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, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка базы данных приложения бронирования студий»

Студент ИУ7-64Б (Группа) _____ (Подпись, дата) Турчанский Н. А. (И. О. Фамилия)

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

2024 г.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие термины с соответствующими определениями.

База данных — это совокупность данных, оргаизованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ [1].

Система управления базами данных — это программное обеспечение для создания и редактирования баз данных, просмотра и поиска информации в них [2].

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

БД — База Данных

СУБД — Система Управления Базами Данных

SQL — Structed Query Language

ВВЕДЕНИЕ

Роль музыки в структуре духовной жизни современного человека довольно велика, а музыкальная культура в целом становится значительным фактором формирования у молодежи общечеловеческих ценностей [3]. Современные технологии дали возможность не только прослушивать композиции через электронные устройства, но и записывать их, а специально оборудованные студии помогают облегчить этот процесс.

Цель курсовой работы — разработка базы данных приложения бронирования студий. Для достижения поставленной цели необходимо выполнить следующие задачи:

1. провести анализ предметной области;
2. выполнить формализацию задачи;
3. сформулировать описание пользователей;
4. спроектировать сущности базы данных;
5. выбрать средства реализации базы данных и приложения;
6. разработать базу данных и приложение;
7. провести исследование зависимости времени от сложности запроса.

1 Аналитический раздел

В данном разделе будет представлен анализ предметной области, проведена формализация задачи, проведена формализация и описание информации для хранения в БД, проведена формализация и описание пользователей и проанализированы модели баз данных.

1.1 Анализ предметной области

Развитие информационных технологий, помимо различных сфер по всей земле, также не обошло и музыкальную сферу. Музыкальная индустрия одной из первых ощутила на себе смену технологических эпох, изменившись за последние 20 лет на всех уровнях [4].

Российский рынок услуг звукозаписи — это динамично развивающаяся отрасль, которая с каждым годом приобретает все большую популярность и значимость. Еще в 2017 году в России было примерно 656 студий звукозаписи, а в 2023 году их насчитывалось около 1160 студий звукозаписи и репетиционных точек [5].

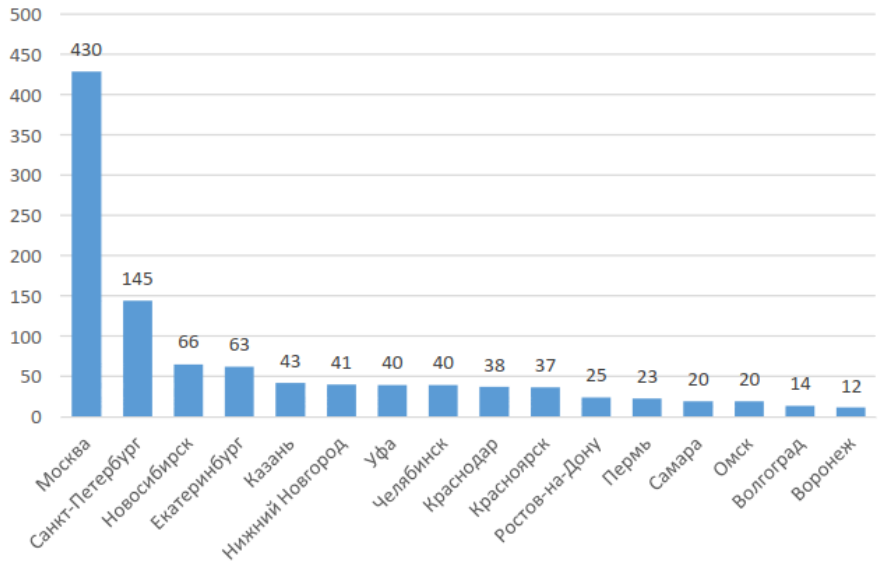

Рисунок 1.1 – Количество студий звукозаписи на момент 20.04.2023 [5]

В связи с этим, онлайн бронирование в студии звукозаписи приобрели весомое значение в нынешнее время.

1.2 Формализация задачи

Для выполнения поставленной цели необходимо разработать БД для хранения информации о студиях, об их составляющем, о пользователях и о бронях, которые пользователи создают.

Также необходимо спроектировать и разработать приложение, которое будет предоставлять интерфейс для работы с БД и давать возможность для каждого авторизованного пользователя создавать бронь на определенное время, резервируя комнату, оборудование, продюсера и инструменталиста.

Нужно предусмотреть возможность добавления, изменение и удаление студий, комнат, оборудования, продюсеров и инструменталистов. Необходимо реализовать разный функционал для разных категорий пользователей.

1.3 Формализация и описание информации, подлежащей хранению в БД

Разрабатываемая БД для приложения бронирования студий должна содержать информацию:

— о зарегистрированных пользователях;

— об имеющихся студиях;

— о комнатах, принадлежащих студиям;

— об оборудовании, принадлежащем студиям;

— о продюсерах, работающих на студиях;

— об инструменталистах, работающих на студиях;

— о бронях на выбранное время на определенную комнату, оборудование, продюсера и инструменталиста.

На рисунке 1.2 представлена ER–диаграмма сущностей в нотации Чена.

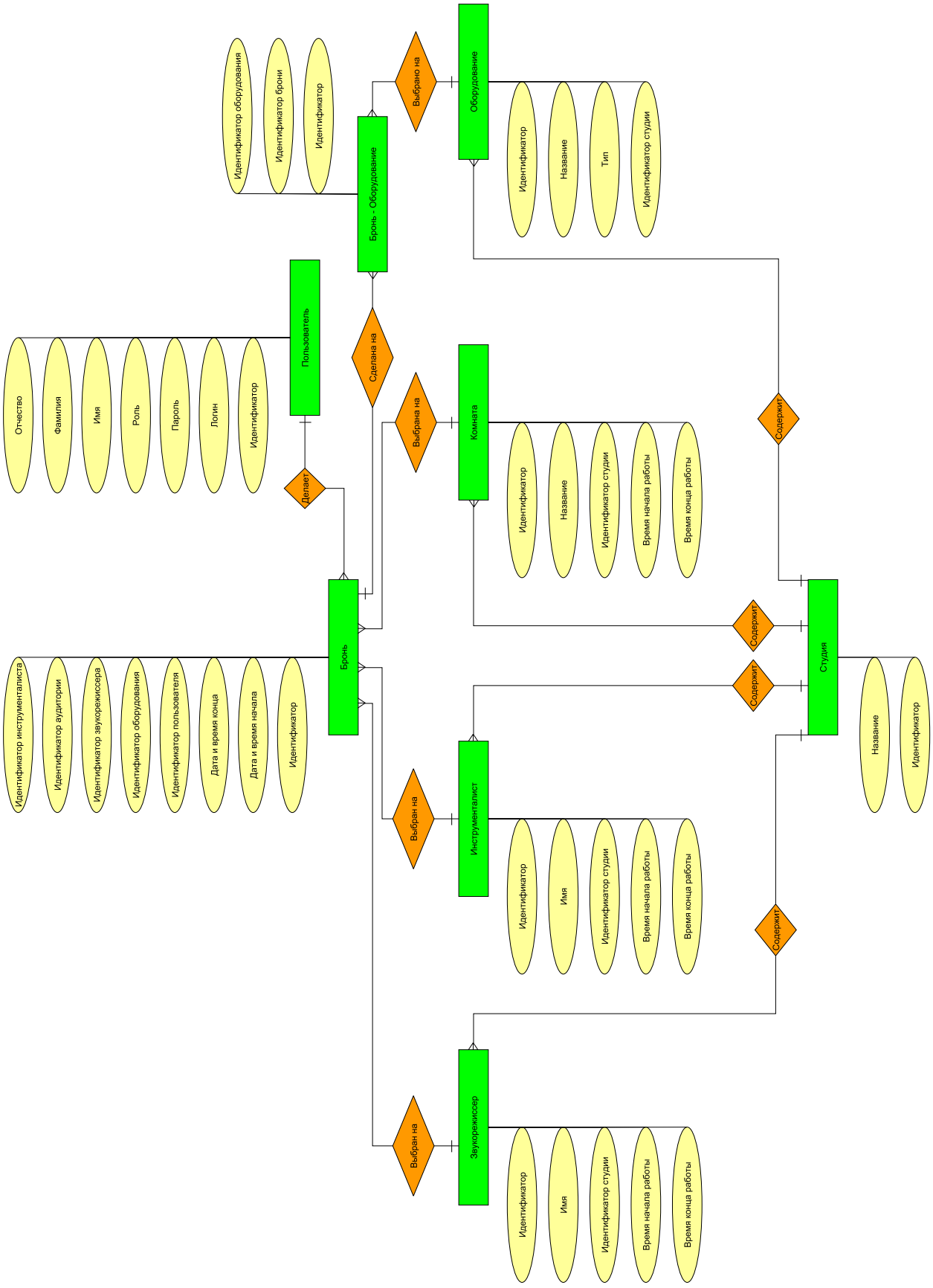

Рисунок 1.2 – ER–диаграмма

1.4 Формализация и описание пользователей проектируемого приложения в БД

Для взаимодействия с приложением было выделено три категории пользователей:

1. гость;
2. авторизованный пользователь;
3. администратор.

Гость имеет право воспользоваться только начальным функционалом приложения: просмотром броней, регистрацией и входом в аккаунт. При успешном прохождении авторизации пользователь автоматически становится авторизованным пользователем.

Функционал авторизованного пользователя является более расширенным и включает в себя: создание, просмотр и отмена уже созданных броней. Также есть возможность изменение личных данных, выхода из профиля и выхода из приложения.

Если пользователь войдет под именем администратора, то он будет иметь возможность:

— добавления, изменения и удаления студий;

— добавления, изменения и удаления комнат;

— добавления, изменения и удаления оборудования;

— добавления, изменения и удаления продюсеров;

— добавления, изменения и удаления инструменталистов.

На рисунке 1.3 представлены пользовательские сценарии.

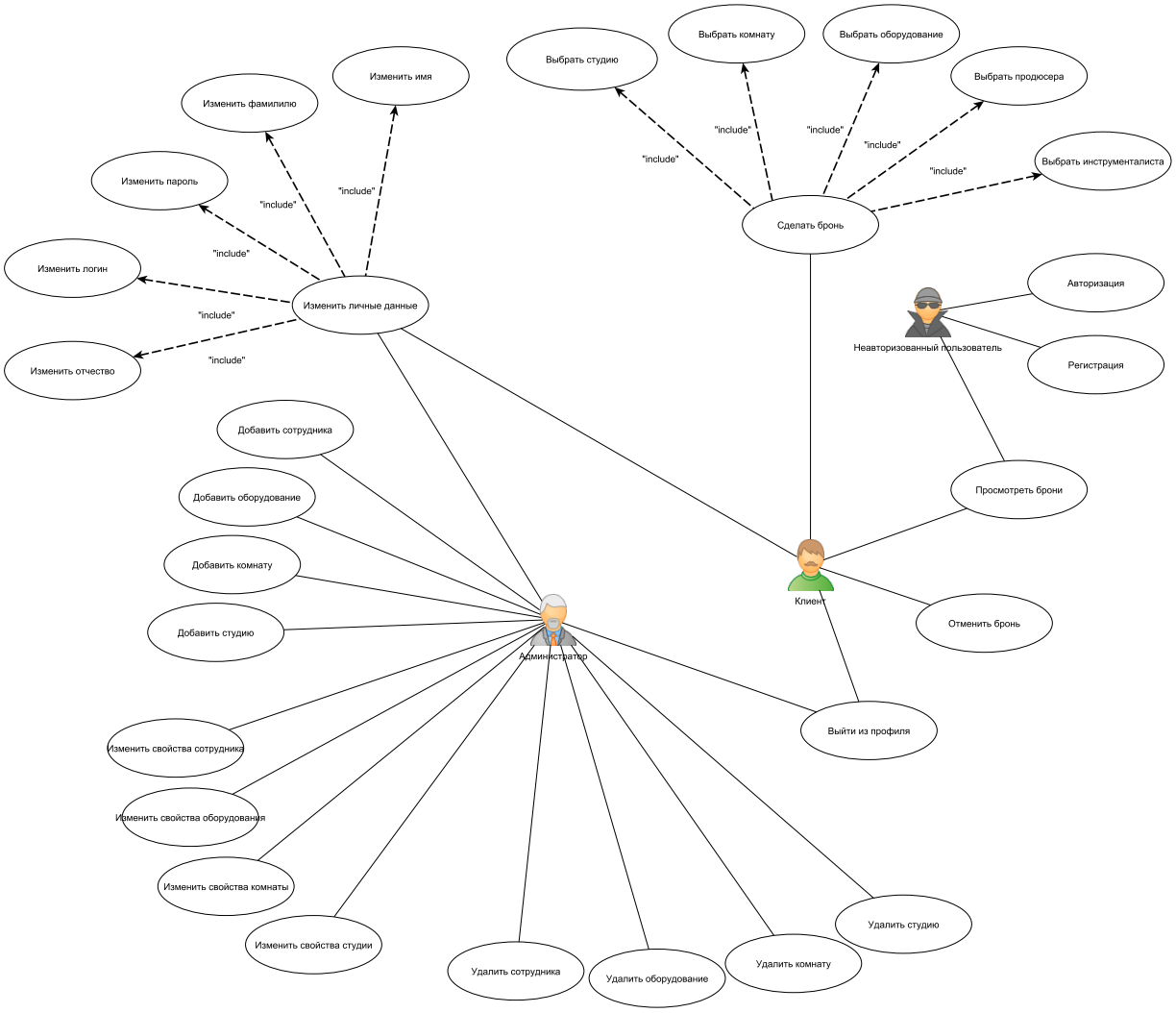

Рисунок 1.3 – Пользовательские сценарии

1.5 Анализ существующих баз данных

Базы данных по способу хранения делят на две группы — колоночные и строковые.

1.5.1 Колоночные базы данных

В колоночных базах данных значения одного атрибута хранятся последовательно друг за другом. Каждая колонка, хранимая на диске, разделена на блоки определенного размера. Блок состоит из заголовка, размер которого пренебрежительно мал по сравнению с размером блока и непосредственно данных. Каждой записи в столбце сопоставляется ее позиция (номер строки) [6].

1.5.2 Строчные базы данных

Под строчным хранением данных обычно понимается физическое хранение кортежа любого отношения в виде одной записи, в котором значение атрибута идут последовательно одно за другим, а за последним атрибутом кортежа в общем случае следует новый кортеж отношения. План запроса представляет собой дерево, у каждого узла которого имеется один родитель и один (или два в случае пересечения) дочерних узла [6].

1.6 Анализ моделей баз данных

Модель данных — совокупность структур данных и операций по их обработке [2].

Существуют модели данных следующих типов:

— дореляционные;

— реляционные;

— постреляционные.

1.6.1 Дореляционные модели

К дореляционным моделям относят модель инвертированных списков, иерархическую модель и сетевую модель:

1. модель инвертированных списков. БД на основе модели инвертированных списков представляет собой совокупность файлов, содержащих записи. Для записей в файле определен некоторый порядок, диктуемый физический организацией данных. Для каждого файла может быть определено произвольное число других упорядоченностей на основании значений некоторых полей записей. Обычно для этого используются индексы. В такой модели данных отсутствуют ограничения целостности как таковые. Все ограничения на возможные экземпляры БД задаются теми программами, которые работают с БД. Одно из немногих ограничений, которое может присутствовать — это ограничение, задаваемое уникальным индексом;

2. иерархическая модель данных строится по принципу иерархии типов объектов, то есть один из типов объекта является главным, а остальные, находящиеся на низших уровнях иерархии, — подчиненными. Между главным и подчиненными объектами устанавливается взаимосвязь «один – ко – многим» [2];

3. В сетевой модели данных любой объект может быть и главным, и подчиненным. Один и тот же объект может одновременно выступать и в роли владельца, и в роли члена набора. Это означает, что любой объект может участвовать в любом числе зависимостей [2].

1.6.2 Реляционные модели

База данных на реляционной модели представляет собой множество отношений. Множество отношений и операций над ними образуют реляционную алгебру. Список операций содержит проекцию, выборку, объединение, пересечение, вычитание, соединение и деление [2].

1.6.3 Постреляционные модели

Классическая реляционная модель предполагает неделимость данных, хранящихся в полях записей таблиц. Это означает, что информация в таблице представляется в первой нормальной форме. Существует ряд случаев, когда это ограничение мешает эффективной реализации приложений. Постреляционная модель данных представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся в записях таблиц. Постреляционная модель данных допускает многозначные поля — поля, значения которых состоят из подзначений. Набор значений многозначных полей считается самостоятельной таблицей, встроенной в основную таблицу [7].

Вывод

В данном разделе была проанализирована предметная область, проведена формализация задачи, проведена формализация и описание информации, проведена формализация и описание пользователей и проанализированы модели баз данных.

2 Конструкторский раздел

В данном разделе будет представлена диаграмма проектируемой БД, описание сущностей, описание проектируемой ролевой модели и описание проектируемых процедур.

2.1 Диаграмма проектируемой базы даннызх

На рисунке 2.1 изображена диаграмма проектируемой базы данных.

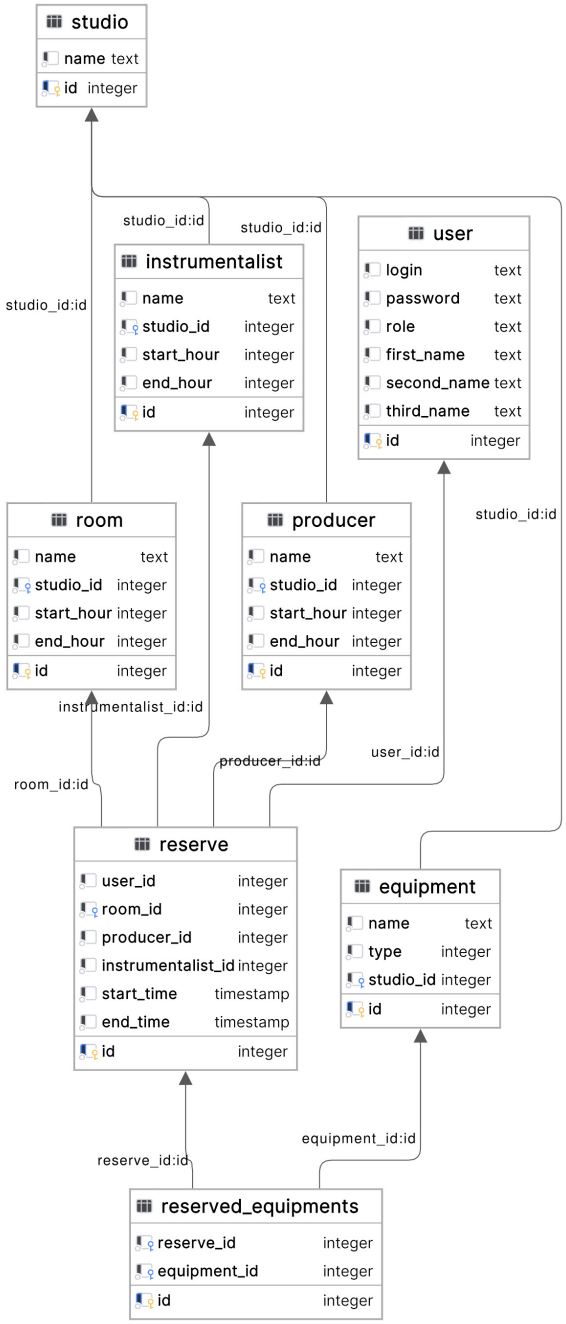

Рисунок 2.1 – Диаграмма базы данных

2.2 Описание сущностей

База данных будет спроектирована из следующих сущностей:

1. таблица *User*, в которой хранятся данные об пользователях;

2. таблица *Reserve*, в которой хранятся данные об бронях;

3. таблица *Studio*, в которой хранятся данные об студиях;

4. таблица *Room*, в которой хранятся данные об комнатах студий;

5. таблица *Equipment*, в которой хранятся данные об оборудовании;

6. таблица *Reserved_ equipment*, в которой хранятся данные об зарезервированном оборудовании;

7. таблица *Producer*, в которой хранятся данные об звукорежиссерах студий;

8. таблица *Instrumentlist*, в которой хранятся данные об инструменталистах студий.

2.2.1 Таблица User

Таблица *User* содержит информацию об идентификаторе пользователя, логине, пароле, роле, имени, фамилии и отчестве. Имеет следующие атрибуты:

— *id* — целое число, первичный ключ, идентификатор пользователя;

— *login* — строка, логин пользователя;

— *password* — строка, пароль пользователя;

— *role* — целое число, роль пользователя;

— *first_ name* — строка, имя пользователя;

— *second_ name* — строка, фамилия пользователя;

— *third_ name* — строка, отчество пользователя.

2.2.2 Таблица Reserve

Таблица *Reserve* содержит информацию об идентификаторе брони, идентификаторе пользователя, идентификаторе комнаты, идентификаторе продюсера, идентификаторе инструменталиста, время начала брони, время конца. Имеет следующие атрибуты:

— *id* — целое число, первичный ключ, идентификатор брони;

— *user_id* — целое число, внешний ключ, идентификатор пользователя;

— *room_id* — целое число, внешний ключ, идентификатор комнаты;

— *producer_id* — целое число, внешний ключ, идентификатор продюссера;

— *instrumentalist_id* — целое число, внешний ключ, идентификатор инструменталиста;

— *start_time* — тип хранения даты и времени, время начала брони;

— *end_time* — тип хранения даты и времени, время конца брони.

2.2.3 Таблица Studio

Таблица *Studio* содержит информацию об идентификаторе студии и названии студии. Имеет следующие атрибуты:

— *id* — целое число, первичный ключ, идентификатор студии;

— *name* — строка, название студии.

2.2.4 Таблица Room

Таблица *Room* содержит информацию об идентификаторе комнаты, названии комнат, идентификаторе студии (к которой принадлежит оборудование), времени начала работы комнаты и времени конца работы комнаты. Имеет следующие атрибуты:

— *id* — целое число, первичный ключ, идентификатор комнаты;

— *name* — строка, название комнаты;

— *studio_id* — целое число, внешний ключ, идентификатор студии;

— *start_time* — тип хранения даты и времени, время начала брони;

— *end_time* — тип хранения даты и времени, время конца брони.

2.2.5 Таблица Equipment

Таблица *Equipment* содержит информацию об идентификаторе оборудования, названии оборудования, типе оборудования, идентификаторе студии (к которой принадлежит оборудование). Имеет следующие атрибуты:

— *id* — целое число, первичный ключ, идентификатор оборудования;

— *name* — строка, название оборудования;

— *type* — целое число, тип оборудования;

— *studio_id* — целое число, внешний ключ, идентификатор студии.

2.2.6 Таблица Reserved_equipment

Таблица *Reserved_equipment* содержит информацию об идентификаторе брони и идентификаторе оборудования (которое принадлежит к брони). Имеет следующие атрибуты:

— *reserve_id* — целое число, внешний ключ, идентификатор брони.

— *equipment_id* — целое число, внешний ключ, идентификатор оборудования.

2.2.7 Таблица Producer

Таблица *Producer* содержит информацию об идентификаторе продюссера, имени продюссера, идентификаторе студии (в которой числится продюссер), времени начала работы продюссера и времени конца работы продюссера. Имеет следующие атрибуты:

— *id* — целое число, первичный ключ, идентификатор продюссера;

— *name* — строка, имя продюссера;

— *studio_id* — целое число, внешний ключ, идентификатор студии;

— *start_time* — тип хранения даты и времени, время начала работы продюссера;

— *end_time* — тип хранения даты и времени, время конца работы продюссера.

2.2.8 Таблица Instrumentalist

Таблица *Instrumentalist* содержит информацию об идентификаторе инструменталиста, имени инструменталиста, идентификаторе студии (в которой числится инструменталист), времени начала работы инструменталиста и времени конца работы инструменталиста. Имеет следующие атрибуты:

— *id* — целое число, первичный ключ, идентификатор инструменталиста;

— *name* — строка, имя инструменталиста;

— *studio_id* — целое число, внешний ключ, идентификатор студии;

— *start_time* — тип хранения даты и времени, время начала работы инструменталиста;

— *end_time* — тип хранения даты и времени, время конца работы инструменталиста.

2.3 Описание проектируемой ролевой модели на уровне базы данных

На уровне взаимодействия с БД представлена следующая ролевая модель:

1. Guest — неавторизованный пользователь системы. Имеет права на:
   — SELECT, INSERT в таблице User;
   — SELECT в таблице Reserve.
2. Client — авторизованный пользователь системы. Имеет права на:
   — SELECT, UPDATE в таблице User;
   — SELECT, INSERT, DELETE в таблице Reserve;
   — SELECT, INSERT, DELETE в таблице Reserved_equipment.
3. Admin — администратор системы. Имеет права на:
   — SELECT, UPDATE в таблице User;

— SELECT, INSERT, UPDATE, DELETE в таблице Studio;

— SELECT, INSERT, UPDATE, DELETE в таблице Room;

— SELECT, INSERT, UPDATE, DELETE в таблице Producer;

— SELECT, INSERT, UPDATE, DELETE в таблице Instrumentalist;

— SELECT, INSERT, UPDATE, DELETE в таблице Equipment.

2.4 Описание проектируемых процедур

На стороне БД при создании были определены две процедуры. Обе из них отвечают корректную работу при создании брони.

2.4.1 Процедура is_reserve

При создании брони запускается процесс транзакции, который включает в себя проверку на занятость выбранных атрибутов (*userId, roomId, producerId, instrumentalistId*) в выбранное время и добавление брони на эти атрибуты. За проверку занятости отвечает процедура *is_reserve*, возвращающая булевое значение. Если ни один атрибут на выбранное время не занят, то возвращается *True*, иначе — *False*.

На рисунке 2.3 представлен алгоритм работы проверки занятости.

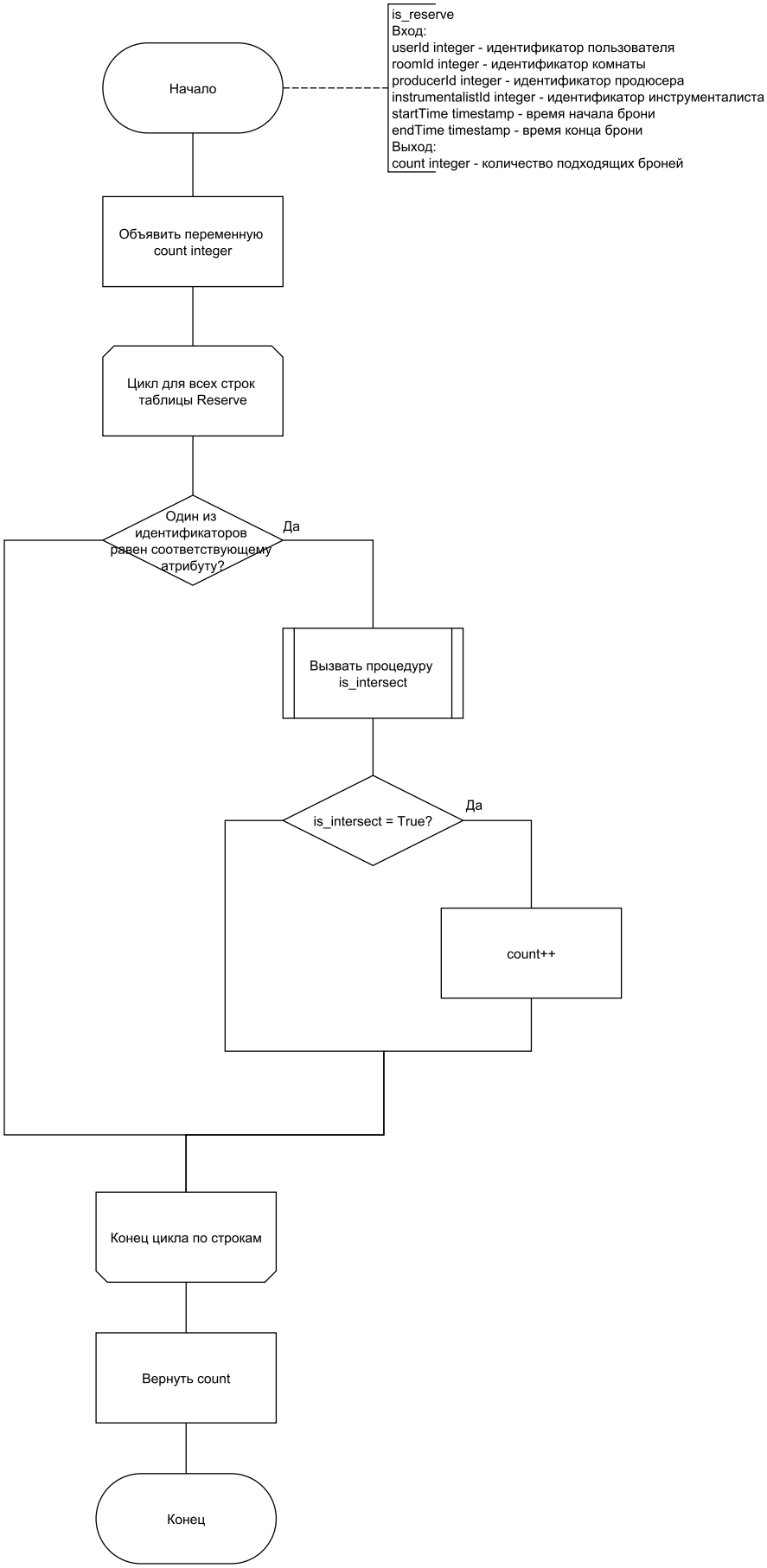

Рисунок 2.2 – Алгоритм работы проверки занятости

2.4.2 Процедура is_intersect

Данная процедура принимает 4 аргумента: выбранное пользователем начало времени, выбранный пользователем конец времени, время начала брони, время конца брони. Если данные временные отрезки пересекаются, то процедура возвращает True, иначе — False.

На рисунке 2.2 представлен алгоритм работы проверки пересечения времени.

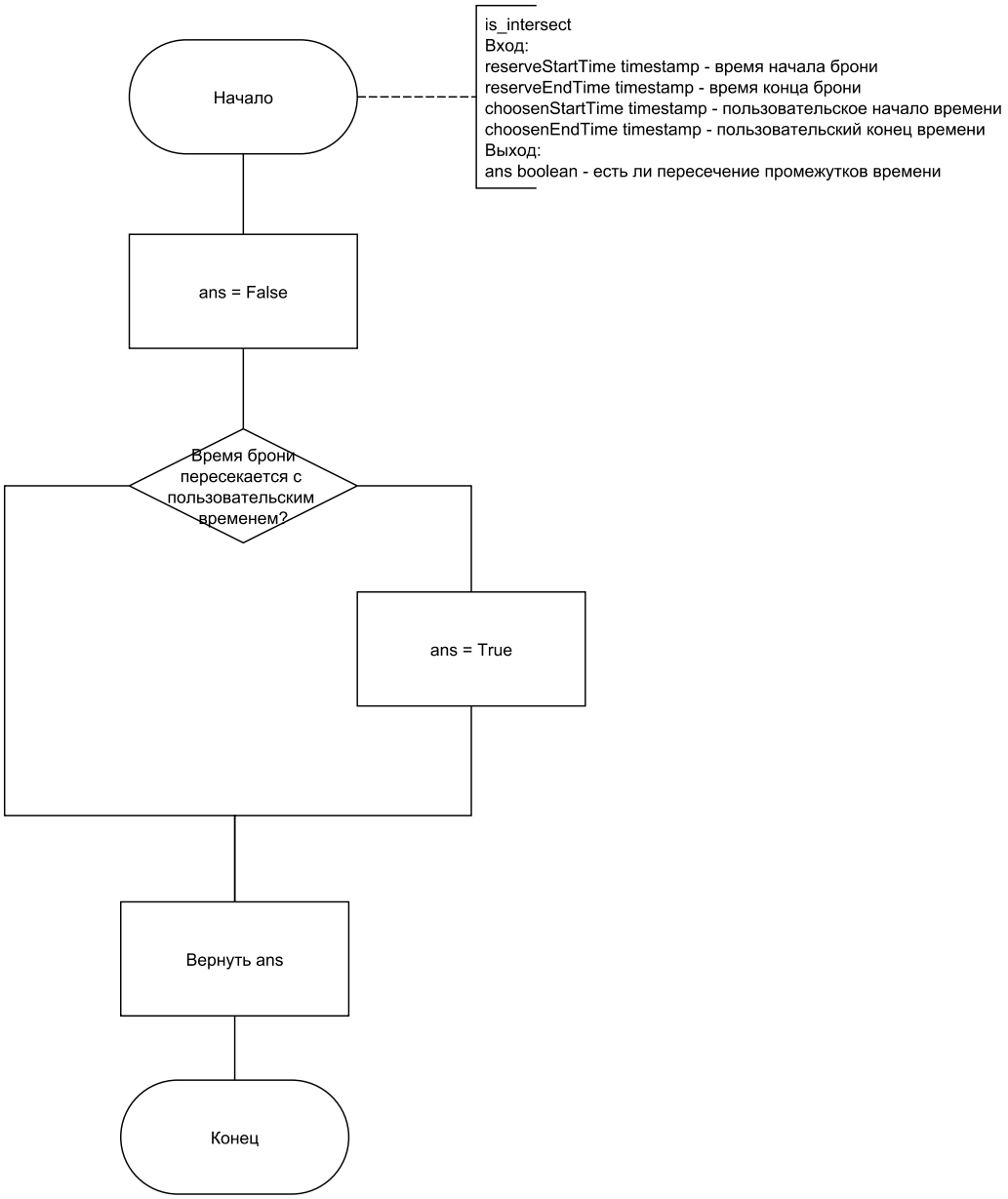

Рисунок 2.3 – Алгоритм работы проверки пересечения времени

Вывод

В данном разделе была представлена диаграмма проектируемой БД, описание сущностей, описание проектируемой ролевой модели и описание проектируемых процедур.

3 Технологический раздел

В данном разделе будет представлены средства реализации, реализация приложения, тестирование и интерфейс приложения.

3.1 Средства реализации

3.1.1 Выбор языка

Для разработки был выбран язык Golang. Данный выбор обусловлен следующими возможностями языка:

— Возможность шифрования данных [8]

— Возможность взаимодействия с выбранными СУБД [9]

— Возможность удобного тестирования ПО [10]

— Возможность работы с JWT-токенами [11]

3.1.2 Выбор СУБД

В аналитическом разделе был сделан выбор в пользу реляционной модели БД. Соответственно, выбор СУБД будет производиться с учетом работы с реляционными моделями. Наиболее популярные и свободно – распространяемые СУБД: MySQL, MariaDB, PostgreSQL, Firebird. Критериями для выбора являются:

— имеющийся опыт работы с выбранной СУБД;

— наличие документации выбранной СУБД;

— поддержка со стороны комьюинити;

Всем этим требованием удовлетворяет СУБД PostregSQL.

3.2 Реализация процедур

Реализация процедур, используемых во время работы алгоритма добавления брони представлены на рисунках 3.1 — 3.2.

```
create or replace function is_reserve(userId integer,
                  roomId integer,
                  producerId integer,
                  instrumentalistId integer,
                  startTime timestamp,
                  endTime timestamp) returns integer language plpgsql as $$
declare
  count integer;
begin
  select count(*) into count
  from reserve
  where (user_id = userId or
    room_id = roomId or
    producer_id = producerId or
    instrumentalist_id = instrumentalistId) and
      is_intersect(reserve.start_time,
            reserve.end_time,
            startTime,
            endTime);

  return count;
end;
$$;
```

Рисунок 3.1 – Реализация алгоритма проверки занятости

```
create or replace function is_intersect(reserveStartTime timestamp,
                    reserveEndTime timestamp,
                    choosenStartTime timestamp,
                    choosenEndTime timestamp) returns boolean language
   plpgsql as $$
declare
  ans boolean;
begin
  ans = false;
  if ((choosenStartTime >= reserveStartTime and choosenStartTime <
   reserveEndTime) or
  (choosenEndTime <= reserveEndTime and choosenEndTime > reserveStartTime)
   or
  (choosenStartTime <= reserveStartTime and choosenEndTime >= reserveEndTime
   )) then
    ans = true;
  end if;
  return ans;
end;
$$;
```

Рисунок 3.2 – Реализация алгоритма проверки пересечения времени

3.3 Реализация приложения

Приложение было реализовано на основе принципа чистой архитектуры. Приложение было разделено на несколько компонентов: Service, Repository и Model.

— Service — отвечает за бизнес – логику приложения;

— Repository — отвечает за работу с хранилищем данных;

— Model — отвечает за хранение сущностей приложения и перенос данных между компонентами приложения.

Именно в компоненте Repository хранятся заготовленные запросы к БД, работающие с помощью библиотеки pgx — драйвера PostgreSQL для языка Golang.

3.4 Тестирование

Для тестирования используемых SQL запросов были использованы две библиотеки. С помощью библиотеки тестирования для Golang [10] были созданы

непосредственно тесты, а с помощью библиотеки testcontainers [12] поднимались контейнеры Docker [13], которые с использованием ранее заготовленных миграций создавали необходимое состояние БД в контейнере под тесты. Данный подход позволяет проводить тестирование независимо от основной рабочей БД.
После успешной подготовки, было выполнено интеграционное тестирование, которое обеспечивает полное покрытие используемых методов, которые, в свою очередь, содержат SQL запросы.
В приложении А приведены код и результат работы тестирования.

3.5 Интерфейс приложения

Интерфейс приложения был создан с помощью сторонней библиотеки tview [14]. Данная библиотека позволяет создавать отдельные страницы для работы с пользователем и дает возможность перемещаться между ними.

3.5.1 Интерфейс гостя

При запуске, приложение считает любого пользователя гостем. Гостю предлагается либо посмотреть уже существующие брони, либо авторизоваться.
Первая страница приложения представлена на рисунке 3.3.

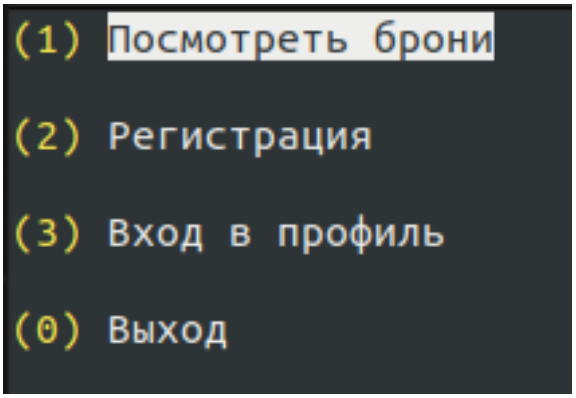

Рисунок 3.3 – Страница с авторизацией

3.5.2 Интерфейс авторизованного пользователя

После успешной регистрации или авторизации пользователь переходит на основную страницу, на которой можно создать бронь, удалить или посмотреть уже созданные. Также можно выйти из профиля или приложения.
Основная страница приложения представлена на рисунке 3.4.

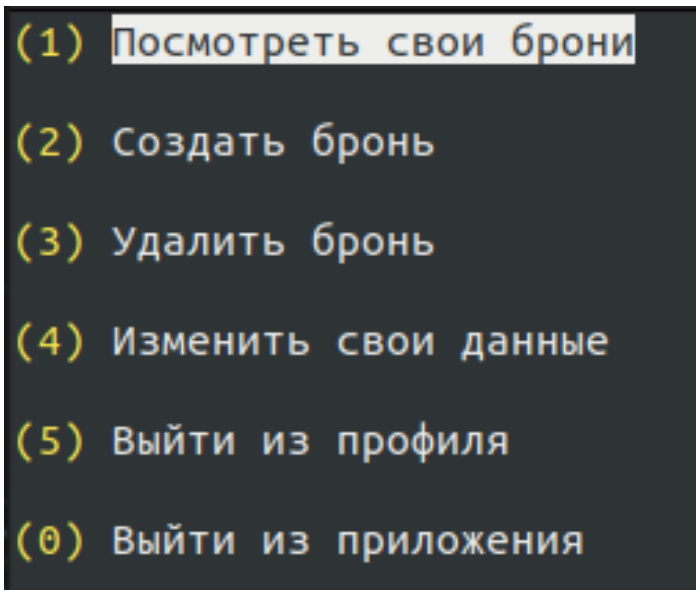

Рисунок 3.4 – Основная страница

При нажатии на кнопку «Создать бронь» откроется меню, в котором необходимо будет выбрать желаемую студию и внести период брони. После нажатия на кнопку «Продолжить» система автоматически подберет все возможные варианты атрибутов для брони и предоставит выбор на следующей странице. После того, как пользователь сделает выбор и нажмет кнопку «Создать бронь», то в соответствующую табличку в БД добавится новый кортеж.
Первая и вторая страница создания брони представлены на рисунке 3.5 и 3.6.

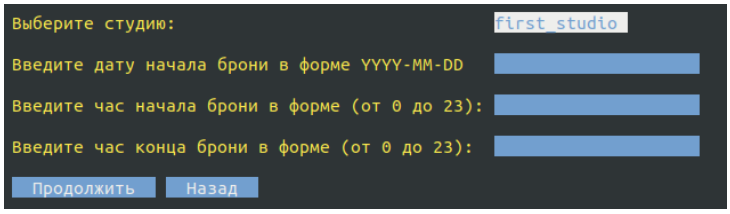

Рисунок 3.5 – Первая страница для создания брони брони

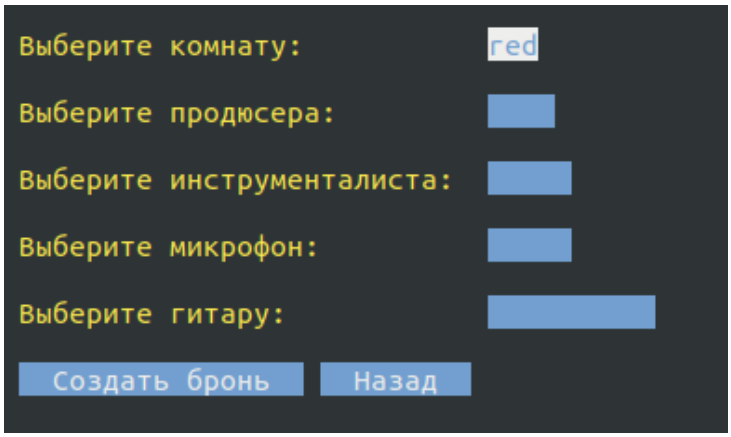

Рисунок 3.6 – Вторая страница для создания брони брони

На рисунке 3.7 представлена страница для снятия брони. Пользователю необходимо выбрать из выпадающего меню нужную дату и нажать кнопку «Снять бронь».

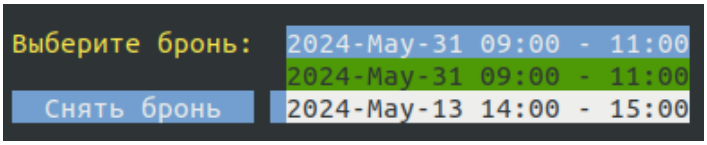

Рисунок 3.7 – Страница для снятия брони

Страница, содержащая созданные пользователем брони и информацию о них, представлена на рисунке 3.8.

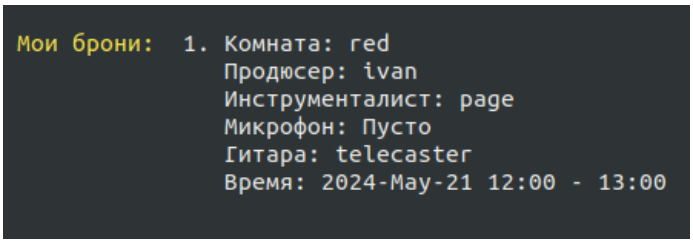

Рисунок 3.8 – Страница броней пользователя

3.5.3 Интерфейс администратора

Основное меню администратора продемонстрировано на рисунке 3.9. Из него можно перейти в другие меню для добавления, удаления или изменения атрибутов.

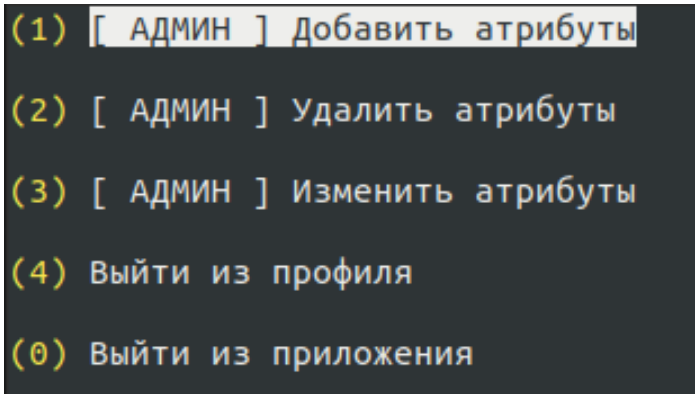

Рисунок 3.9 – Основная страница администратора

Каждая из первых трех функций на странице администратора содержит еще дополнительные меню. В каждом из меню необходимо выбрать атрибут, над которым будет проведено выбранное действие.
Пример данного меню представлен на рисунке 3.10.

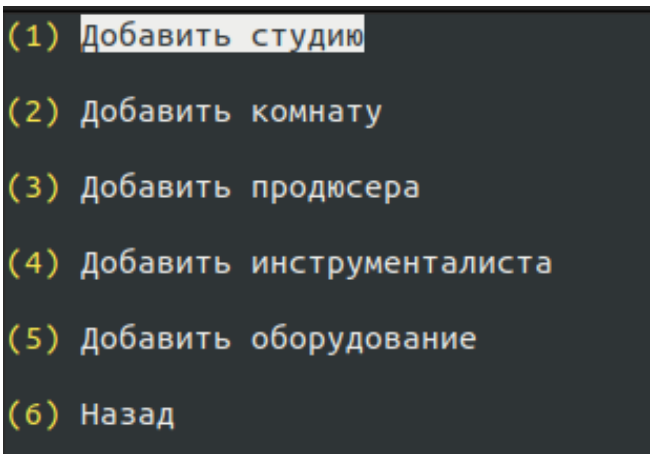

Рисунок 3.10 – Меню для добавления

Вывод

В данном разделе были представлены средства реализации, реализация приложения, тестирование и интерфейс приложения.

4 Исследовательский раздел

В данном разделе будут представлены технические характеристики и будет проведено исследование.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

— операционная система: Ubuntu 20.04 [15];

— размер оперативной памяти: 16 Гбайт;

— процессор AMD Ryzen 5 5500U with Radeon Graphics.

На протяжении всего тестирования компьютер был подключен к сети питания.

4.2 Исследование

Задача заключалась в исследования зависимости времени работы от сложности запроса. Исследование проводилось на заранее заготовленной таблице, количество записей в которой было равно 500 штук.
Градация сложности запросов была следующая:

1. запрос первой сложности:

```
    select equipment.id from equipment
```

2. запрос второй сложности:

```
    select equipment.id,
      equipment.name,
      equipment.type,
      equipment.studio_id
    from equipment
```

3. запрос третьей сложности:

```
    select equipment.id,
      equipment.name,
      equipment.type,
      equipment.studio_id
    from equipment where
      equipment.studio_id = $1
```

4. запрос четвертой сложности:

```
    select equipment.id,
      equipment.name,
      equipment.type,
      equipment.studio_id
    from equipment where
      equipment.studio_id = $1 and
      equipment.type = $2
```

5. запрос пятой сложности

```
      select equipment.id,
        equipment.name,
        equipment.type,
        equipment.studio_id
      from equipment where
        equipment.studio_id = $1 and
        equipment.type = $2 and
        not exists
          (select * from reserved_equipments where equipment.id =
    reserved_equipments.equipment_id)
```

6. запрос шестой сложности

```
      select equipment.id,
        equipment.name,
        equipment.type,
        equipment.studio_id,
        to_char(reserve.start_time, 'YYYY-MM-DD HH24:MI:SS'),
        to_char(reserve.end_time, 'YYYY-MM-DD HH24:MI:SS')
      from equipment, reserve where
        equipment.studio_id = $1 and
        equipment.type = $2 and
        exists
          (select * from reserved_equipments where equipment.id =
    reserved_equipments.equipment_id)
```

Для каждого запроса время замерялось 1000 раз и суммировалось. После чего бралось среднее значение.

По итогам исследования получились следующие результаты, представленные в таблице 4.1 и на рисунке 4.1.

Уровень сложности запроса	Время работы, мс
1	1.181
2	1.383
3	1.397
4	1.402
5	1.445
6	1.532

Таблица 4.1 – Результаты исследования

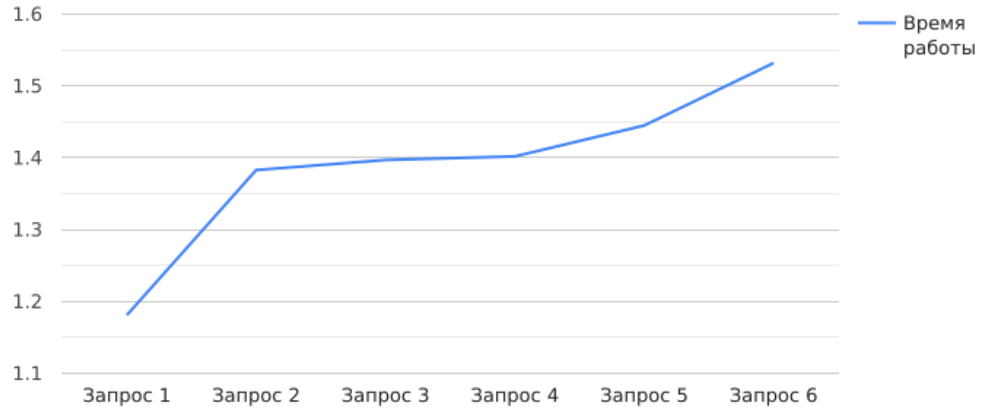

Рисунок 4.1 – График зависимости времени работы от сложности запроса

Вывод

В ходе выполнения исследовательской части было выявлено, что время работы напрямую зависит от сложности запроса — чем сложнее запрос, тем больше времени системе требуется на его обработку. Также на графике можно наблюдать довольно резкий скачок времени работы в 1.16 раз между Запросом 1 и Запросом 2. Это можно объяснить тем, что в инструкции SELECT Запроса 1 необходимо вернуть одно значение, а в инструкции SELECT Запроса 2 4 значения. При проходе по всей таблице, системе необходимо вернуть ответ, который будет включать в себя больше значений, чем при Запросе 1. Также большое время при выполнении Запроса 6 можно объяснить работой с двумя таблицами.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы была разработана база данных и приложение для бронирования студий.
Также были достигнуты следующие задачи:

1. проведен анализ предметной области;
2. выполнена формализацию задачи;
3. сформулированы описание пользователей;
4. спроектированы сущности базы данных;
5. выбраны средства реализации базы данных и приложения;
6. разработана база данных и приложение;
7. проведено исследование зависимости времени от сложности запроса.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Карпова И. П.* Базы данных. Курс лекций и материалов для практиечских заданий. Учебное пособие. — 2013.
2. *Соколов В. А.* Современные системы управления базами данных // Экономика и социум. — 2017. — № 9.
3. *Воробьев Ю. Л.*, *Милорадова И. Н.* Влияние музыки на формирование личности в эпоху интернет // Известия ТулГУ. — 2011. — № 1.
4. *Катаев П. В.* Музыкальные медиа в сетевом обществе: возможности и вызовы функционального многообразия // Вестник Пермского Университета. — 2018. — № 1.
5. *Кобцева Е.*, *Прокопец Т.* РАЗВИТИЕ МУЗЫКАЛЬНОЙ ИНДУСТРИИ В РОССИИ И ЗА РУБЕЖОМ. —.
6. *Григорьев Ю. А.*, *Ермаков Е. Ю.* Сравнение процессов обработки запроса к одной таблице в параллельной строчной и колоночной системе баз данных // Инженерный журнал: наука и инновации. — 2012. — № 3.
7. *Сергеева Т. И.*, *Сергеев М. Ю.* Базы данных: модели данных, проектирование, язык SQL // ФГБОУ ВПО «Воронежский государственный технический университет». — 2012.
8. Документация библиотеки bcrypt Golang.
9. Документация библиотеки pgx Golang.
10. Документация библиотеки тестирования Golang.
11. Документация библиотеки jwt-go Golang.
12. Документация библиотеки testcontainers Golang.
13. Документация контейнеров Docker.
14. Документация библиотеки tview Golang.
15. Документация Ubuntu 20.04.

ПРИЛОЖЕНИЕ А
Тестирование БД

```
func TestStudioPostrgresql_Get(t *testing.T) {
type fields struct {
  db *pgxpool.Pool
}

type args struct {
  ctx     context.Context
  request *dto.GetStudioRequest
}

tests := []struct {
  name string
  args       args
  wantStudio *model.Studio
  wantErr    bool
}{
  {
    name: "test_pos_01",
    args: args{
      ctx: context.Background(),
      request: &dto.GetStudioRequest{
      Id: 1,
      },
    },
    wantStudio: &model.Studio{
      Id:   1,
      Name: "first",
    },
    wantErr: false,
  },
}

for _, tt := range tests {
  t.Run(tt.name, func(t *testing.T) {
    p := postgresql.NewStudioRepository(testDbInstance)

    gotStudio, err := p.Get(tt.args.ctx, tt.args.request)
    if (err != nil) != tt.wantErr {
      t.Errorf("Get() error = %v, wantErr %v", err, tt.wantErr)
    return
    }
    if !reflect.DeepEqual(gotStudio, tt.wantStudio) {
      t.Errorf("Get() gotStudio = %v, want %v", gotStudio, tt.wantStudio)
    }
```

```
  })
}

```

Рисунок А.1 – Код тестирования

```
=== RUN   TestEquipmentRepository_GetFullTimeFreeByStudioAndType
--- PASS: TestEquipmentRepository_GetFullTimeFreeByStudioAndType (0.01s)
=== RUN   TestEquipmentRepository_GetFullTimeFreeByStudioAndType/test_pos_01
  --- PASS: TestEquipmentRepository_GetFullTimeFreeByStudioAndType/
    test_pos_01 (0.01s)
=== RUN   TestEquipmentRepository_GetNotFullTimeFreeByStudioAndType
--- PASS: TestEquipmentRepository_GetNotFullTimeFreeByStudioAndType (0.01s)
=== RUN   TestEquipmentRepository_GetNotFullTimeFreeByStudioAndType/
    test_pos_01
  --- PASS: TestEquipmentRepository_GetNotFullTimeFreeByStudioAndType/
    test_pos_01 (0.01s)
=== RUN   TestReserveRepository_Add
--- PASS: TestReserveRepository_Add (0.00s)
=== RUN   TestReserveRepository_GetByRoomId
--- PASS: TestReserveRepository_GetByRoomId (0.01s)
=== RUN   TestReserveRepository_GetByRoomId/test_pos_01
  --- PASS: TestReserveRepository_GetByRoomId/test_pos_01 (0.01s)
=== RUN   TestReserveRepository_IsRoomReserve
--- PASS: TestReserveRepository_IsRoomReserve (0.00s)
=== RUN   TestReserveRepository_IsRoomReserve/test_pos_01
  --- PASS: TestReserveRepository_IsRoomReserve/test_pos_01 (0.00s)
=== RUN   TestReserveRepository_IsRoomReserve/test_pod_02
  --- PASS: TestReserveRepository_IsRoomReserve/test_pod_02 (0.00s)
=== RUN   TestReserveRepository_IsEquipmentReserve
--- PASS: TestReserveRepository_IsEquipmentReserve (0.00s)
=== RUN   TestReserveRepository_IsEquipmentReserve/test_pos_01
  --- PASS: TestReserveRepository_IsEquipmentReserve/test_pos_01 (0.00s)
=== RUN   TestRoomRepository_GetByStudio
--- PASS: TestRoomRepository_GetByStudio (0.00s)
=== RUN   TestRoomRepository_GetByStudio/test_pos_01
  --- PASS: TestRoomRepository_GetByStudio/test_pos_01 (0.00s)
=== RUN   TestStudioPostrgresql_Get
--- PASS: TestStudioPostrgresql_Get (0.00s)
=== RUN   TestStudioPostrgresql_Get/test_pos_01
  --- PASS: TestStudioPostrgresql_Get/test_pos_01 (0.00s)
=== RUN   TestStudioRepository_Update
--- PASS: TestStudioRepository_Update (0.04s)
=== RUN   TestStudioRepository_Update/test_pos_01
  --- PASS: TestStudioRepository_Update/test_pos_01 (0.04s)
=== RUN   TestStudioRepository_Add
--- PASS: TestStudioRepository_Add (0.11s)
=== RUN   TestStudioRepository_Add/test_pos_01
  --- PASS: TestStudioRepository_Add/test_pos_01 (0.11s)
```

```
=== RUN   TestStudioRepository_Delete
--- PASS: TestStudioRepository_Delete (0.02s)
=== RUN   TestStudioRepository_Delete/test_pos_01
  --- PASS: TestStudioRepository_Delete/test_pos_01 (0.02s)
=== RUN   TestUserRepository_GetByLogin
--- PASS: TestUserRepository_GetByLogin (0.01s)
=== RUN   TestUserRepository_GetByLogin/test_neg_01
  --- PASS: TestUserRepository_GetByLogin/test_neg_01 (0.01s)
PASS
```

Рисунок 4.2 – Результат тестирования