Оглавление

**Введение2**

**1 Аналитическая часть3**

1.1 Постановка задачи3

1.2 Анализ существующих решений3

1.3 Формализация данных4

1.4 Типы пользователей5

1.5 Описание существующих СУБД6

1.5.1 Основные функции СУБД6

1.5.2 Классификации СУБД6

Выводы9

**2 Конструкторская часть10**

2.1 Проектирование БД10

2.2 Требования к программе21

2.3 Структура ПО21

2.4 Способы и этапы тестирования21

2.5 Диаграмма классов21

Выводы23

**3 Технологическая часть24**

3.1 Выбор СУБД24

3.2 Выбор ЯП и среды программирования 24

3.3 Детали реализации24

3.4 Интерфейс программы28

Выводы31

**4 Исследовательская часть24**

4.1 Технические характеристики24

4.2 Постановка эксперимента24

4.3 Результаты эксперимента24

Выводы31

**Заключение34**

**Список литературы35**

**Введение**

Проблема поиска места для репетиций является актуальной для любого музыканта, тем более группы. В крупных городах есть достаточно много репетиционных баз (далее: реп. баз). Все они имеют разные цены и характеристики. Поэтому существует потребность в приложении, которое собирало бы воедино всю имеющуюся информацию о различных репетиционных базах, таким образом освобождая музыкантов от необходимости вручную искать и изучать каждую реп. базу, заходить на их сайты, звонить лично, чтобы забронировать репетицию и т. д.

Цель данной работы – реализовать приложение, содержащее информацию о репетиционных базах. В приложении, работающем с этой БД, должна быть возможность для музыканта бронировать или отменять свои репетиции, а для владельца реп. базы - отслеживать записи на свою реп. базу.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

* формализовать задание, определить необходимый функционал;
* провести анализ СУБД;
* описать структуру БД;
* создать и заполнить БД;
* реализовать интерфейс для доступа к БД;
* разработать ПО, которое позволит пользователю-музыканту бронировать и отменять свои репетиции, а владельцу отслеживать их;
* провести нагрузочное тестирование разработанного приложения.

**1 Аналитическая часть**

В этом разделе будет проанализирована поставленная задача, и рассмотрены различные способы её реализации.

**1.1 Постановка задачи**

Необходимо разработать программу для отображения информации о репетиционных базах с возможностью для музыканта бронировать или отменять свои репетиции, а для владельца реп. базы - отслеживать записи на свою реп. базу.

**1.2 Анализ существующих решений**

На сегодняшний день существует всего два подобных приложения:

* MUSbooking

Это наиболее известное и популярное приложение по бронированию творческих площадок.

Его основные преимущества:

* + возможность бронирования не только репетиционных баз, но и, например, студий звукозаписи, танцевальных залов и т. д.;
  + возможность бронирования творческих площадок в других городах России (не только в Москве).

Основной недостаток: нельзя посмотреть сразу весь список зарегистрированных реп. баз в текущем городе. Список доступных реп. баз можно посмотреть только после указания конкретного времени репетиции, что крайне неудобно в определённых ситуациях.

* TONESKY

Основное преимущество: возможность заранее посмотреть весь список зарегистрированных реп. баз.

Из недостатков:

* + нет поиска по реп. базам;
  + нет цены на превью репетиционной комнаты (т. е., чтобы посмотреть цену, надо зайти «вглубь» и посмотреть подробную информацию);
  + ориентировано только на Москву (не существенно в рамках моей задачи).

**1.3 Формализация данных**

База данных должна хранить информацию о:

* репетиционной базе и её комнатах;
* оборудовании;
* пользователях (музыкантах и владельцах) и об их забронированных репетициях или реп. базах соответственно.

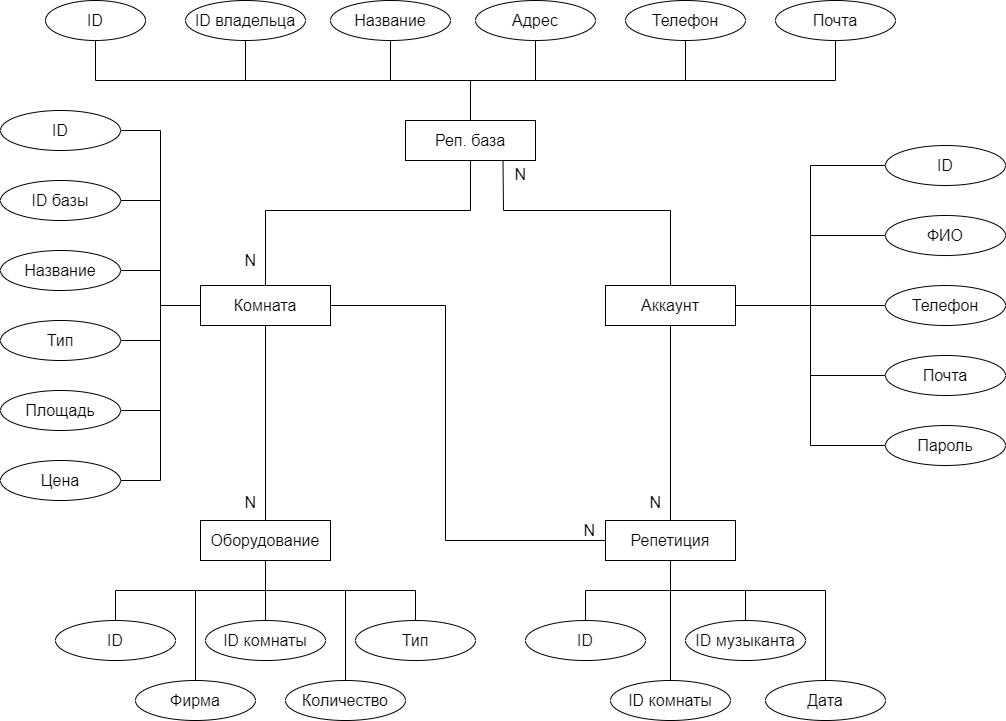


Рис. 1.1: ER диаграмма разрабатываемой базы данных

Таблица 1.1: категории и сведения о данных

|  |  |
| --- | --- |
| Категория | Сведения |
| Репбаза | Название, адрес, телефон, почта, кому принадлежит |
| Комната | Название, тип (вокал/группа и т. д.), площадь, стоимость за 3 часа, к какой репбазе относится |
| Оборудование в комнате | Тип (усилитель/ударные/микрофон и т. д.), бренд, количество, к какой комнате относится |
| Аккаунт | ФИО, телефон, почта |
| Репетиция | Время, какой музыкант (аккаунт), какая комната |

**1.3 Типы пользователей**

Из задачи ясно, что есть два типа пользователей: обычный музыкант и владелец реп. базы. Помимо этого, будет также выделена роль администратора приложения. Для всех троих будет нужна авторизация.

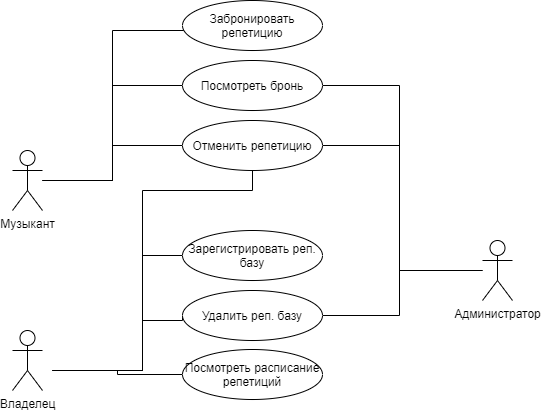


Рис. 1.2: Use-case диаграмма приложения

Таблица 1.2: типы пользователей и их полномочия

|  |  |
| --- | --- |
| Тип пользователя | Полномочия |
| Музыкант | Бронирование репетиций, отмена репетиций, просмотр брони |
| Владелец | Добавление репбазы, удаление своей репбазы, просмотр записей на репетиции, отмена репетиций |
| Администратор | Удаление репбазы, отмена репетиций, просмотр бронирований |

**1.4 Описание существующих СУБД**

Система управления базами данных (СУБД) – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием [баз данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) [1].

**1.4.1 Основные функции СУБД**

Основными функциями СУБД являются:

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

**1.4.2 Классификации СУБД**

По модели данных:

* дореляционные;

**Иерархические.** Это модель данных, где используется представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней.

**Сетевые.** Это логическая модель данных, являющаяся расширением иерархического подхода, строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных.

Разница между иерархической моделью данных и сетевой состоит в том, что в иерархических структурах запись-потомок должна иметь в точности одного предка, а в сетевой структуре данных у потомка может иметься любое число предков.

* реляционные.

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.

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 [2].

По степени распределённости:

* локальные (все части локальной СУБД размещаются на одном компьютере);
* распределённые (части СУБД могут размещаться не только на одном, но на двух и более компьютерах).

По способу доступа к БД:

* файл-серверные;

В файл-серверных СУБД файлы данных располагаются централизованно на файл-сервере. СУБД располагается на каждом клиентском компьютере (рабочей станции). Доступ СУБД к данным осуществляется через локальную сеть. Синхронизация чтений и обновлений осуществляется посредством файловых блокировок.

Преимуществом этой архитектуры является низкая нагрузка на процессор файлового сервера.

Недостатки: потенциально высокая загрузка локальной сети; затруднённость или невозможность централизованного управления; затруднённость или невозможность обеспечения таких важных характеристик, как высокая надёжность, высокая доступность и высокая безопасность. Применяются чаще всего в локальных приложениях, которые используют функции управления БД; в системах с низкой интенсивностью обработки данных и низкими пиковыми нагрузками на БД.

На данный момент файл-серверная технология считается устаревшей, а её использование в крупных информационных системах — недостатком [3].

Пример: Microsoft Access.

* клиент-серверные;

Клиент-серверная СУБД располагается на сервере вместе с БД и осуществляет доступ к БД непосредственно, в монопольном режиме. Все клиентские запросы на обработку данных обрабатываются клиент-серверной СУБД централизованно.

Недостаток клиент-серверных СУБД состоит в повышенных требованиях к серверу.

Достоинства: потенциально более низкая загрузка локальной сети; удобство централизованного управления; удобство обеспечения таких важных характеристик, как высокая надёжность, высокая доступность и высокая безопасность.

Примеры: Oracle Database, MS SQL Server, PostgreSQL, MySQL.

* встраиваемые.

СУБД, которая может поставляться как составная часть некоторого программного продукта, не требуя процедуры самостоятельной установки. Встраиваемая СУБД предназначена для локального хранения данных своего приложения и не рассчитана на коллективное использование в сети.

Физически встраиваемая СУБД чаще всего реализована в виде подключаемой библиотеки. Доступ к данным со стороны приложения может происходить через SQL либо через специальные программные интерфейсы.

Примеры: SQLite, Microsoft SQL Server Compact.

**Выводы**

В этом разделе была проанализирована поставленная задача и уже существующие решения. А также были формализованы данные, типы пользователей и их полномочия. После чего были рассмотрены разные типы СУБД и их функции.

**2 Конструкторская часть**

В этом разделе будет спроектирована база данных и приложение.

**2.1 Проектирование БД**

База данных должна хранить представленные в таблице 1.1 данные. Исходя из этого, в проектируемой базе данных можно выделить следующие таблицы:

* таблица с реп. базами (reh\_base);
* таблица с комнатами реп. баз (room);
* таблица с оборудованием в комнатах (equipment);
* таблица с аккаунтами пользователей (account);
* таблица с забронированными репетициями (rehearsal).

Таблица reh\_base должна содержать информацию о:

* id – идентификатор реп. базы (PK);
* name – название реп. базы (text);
* address – адрес реп. базы (text);
* phone – номер телефона для связи (text);
* mail – электронная почта для связи (text);
* ownerid – id владельца реп. базы (связь с таблицей account) (FK, один-ко-многим).

Таблица room должна содержать информацию о:

* id – идентификатор комнаты (PK);
* name – название комнаты (text);
* type – тип комнаты (text);
* area – площадь комнаты (integer);
* cost – стоимость репетиции в этой комнате за 3 часа (integer);
* baseid – id реп. базы, которой принадлежит комната (связь с таблицей reh\_base) (FK, один-ко-многим).

Таблица equipment должна содержать информацию о:

* id – идентификатор оборудования (PK);
* type – тип оборудования (text);
* brand – фирма оборудования (text);
* amount – количество оборудования такого типа в соответствующей комнате (integer);
* roomid – id комнаты, в которой находится это оборудование (связь с таблицей room) (FK, один-ко-многим).

Таблица account должна содержать информацию о:

* id – идентификатор пользователя (PK);
* fio – ФИО пользователя (text):
* phone – номер телефона пользователя (text);
* mail – электронная почта пользователя (text);
* password – пароль от аккаунта (text);
* type – тип пользователя (text).

Таблица rehearsal должна содержать информацию о:

* id – идентификатор репетиции (PK);
* rehdate – дата и время репетиции (timestamp);
* musicianid – id музыканта, забронировавшего репетицию (связь с таблицей account) (FK, один-ко-многим);
* roomid – id комнаты, в которой забронирована репетиция (связь с таблицей room) (FK, один-ко-многим).

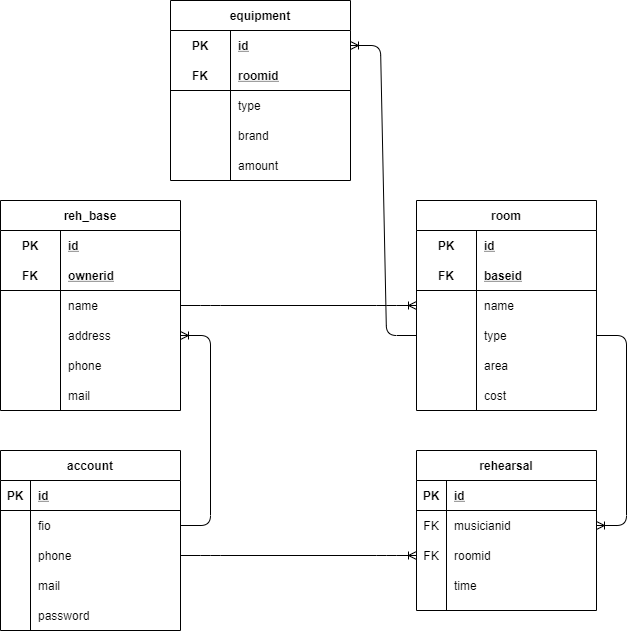


Рис. 2.1: ER диаграмма проектируемой БД

**2.2 Требования к программе**

Разрабатываемое ПО должно предоставлять следующие возможности:

* регистрация нового пользователя;
* авторизация пользователя;
* вывод списка комнат, доступных для бронирования;
* вывод подробной информации о каждой комнате;
* бронь комнаты на соответствующее время;
* отмена брони;
* вывод уже забронированных (будущих) репетиций;
* вывод зарегистрированных реп. баз для данного аккаунта;
* регистрация новой реп. базы;
* добавление комнаты в зарегистрированную реп. базу;
* добавление оборудования в соответствующую комнату;
* вывод всех будущих репетиций на соответствующей реп. базе;
* удаление реп. базы;
* поиск по имени реп. базы;
* поиск по дате и времени репетиции.

Ограничения, в рамках которых будет работать программа:

* аккаунт нельзя удалить;
* нельзя зарегистрироваться стандартным способом в качестве администратора (администраторы добавляются «вручную» на уровне БД);
* для обновления информации нужно закрыть соответствующее окно и снова его открыть;
* для выхода из аккаунта нужно закрыть окно;
* пароль от аккаунта хранится в БД в обычном виде, без шифрования;
* календарь для бронирования репетиций не предоставляется.

**2.3 Структура ПО**

Всё проектируемое ПО можно разделить на две основные части:

* frontend (часть приложения, с которой непосредственно взаимодействует пользователь, отображение данных);
* backend (взаимодействие с базой данных).

Структура frontend части в свою очередь будет основана на паттерне MVC (Model, View, Controller).

Таким образом, проект будет разделён на несколько файлов:

* connect – взаимодействие непосредственно с БД;
* gui – здесь будут находиться модели и контроллеры frontend части;
* отдельные файлы для каждого окна итогового приложения, предоставляющие интерфейс пользователю (т. е. отвечающие за view часть).

Файл connect будет состоять из набора следующих функций:

* connect\_\* – для подключения к БД с соответствующей ролью;
* функции для формирования и посылки запросов к БД.

Файл gui в свою очередь будет состоять из набора классов, каждый из которых будет соответствовать определённому окну приложения.

**2.4 Способы и этапы тестирования**

Для проверки работоспособности ПО будет применяться функциональное тестирование.

Тестирование ПО будет разделено на следующие этапы:

* регистрация нового пользователя (как музыканта, так и владельца);
* попытка зарегистрироваться ещё раз с той же почтой;
* попытка авторизоваться с неправильной почтой или паролем;
* авторизация в качестве музыканта;
  + бронирование репетиции;
  + попытка забронировать репетицию в уже забронированной на это время комнате;
  + просмотр своих будущих репетиций;
  + отмена брони;
  + попытка забронировать репетицию «в прошлом»;
* авторизация в качестве владельца;
  + регистрация новой реп. базы;
  + попытка зарегистрировать уже существующую реп. базу (с той же почтой либо с тем же названием и адресом);
  + добавление комнаты в реп. базу;
  + попытка добавить уже существующую комнату (с тем же названием);
  + добавление оборудования в комнату;
  + попытка добавить оборудование в несуществующую комнату;
  + попытка повторно добавить то же самое оборудование в ту же самую комнату;
  + просмотр предстоящих репетиций на соответствующей реп. базе;
  + удаление реп. базы;
* авторизация в качестве администратора;
  + поиск реп. баз по названию (успешный);
  + попытка найти несуществующую реп. базу;
  + поиск репетиций по дате (успешный);
  + попытка найти несуществующую репетицию.

**2.5 Диаграмма классов**

На рисунке 2.2 представлена диаграмма классов разрабатываемого приложения.

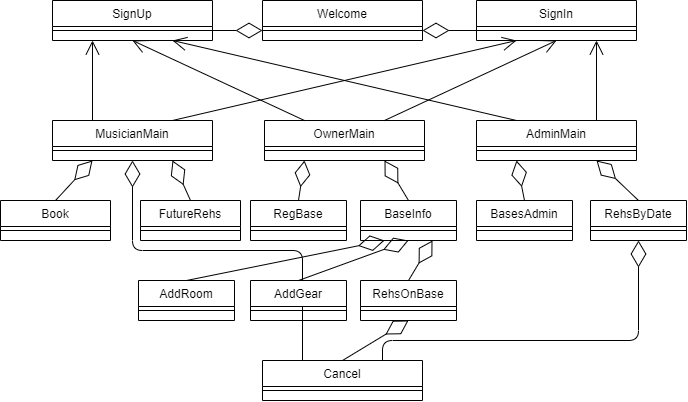


Рис. 2.2: Диаграмма классов

**Выводы**

На основе теоретических данных, полученных в аналитическом разделе, были спроектированы база данных и приложение. А также были приведены: требования к программе, способы и этапы тестирования и диаграмма классов.

**3 Технологическая часть**

В этом разделе будет разработан и протестирован исходный код программного обеспечения.

**3.1 Выбор СУБД**

В таблице 3.1 произведено сравнение наиболее популярных СУБД, которые могут быть использованы для реализации хранения данных в разрабатываемом программном продукте.

Таблица 3.1: Особенности различных СУБД

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Особенность | MySQL | PostgreSQL | MS SQL Server | Oracle Database |
| Open source | GNU GPL с открытым исходным кодом | Открытый исходный код | Коммерческая. Требуется приобретение лицензии | Коммерческая. Требуется приобретение лицензии |
| Соответствие ACID | Частичное | Полное | Полное | Полное |
| NoSQL/JSON | Поддержка некоторых функций | JSON, NoSQL | JSON, NoSQL | JSON, NoSQL |
| Поддержка MERGE | Да | Нет | Да | Да |

Таким образом, для реализации проекта в качестве СУБД будет использоваться PostgreSQL, так как он является свободно распространяемым и удовлетворяет всем необходимым требованиям в рамках поставленной задачи.

**3.2 Выбор ЯП и среды программирования**

Для реализации проекта в качестве языка программирования был выбран язык Python, так как:

* этот язык поддерживает работу с PostgreSQL (в моём случае для этого будет использоваться библиотека psycopg2);
* данный язык является полностью объектно-ориентированным (всё является объектами) [6], что позволяет в полной мере использовать классы при написании программы;
* в этом языке присутствует PyQt (набор расширений графического фреймворка [Qt](https://ru.wikipedia.org/wiki/Qt) для языка программирования Python, выполненный в виде расширения Python [5]), позволяющий легко создавать графический интерфейс для приложения.

В качестве среды разработки был выбран PyCharm по следующим причинам:

* наличие бесплатной версии Community Edition;
* множество удобств, облегчающих процесс написания и отладки кода.

**3.3 Детали реализации**

В листингах 3.1 – 3.2 представлен пример реализации frontend части приложения.

Листинг 3.1: GUI окна входа/регистрации

|  |
| --- |
| **class** **Ui\_MainWindow(object):**  **def** setupUi**(**self**,** MainWindow**):**  MainWindow**.**setObjectName**(**"MainWindow"**)**  MainWindow**.**resize**(**800**,** 600**)**  self**.**centralwidget **=** QtWidgets**.**QWidget**(**MainWindow**)**  self**.**centralwidget**.**setObjectName**(**"centralwidget"**)**  self**.**signin\_button **=** QtWidgets**.**QPushButton**(**self**.**centralwidget**)**  self**.**signin\_button**.**setGeometry**(**QtCore**.**QRect**(**240**,** 290**,** 161**,** 31**))**  font **=** QtGui**.**QFont**()**  font**.**setPointSize**(**12**)**  self**.**signin\_button**.**setFont**(**font**)**  self**.**signin\_button**.**setObjectName**(**"signin\_button"**)**  self**.**signup\_button **=** QtWidgets**.**QPushButton**(**self**.**centralwidget**)**  self**.**signup\_button**.**setGeometry**(**QtCore**.**QRect**(**420**,** 290**,** 161**,** 31**))**  font **=** QtGui**.**QFont**()**  font**.**setPointSize**(**12**)**  self**.**signup\_button**.**setFont**(**font**)**  self**.**signup\_button**.**setObjectName**(**"signup\_button"**)**  self**.**label **=** QtWidgets**.**QLabel**(**self**.**centralwidget**)**  self**.**label**.**setGeometry**(**QtCore**.**QRect**(**300**,** 250**,** 251**,** 20**))**  font **=** QtGui**.**QFont**()**  font**.**setPointSize**(**12**)**  self**.**label**.**setFont**(**font**)**  self**.**label**.**setObjectName**(**"label"**)**  MainWindow**.**setCentralWidget**(**self**.**centralwidget**)**  self**.**menubar **=** QtWidgets**.**QMenuBar**(**MainWindow**)**  self**.**menubar**.**setGeometry**(**QtCore**.**QRect**(**0**,** 0**,** 800**,** 18**))**  self**.**menubar**.**setObjectName**(**"menubar"**)**  MainWindow**.**setMenuBar**(**self**.**menubar**)**  self**.**statusbar **=** QtWidgets**.**QStatusBar**(**MainWindow**)**  self**.**statusbar**.**setObjectName**(**"statusbar"**)**  MainWindow**.**setStatusBar**(**self**.**statusbar**)**  self**.**retranslateUi**(**MainWindow**)**  QtCore**.**QMetaObject**.**connectSlotsByName**(**MainWindow**)**  **def** retranslateUi**(**self**,** MainWindow**):**  \_translate **=** QtCore**.**QCoreApplication**.**translate  MainWindow**.**setWindowTitle**(**\_translate**(**"MainWindow"**,** "MainWindow"**))**  self**.**signin\_button**.**setText**(**\_translate**(**"MainWindow"**,** "Войти"**))**  self**.**signup\_button**.**setText**(**\_translate**(**"MainWindow"**,** "Зарегистрироваться"**))**  self**.**label**.**setText**(**\_translate**(**"MainWindow"**,** "Добро пожаловать в HearBase!"**))** |

Листинг 3.2: Контроллер окна входа/регистрации

|  |
| --- |
| **class** **Welcome(**QtWidgets**.**QMainWindow**,** welcome**.**Ui\_MainWindow**):**  **def** \_\_init\_\_**(**self**):**  **super().**\_\_init\_\_**()**  self**.**setupUi**(**self**)**  self**.**signin\_button**.**clicked**.**connect**(**self**.**sign\_in**)**  self**.**signup\_button**.**clicked**.**connect**(**self**.**sign\_up**)**  **def** sign\_in**(**self**):**  self**.**window **=** SignIn**()**  self**.**window**.**show**()**  **def** sign\_up**(**self**):**  self**.**window **=** SignUp**()**  self**.**window**.**show**()** |

В листинге 3.3 представлена реализация соединения с базой данных в соответствии с ролевой моделью.

Листинг 3.3: Подключение к БД с разными ролями

|  |
| --- |
| **def** connect**():**  connection **=** **None**  **try:**  connection **=** psycopg2**.**connect**(**host**=**"localhost"**,** database**=**"DB\_course"**,**  user**=**"postgres"**,** password**=**"\*\*\*\*"**)**  **except** OperationalError **as** e**:**  **print(**"The error '{e}' occurred"**)**  **return** connection  **def** connect\_musician**():**  connection **=** **None**  **try:**  connection **=** psycopg2**.**connect**(**host**=**"localhost"**,** database**=**"DB\_course"**,**  user**=**"musician"**,** password**=**"\*\*\*\*"**)**  **print(**"Connection of musician successful"**)**  **except** OperationalError **as** e**:**  **print(**"The error '{e}' occurred"**)**  **return** connection  **def** connect\_owner**():**  connection **=** **None**  **try:**  connection **=** psycopg2**.**connect**(**host**=**"localhost"**,** database**=**"DB\_course"**,**  user**=**"base\_owner"**,** password**=**"\*\*\*\*"**)**  **print(**"Connection of owner successful"**)**  **except** OperationalError **as** e**:**  **print(**f"The error '{e}' occurred"**)**  **return** connection  **def** connect\_admin**():**  connection **=** **None**  **try:**  connection **=** psycopg2**.**connect**(**host**=**"localhost"**,** database**=**"DB\_course"**,**  user**=**"app\_admin"**,** password**=**"\*\*\*\*"**)**  **print(**"Connection of admin successful"**)**  **except** OperationalError **as** e**:**  **print(**f"The error '{e}' occurred"**)**  **return** connection |

Помимо этого, на уровне базы данных была реализована процедура, необходимая для удаления реп. базы (листинги 3.4 – 3.5).

Листинг 3.4: Процедура удаления реп. базы

|  |
| --- |
| **CREATE** **OR** **REPLACE** **PROCEDURE** del\_all**(**base\_id int**)**  **AS** $$  **BEGIN**  **delete** **from** rehearsal **where** roomid **in** **(select** id **from** room **where** baseid **=** base\_id**);**  **delete** **from** equipment **where** roomid **in** **(select** id **from** room **where** baseid **=** base\_id**);**  **delete** **from** room **where** baseid **=** base\_id**;**  **delete** **from** reh\_base **where** id **=** base\_id**;**  **END;**  $$ **LANGUAGE** PLPGSQL**;** |

Листинг 3.5: Применение процедуры del\_all при удалении реп. базы

|  |
| --- |
| **def** del\_base**(**conn**,** base\_id**):**  cur **=** conn**.**cursor**()**  query **=** "CALL del\_all(%s)"  cur**.**execute**(**query**,** **(**base\_id**,))**  conn**.**commit**()**  cur**.**close**()** |

**3.4 Интерфейс программы**

На рисунках ниже представлен основной интерфейс приложения.

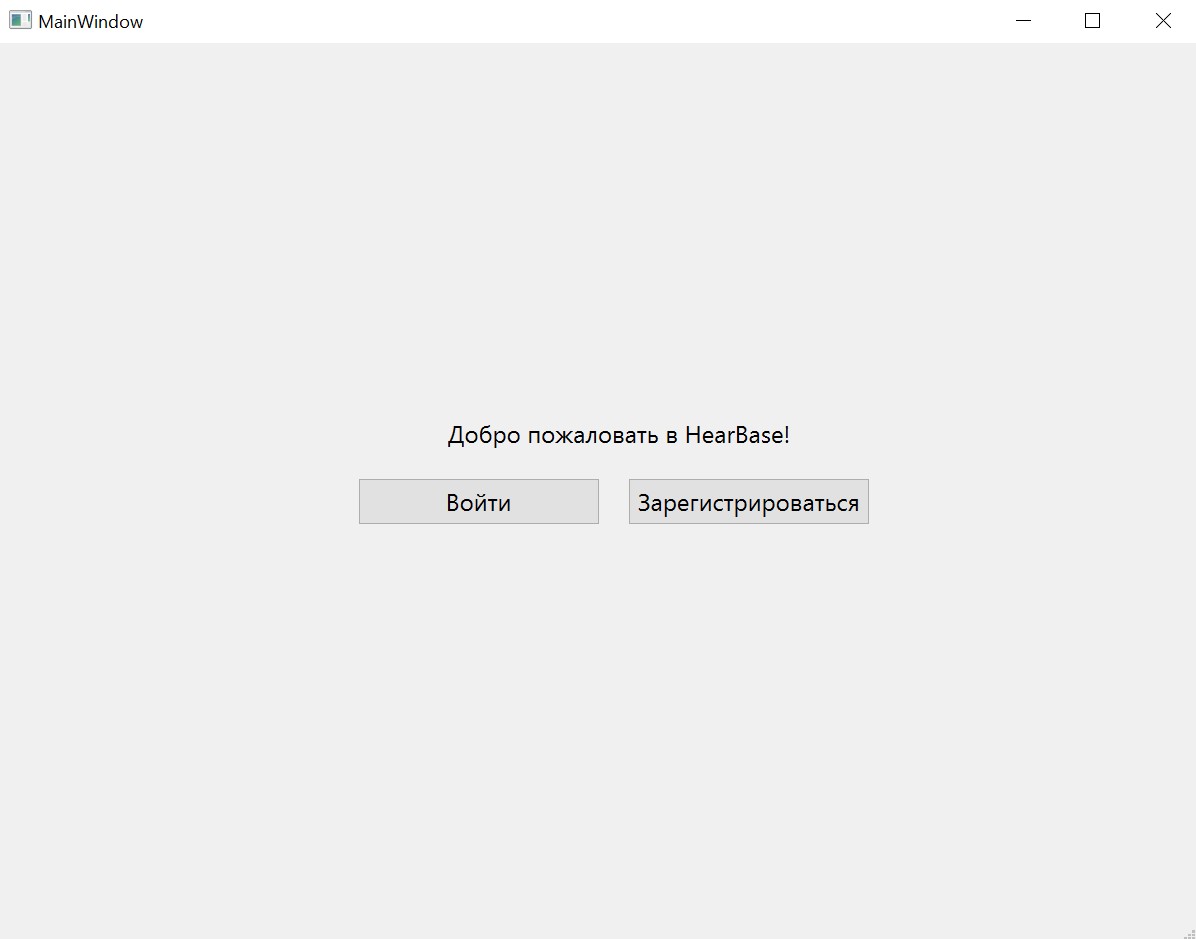


Рис. 3.1: Окно с приглашением

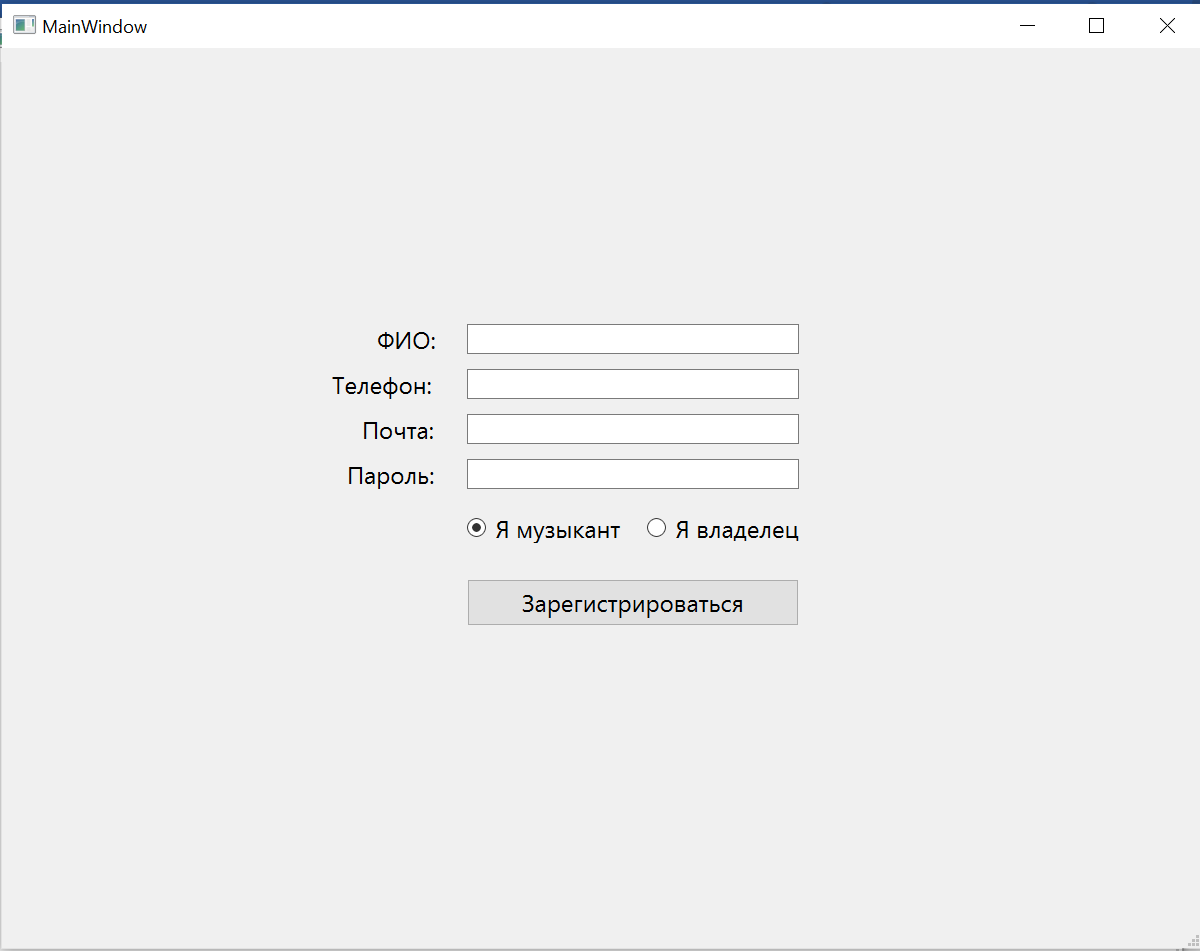


Рис. 3.2: Окно регистрации

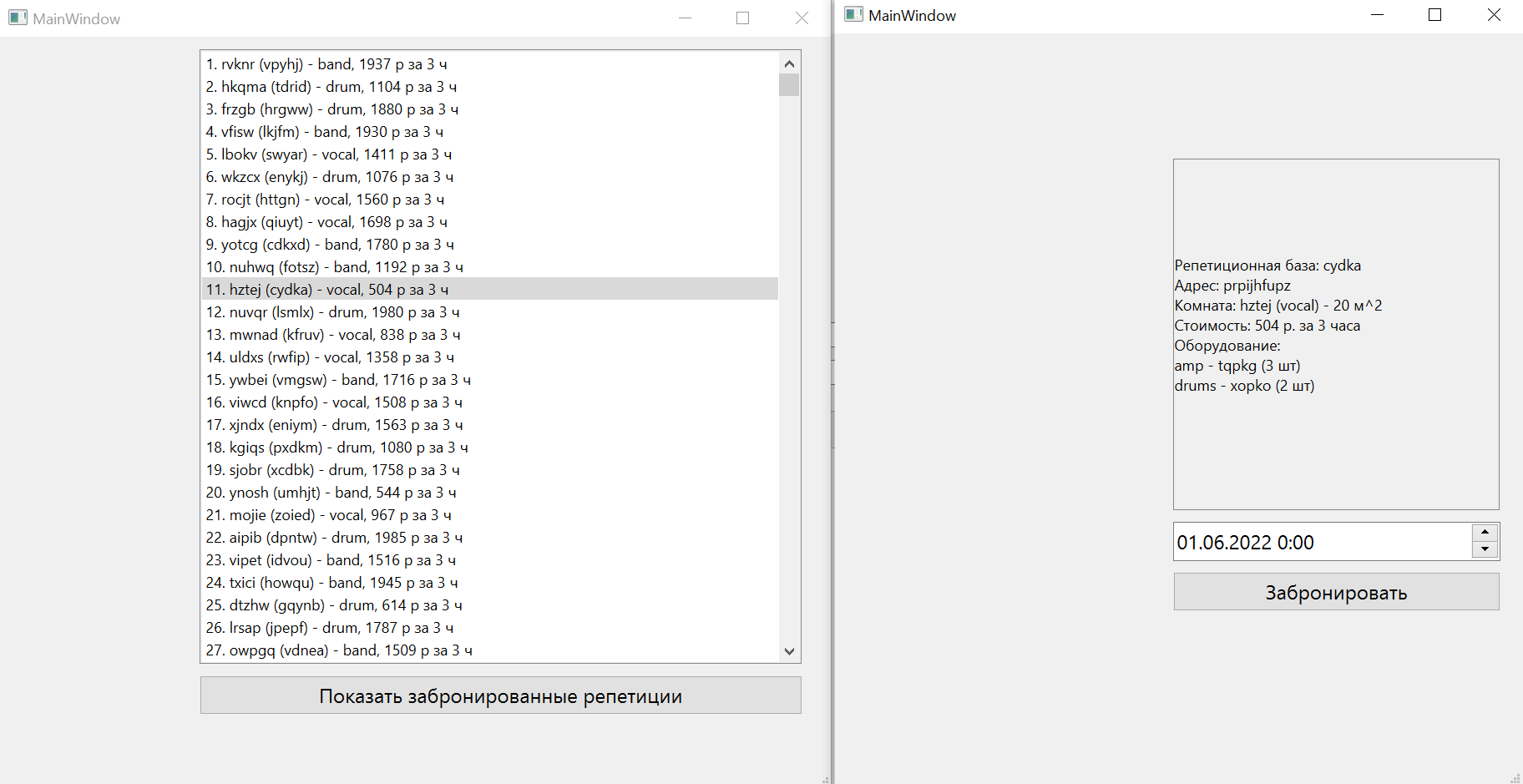


Рис. 3.3: Интерфейс при входе в качестве музыканта

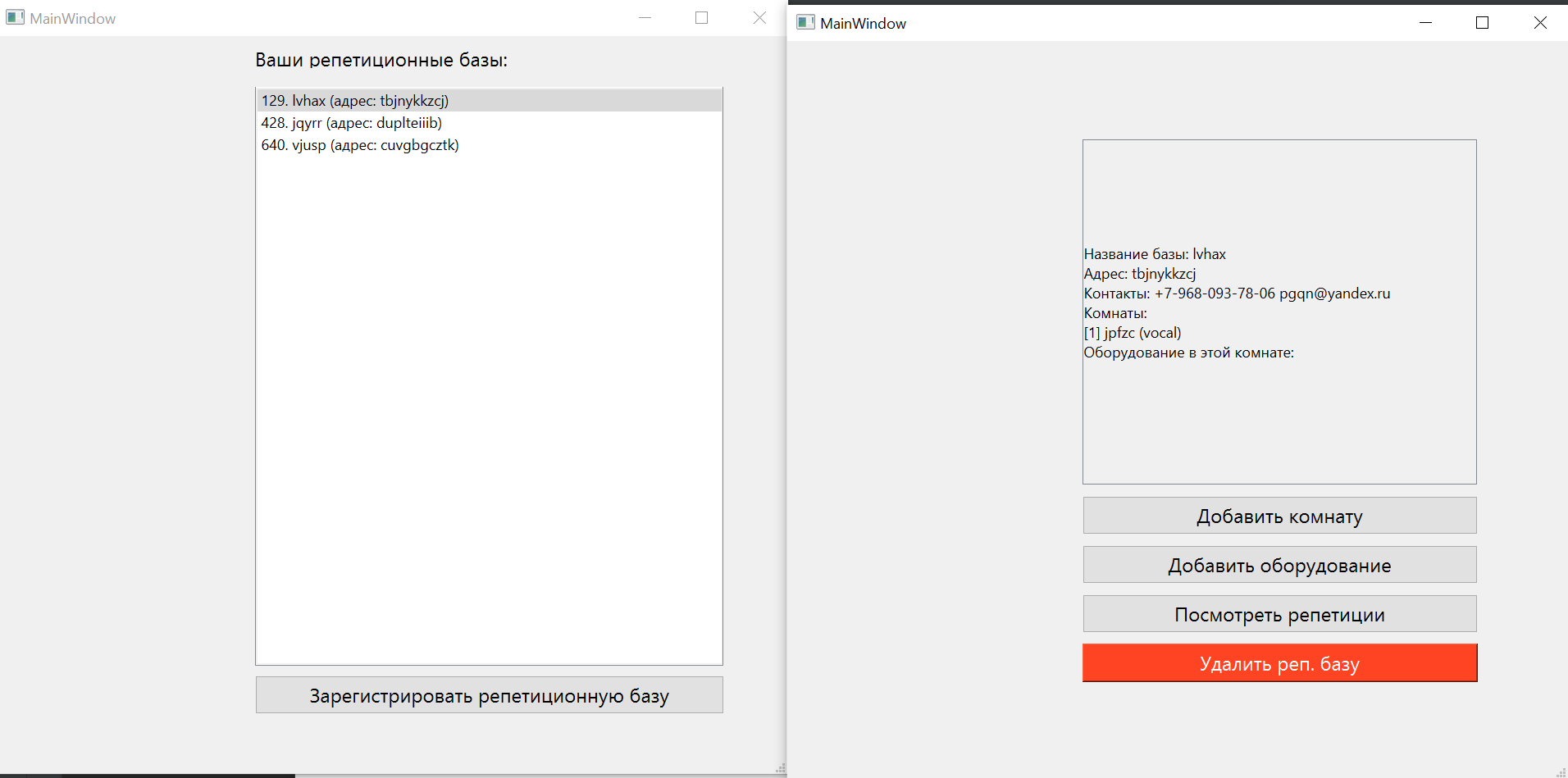


Рис. 3.3: Интерфейс при входе в качестве владельца

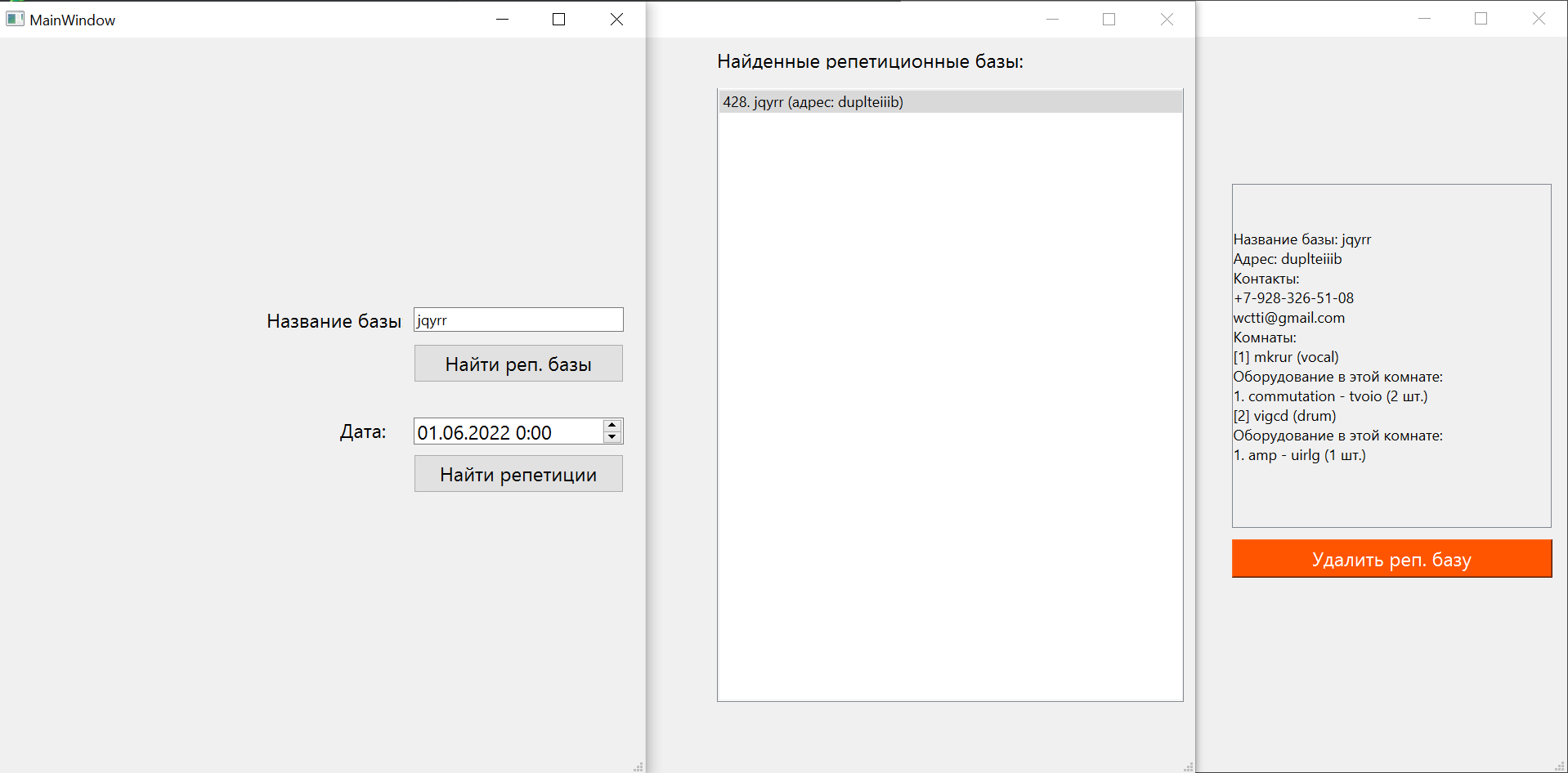


Рис. 3.4: Интерфейс при входе в качестве администратора

**Выводы**

В данном разделе были выбраны СУБД, язык программирования и среда разработки.

В качестве СУБД выбрана PostgreSQL, так как она:

* свободно распространяемая;
* полностью соответствует требованиям ACID;
* поддерживает все необходимые в рамках поставленной задачи функции (такие, как вложенные селекты, транзакции, триггеры, процедуры).

В качестве языка программирования был выбран Python, так как он:

* поддерживает работу с PostgreSQL;
* объектно-ориентированный;
* обладает необходимыми расширениями для работы с графическим интерфейсом.

В качестве среды разработки был выбран PyCharm, так как он:

* имеет свободно распространяемую версию;
* содержит множество удобств для написания и отладки кода, а также для работы с СУБД.

Помимо этого, в данном разделе был разработан и протестирован исходный код программы. Программа тестировалась в соответствии с этапами, приведёнными в разделе 2.4. Все тесты были успешно пройдены.

**4 Исследовательская часть**

**Список литературы**

1. ISO/IEC TR 10032:2003 Information technology – Reference model of data management.
2. Дейт К. Дж. Введение в системы баз данных. – 8-е изд. – М.: «Вильямс», 2006.
3. Еленев Д.В. и др. Автоматизация системы управления национальным исследовательским университетом и мониторинга его деятельности // Программные продукты и системы, №3, 2012.
4. PostgreSQL: Documentation: 11: Procedural Languages. [Электронный ресурс]. Режим доступа: <https://www.postgresql.org/docs/11/xplang.html>, (дата обращения: 28.04.2022).
5. What is PyQt? [Электронный ресурс]. Режим доступа: <https://riverbankcomputing.com/software/pyqt/intro> (дата обращения: 20.05.2022).
6. Yogesh Rana. Python: Simple though an Important Programming language // International Research Journal of Engineering and Technology (IRJET), 2019.