировать и запусть контейнер Docker:

– 'docker compose -f deployment/docker-compose.prod.yaml up -d'

Для запуска интерфейса организатора в корневой директории проекта необходимо создать файл «.env », содержащий переменные среды, необходимые для корректной работы приложения. Список необходимых переменных среды содержится в файле «.env.example». Для запуска решения необходимо выполнить команду:

– 'dotnet run'

4 Тестирование

Целью проведения тестирования является проверка работоспособности сервиса автоматизации регистрации участников на мероприятия.

4.1 Разработка тестов

Для комплексного тестирования необходимо разработать ряд наборов данных, которые проверят работоспособность и выполнение заявленного функционала конечных точек API.

В таблице 21 представлен набор тестов для оценки работоспособности сервиса автоматизации регистрации:

Таблица 21 – Тесты для оценки работоспособности

Номер теста	Действие	Ожидаемый результат
1	Регистрация с корректными данными.	Успешное создание аккаунта.
2	Регистрация при существовании пользователя с такими же аутентификационными данными.	Получение ошибки о некорректности аутентификационных данных.
3	Авторизация в условиях существования пользователя с такими аутентификационными данными.	Успешная аутентификация и получение JWT токена.
4	Авторизация в условиях некорректного пароля или электронной почты.	Получение сообщения о невалидности аутентификации.
5	Генерация telegram-бота с валидной JSON-схемой в условиях наличия корректного JWT токена.	Успешное создание бота.
6	Генерация telegram-бота с недостижимым состоянием.	Получение сообщения о недостижимости

		состояния с определённым номером.
7	Генерация telegram-бота с некорректно введёнными json-атрибутами схемы.	Получение сообщения о невозможности преобразовать JSON схему в объект приложения.

Содержимое JSON схем представлено в Приложении А.

4.2 Результаты тестирования

Результаты тестирования представлены на рисунках 4-6.

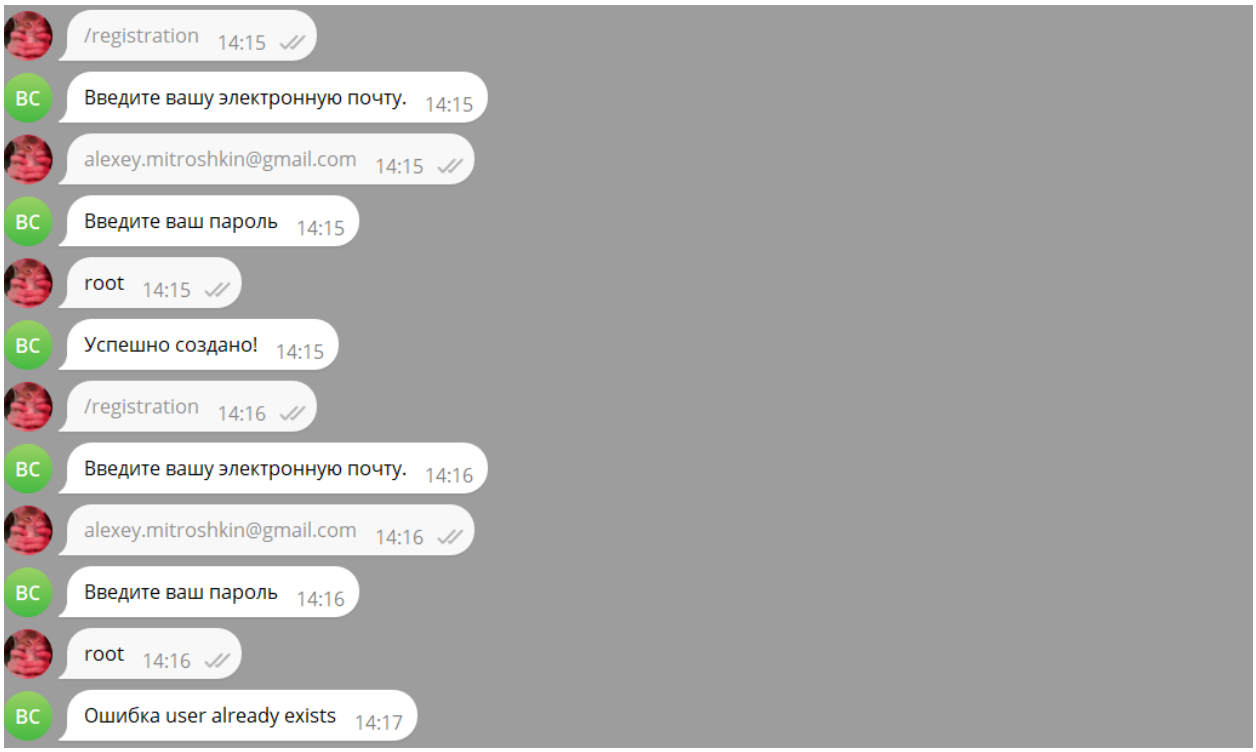

Рисунок 4 - Тест 1-2

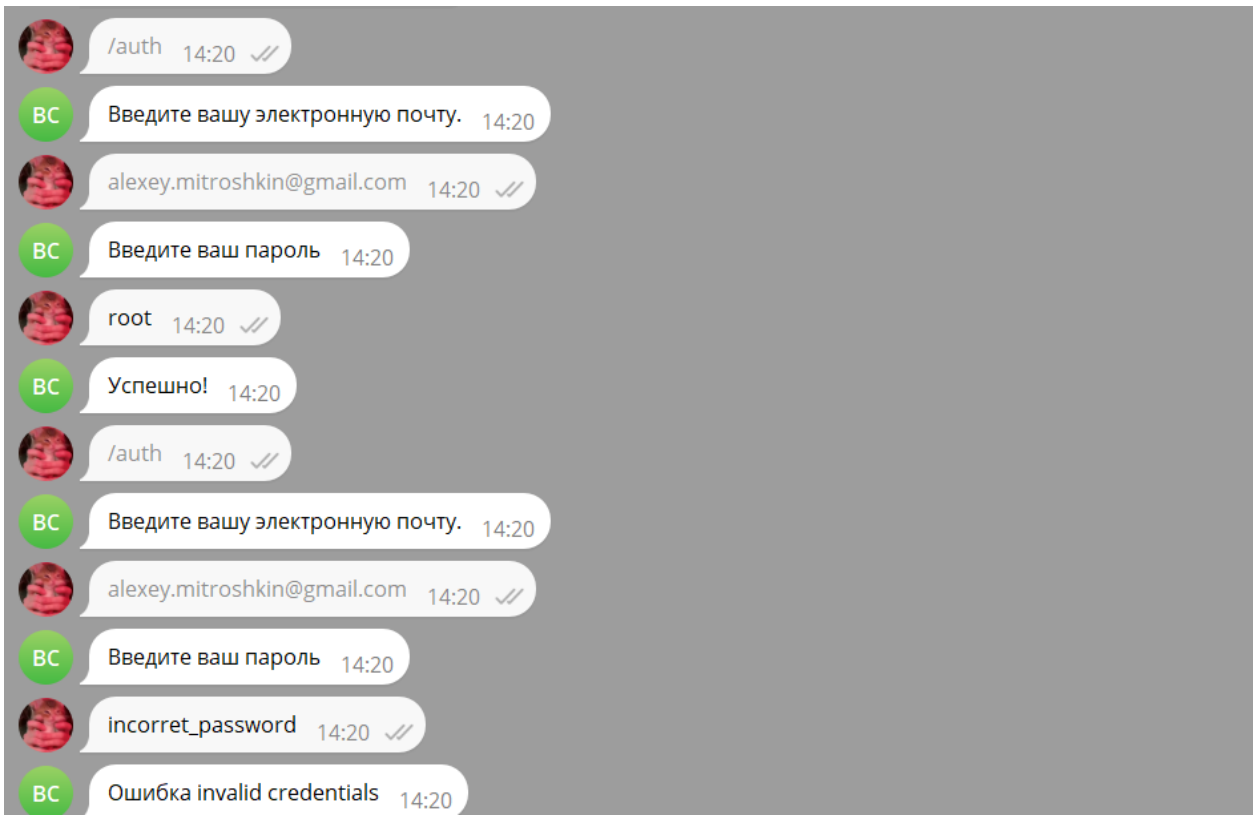

Рисунок 5 - Тесты 3-4

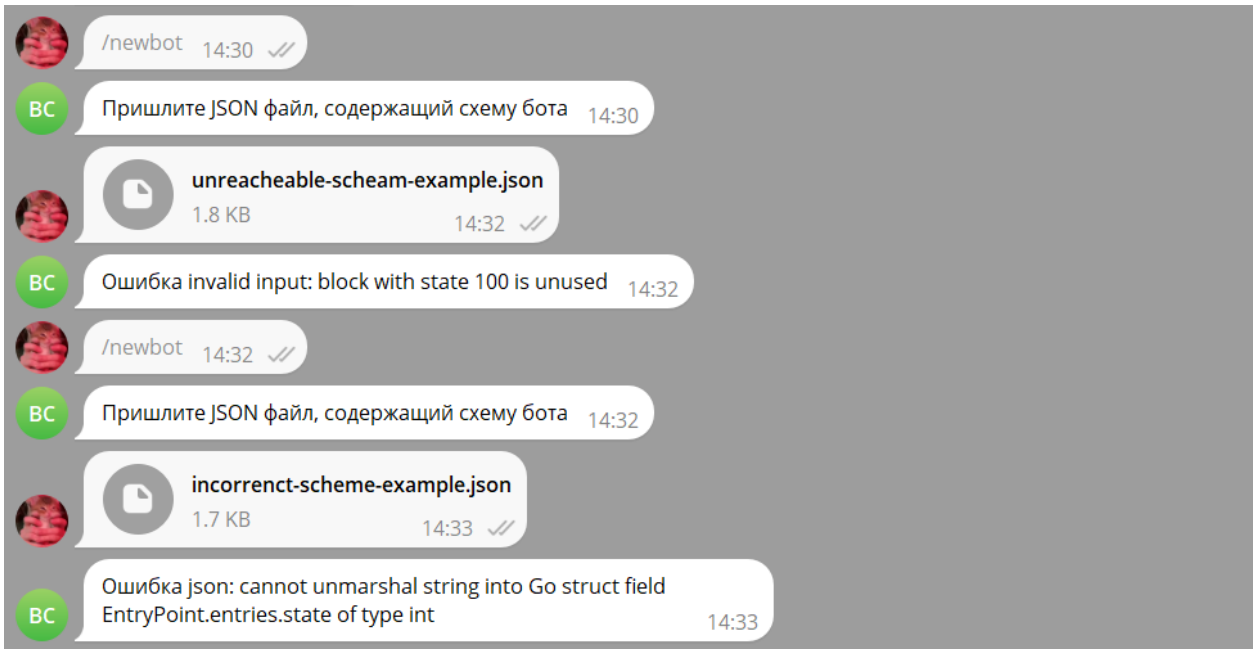

Рисунок 6 - Тесты 5-7

5 Заключение

В ходе выполнения данной курсовой работы был проведен анализ различных подходов к автоматизации регистрации на мероприятия и обоснован выбор использования Telegram-ботов для взаимодействия с участниками. Данный выбор был обусловлен возможностью создания адаптивного интерфейса, интеграцией с мессенджером Telegram и удобством автоматизации процесса уведомлений. Программная реализация была выполнена с учетом требований, изложенных в разделе проектирования.

Результатом курсовой работы стала полноценная система для регистрации участников, включающая интерфейс для организаторов мероприятий, систему управления ботами и базу данных для хранения информации об участниках и их ответах. Тестирование показало, что система успешно обрабатывает различные сценарии регистрации, предоставляет удобный интерфейс для пользователей и корректно управляет процессами обработки данных и отправки уведомлений. Система продемонстрировала стабильную работу при взаимодействии с базой данных и поддержке многопользовательских сценариев, обеспечивая высокую степень автоматизации регистрации.

В качестве дальнейших направлений развития проекта можно выделить добавление возможности отправки автоматических уведомлений о статусе регистрации для участников, расширение системы для поддержки различных сценариев регистрации, а также интеграцию с внешними системами аналитики, что позволит организаторам лучше анализировать активность и вовлеченность участников.

6 Список использованных источников

[1] What is PostgreSQL? - URL: https://www.postgresql.org/about/ (дата обращения 2024-05-25).

[2] Документация .NET 8.0 – URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/

[3] Документация Go – URL: https://go.dev/doc/

[4] Документация Docker – URL: https://docs.docker.com/engine/

[5] Документация NATS – URL: https://docs.nats.io/

7 Приложение А

В таблицах 22-24 представлены JSON схемы, использованные при тестировании.

Таблица 22 – Корретная JSON схема бота

```
{
  "botUUID": "test_bot_correct",
  "name": "my-bot",
  "token": "6277601007:AAECQkdMAKkIrkoDxXhG4Bg2CJymJddeqw8",
  "entries": [
    {
      "key": "start",
      "state": 1
    },
    {
      "key": "ask_5",
      "state": 5
    }
  ],
  "blocks": [
    {
      "type": "selection",
      "state": 1,
      "nextState": 6,
      "title": "is_bmstu",
      "text": "Вы студент МГТУ им. Н.Э. Баумана?",
      "options": [
        {
          "text": "Да",
          "next": 2
        },
        {
          "text": "Нет",
          "next": 3
        }
      ]
    },
    {
      "type": "message",
      "state": 2,
      "nextState": 4,
      "title": "thanks",
      "text": "Отлично, бауманец!"
    },
    {
      "type": "question",
      "state": 3,
      "nextState": 2,
      "title": "Uni",
```

```
      "text": "Укажите ваш ВУЗ пожалуйста."
    }
    ,
    {
      "type": "message",
      "state": 4,
      "nextState": 0,
      "title": "msg",
      "text": "Спасибо за регистрацию!"
    },
    {
      "type": "message",
      "state": 5,
      "nextState": 0,
      "title": "msg",
      "text": "Привет бауманец"
    }
    ,
    {
      "type": "message",
      "state": 6,
      "nextState": 1,
      "title": "err",
      "text": "Ошибка, нажмите кнопку!"
    }
  ],
  "mailings": [
    {
      "name": "bmstu",
      "entryKey": "ask_5",
      "requiredState": 2
    }
  ]
}
```

Таблица 23 – JSON-схема бота, имеющая недостижимое состояние

```
{
    "botUUID": "test_bot_uncorrect",
    "name": "my-bot",
    "token": "6277601007:AAECQkdMAKkIrkoDxXhG4Bg2CJymJddeqw8",
    "entries": [
      {
        "key": "start",
        "state": 1
      },
```

```
    {
      "key": "ask_5",
      "state": 5
    }
  ],
  "blocks": [
    {
      "type": "selection",
      "state": 1,
      "nextState": 6,
      "title": "is_bmstu",
      "text": "Вы студент МГТУ им. Н.Э. Баумана?",
      "options": [
        {
          "text": "Да",
          "next": 2
        },
        {
          "text": "Нет",
          "next": 3
        }
      ]
    },
    {
      "type": "message",
      "state": 2,
      "nextState": 4,
      "title": "thanks",
      "text": "Отлично, бауманец!"
    },
    {
      "type": "question",
      "state": 3,
      "nextState": 2,
      "title": "Uni",
      "text": "Укажите ваш ВУЗ пожалуйста."
    }
    ,
    {
      "type": "message",
      "state": 4,
      "nextState": 0,
      "title": "msg",
      "text": "Спасибо за регистрацию!"
    },
    {
      "type": "message",
      "state": 100,
      "nextState": 0,
      "title": "msg",
```

```
        "text": "Привет бауманец"
      }
      ,
      {
        "type": "message",
        "state": 6,
        "nextState": 1,
        "title": "err",
        "text": "Ошибка, нажмите кнопку!"
      }
      ,
      {
        "type": "message",
        "state": 5,
        "nextState": 1,
        "title": "err",
        "text": "Привет"
      }
    ],
    "mailings": [
      {
        "name": "bmstu",
        "entryKey": "ask_5",
        "requiredState": 2
      }
    ]
  }
```

Таблица 24 – некорректная JSON-схема бота

```
{
    "botUUID": "test_bot_uncorrect1",
    "name": "my-bot",
    "token": "6277601007:AAECQkdMAKkIrkoDxXhG4Bg2CJymJddeqw8",
    "entries": [
      {
        "key": "start",
        "state": "1"
      },
      {
        "key": "ask_5",
        "state": "5"
      }
    ],
    "blocks": [
      {
        "type": "selection",
        "state": "1",
```

```
    "nextState": "6",
    "title": "is_bmstu",
    "text": "Вы студент МГТУ им. Н.Э. Баумана?",
    "options": [
      {
        "text": "Да",
        "next": "2"
      },
      {
        "text": "Нет",
        "next": "3"
      }
    ]
  },
  {
    "type": "message",
    "state": "2",
    "nextState": "4",
    "title": "thanks",
    "text": "Отлично, бауманец!"
  },
  {
    "type": "question",
    "state": "3",
    "nextState": "2",
    "title": "Uni",
    "text": "Укажите ваш ВУЗ пожалуйста."
  }
  ,
  {
    "type": "message",
    "state": "5",
    "nextState": "0",
    "title": "msg",
    "text": "Спасибо за регистрацию!"
  },
  {
    "type": "message",
    "state": "5",
    "nextState": "0",
    "title": "msg",
    "text": "Привет бауманец"
  }
  ,
  {
    "type": "message",
    "state": "6",
    "nextState": "1",
    "title": "err",
    "text": "Ошибка, нажмите кнопку!"
```

```
    }
  ],
  "mailings": [
    {
      "name": "bmstu",
      "entryKey": "ask_5",
      "requiredState": "2"
    }
  ]
}
```