

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

Факультет информационных технологий
Кафедра «Инфокогнитивные технологии»

КУРСОВОЙ ПРОЕКТ

на тему: *«Разработка клиент-серверного desktop-приложения для
поддержки бизнес-процессов компании».*

Направление подготовки 09.03.03 «Прикладная информатика»
Профиль «Корпоративные информационные системы»

Выполнил:

студент группы 221-361

Дубровских Никита Евгеньевич

01.01.2023

(подпись)

Москва 2023

Введение

В современном бизнес-мире эффективное управление бизнес-процессами является ключевым фактором успеха для компаний во всех отраслях. Особенно важным аспектом является обеспечение бесперебойной и эффективной работы организации, специализирующейся в области ветеринарной медицины. Ветеринарные клиники, подобно другим предприятиям, сталкиваются с множеством задач, связанных с управлением клиентскими данными, записью на приемы и другими задачами, требующими автоматизации.

Разработка клиент-серверного desktop-приложения для ветеринарной клиники может значительно упростить управление бизнес-процессами и повысить качество предоставляемых услуг. Приложение может предоставить возможность эффективного планирования приемов, ведения медицинских карт животных.

Такое приложение может быть разработано в виде клиент-серверной системы, где клиентская часть будет установлена на любых устройствах с доступом в сеть Интернет, а серверная часть будет обеспечивать хранение и обработку данных. Это позволит сотрудникам клиники иметь доступ к необходимой информации в режиме реального времени и облегчит координацию работы между различными отделами и специалистами. В свою очередь клиенты клиники смогут быстро записаться на приём к ветеринару, не выходя из дома.

В результате разработки такого клиент-серверного desktop-приложения, ветеринарная клиника сможет значительно повысить эффективность своих бизнес-процессов, сократить время, затрачиваемое на управление информацией и повысить уровень обслуживания клиентов.

1 Постановка задачи

Цель работы – разработать простое оконное клиент-серверное приложение, позволяющее пользователю посредством графического интерфейса и согласно предоставляемым ему правам доступа в ролевой модели обрабатывать данные, хранящиеся на сервере и моделирующие следующую предметную область:

1. Для каждого принимаемого животного необходимо сохранить его имя, породу (если имеется) и владельца. Каждому животному должен быть присвоен уникальный цифровой идентификатор.
2. Владельцы животных являются клиентами клиники. Для каждого владельца должны быть сохранены его имя, адрес и номер телефона, а также сгенерирован уникальный числовой идентификатор.
3. Животное может быть бездомным, то есть не иметь хозяина.
4. Клиника должна иметь возможность хранить информацию о конкретной породе животного, даже если ни одно животное этой породы не проходило лечение в клинике.
5. На каждом приеме всегда присутствует ответственный врач. Все встречи начинаются в определенную дату и время, и на них присутствуют животное и его владелец.
6. Каждый врач, работающий в клинике, должен иметь сохраненные данные, включающие его имя, адрес и номер телефона, а также сгенерированный уникальный числовой идентификатор.
7. Во время приема животного могут быть обнаружены несколько заболеваний. Каждое заболевание имеет общее и научное название. В клинике не может быть двух заболеваний с одинаковым научным названием.
8. Клиника должна иметь возможность хранить информацию о наиболее распространенных заболеваниях для каждой отдельной породы.

Программное решение должно соответствовать следующим техническим требованиям:

1. Средства разработки – язык программирования Java и система управления базами данных MySQL, графическая библиотека – JavaFX.
2. Разработка программы должна сопровождаться ведением удаленного репозитория посредством системы контроля версий Git.
3. Структура кода программы должна строго соответствовать шаблону проектирования MVC (Model-View-Controller). В программе может быть несколько моделей, для каждой из них следует разработать по одному контроллеру и представлению.
4. База данных должна быть создана на сервере, доступ к которому предоставляется через fit.mospolytech.ru.
5. Каждая таблица базы данных должна удовлетворять нормальной форме Бойса-Кодда.
6. Должны быть предприняты исчерпывающие меры для обеспечения целостности данных при эксплуатации базы данных.
7. Должна быть реализована система регистрации и авторизации пользователей. В базе данных должны храниться хеш-значения паролей, но не сами пароли. Пользователь должен иметь возможность изменять свои регистрационные данные.
8. Для пользователей с различными привилегиями должен быть реализован соответствующий функционал в концепции CRUD (Create-Read-Update-Delete), доступный через графический интерфейс пользователя.
9. Запросы к базе данных должны быть параметрическими (с целью защиты от SQL-инъекций).

10. Класс для подключения к базе данных должен соответствовать шаблону проектирования Singleton (объект класса должен создаваться гарантированно единожды).
11. Результат запроса к базе данных должен «оборачиваться» в объект специально разработанного класса.
12. Должен быть создан исполняемый JAR-файл программы.

Задачи:

1. Изучить предметную область и на ее основании построить инфологическую модель базы данных.
- 2.

2 Проектирование и разработка приложения

Как видно из Рисунок 1: на протяжении всей работы, разработка программы сопровождалась ведением удаленного репозитория посредством системы контроля версий Git. Ссылка на репозиторий с проектом: <https://github.com/ssushnost/course1>

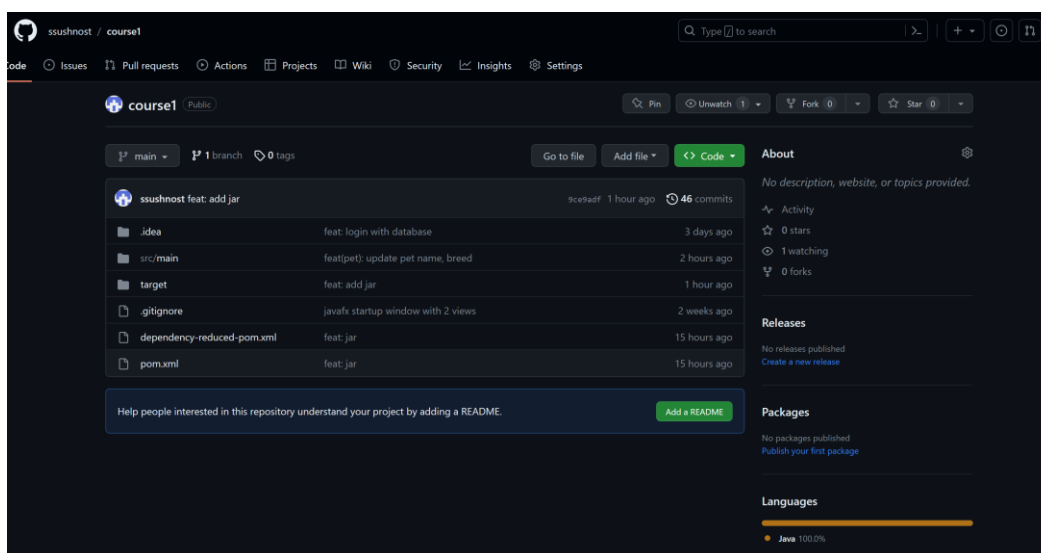


Рисунок 1. Репозиторий проекта на сайте <https://github.com>

Для информационной системы была разработана инфологическая модель базы данных (Рисунок 2**Error! Reference source not found.**). Анализируя предметную область можно выделить сущности Person, Pet, Breed, Disease, Appointment. Они представляют собой определенную концепцию, каждый экземпляр сущности однозначно определяется его атрибутами.

Проанализируем связи между сущностями:

1. Один человек может иметь много питомцев, один питомец может иметь только одного хозяина.
2. Один человек может записаться на несколько приёмов, один приём может иметь несколько человек.
3. Один приём может иметь несколько питомцев, один питомец может участвовать в нескольких приёмах одновременно
4. На одном приёме может быть выявлено несколько заболеваний, одно и тоже заболевание может быть выявлено на разных приёмах.
5. Один питомец может иметь только одну породу, одна порода может быть одновременно у несколько питомцев.
6. Одна порода может иметь несколько распространенных заболеваний, одно заболевание может быть распространено среди многих пород.

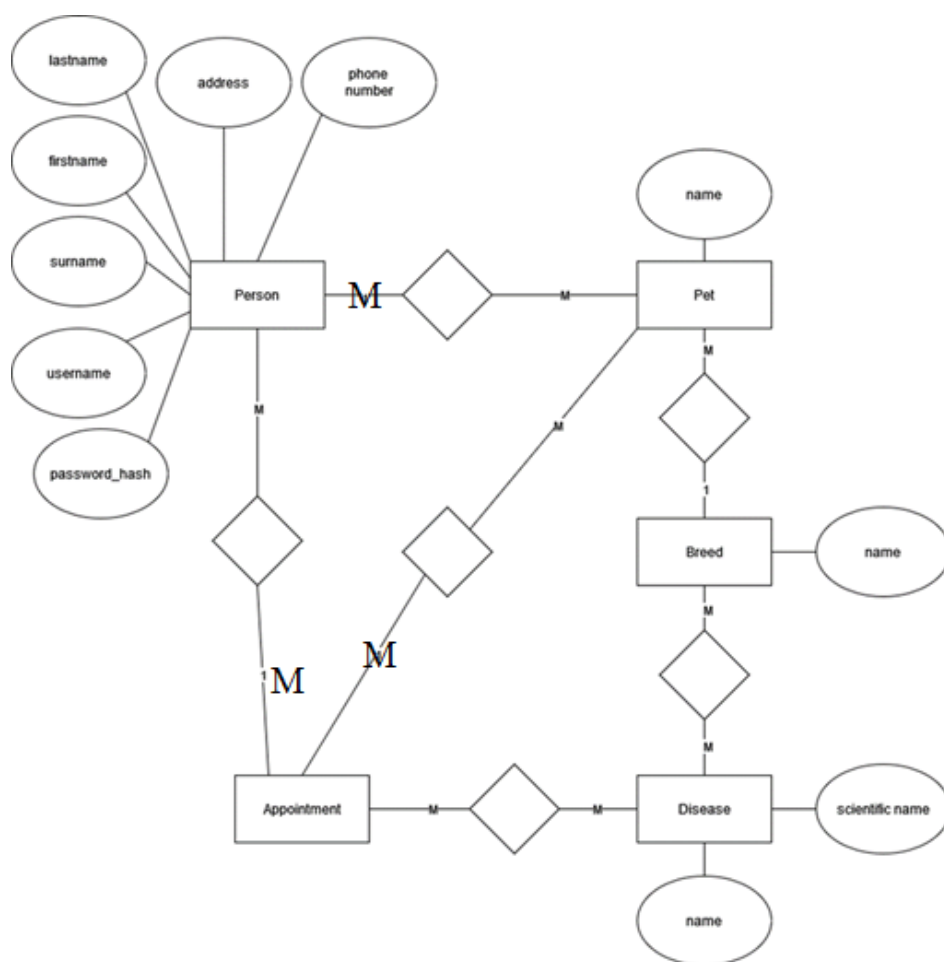


Рисунок 2. Инфологическая модель

Проект использует MySQL в качестве СУБД. База данных создана на сервере, доступ к которому осуществляется через сайт fit.mospolytech.ru. На Рисунок 3 представлена реляционная модель базы данных проекта.

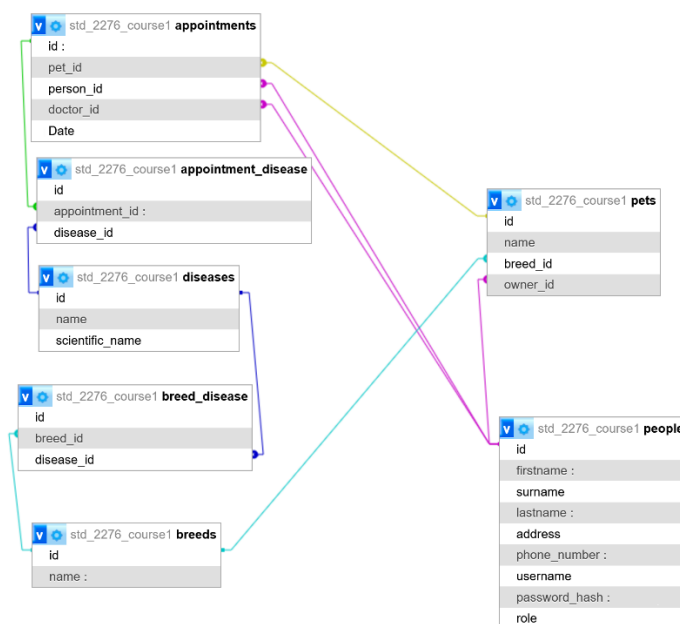


Рисунок 3. Реляционная модель

Ролевое управление доступом реализовано с помощью атрибута «role» в таблице people, который выдаёт пользователю доступ к частям программы, предназначенным именно для него. Программное решение имеет отдельное представление для каждой роли (Рисунок 4), которое загружается только когда пользователь авторизуется в аккаунт имеющий соответствующую роль (Рисунок 5). Так роль «client» выдаётся пользователем, не являющимся работниками клиники, и предоставляет им возможность добавить своих питомцев, записаться на приём (На Рисунок 7 представлено соответствующее роли представление, реализующее соответствующий функционал). Роль «admin» позволяет зарегистрировать в системе работников клиники (Рисунок 7), таких как доктор (имеющий роль «doctor») и обслуживающий персонал («employee»). Обслуживающий персонал, как показано на Рисунок 8, имеет возможность записать посетителя на приём (добавить данные посетителя в базу данных, добавить его питомца, выбрать дату приёма, указать лечащего доктора).


```
</> admin.fxml  
</> authorization.fxml  
</> client.fxml  
</> connecting.fxml  
</> employee.fxml  
</> pet.fxml  
</> registration.fxml
```

Рисунок 4. Представления проекта

```
private void loadCorrespondingView() {  
    AuthenticatedUser authenticatedUser = AuthenticatedUser.getInstance();  
    try {  
        App.setRoot(authenticatedUser.getRole());  
    } catch (IOException e) {  
        throw new RuntimeException(e);  
    }  
}
```

Рисунок 5. Метод загружающий, соответствующее роли представление

[Log out](#)

Welcome, {firstname} {lastname}

Add an employee

Surname

Ivanov

Firstname

Ivan

Lastname

Ivanovich

Username

vanya1561

Password

Role

Choose... ▼

add

Рисунок 6. Представление админ-панели

Menu

Log out

Welcome, Никита Евгеньевич

Sign up for an appointment

Pet

Choose...

Doctor

Choose...

Submit

Your appointments

Pet	Doctor	Date
No content in table		

Рисунок 7. Представление клиентской части

11

Add personAdd petAdd appointment

Log out

Welcome, {firstname} {lastname}

Sign up for an appointment

date

PersonChoose...

PetChoose...

DoctorChoose...

Submit

Persons appointments

Pet	Doctor	Date
No content in table		

Рисунок 8. Представление обслуживающего персонала

В качестве шаблона проектирования использована концепция MVC (Model-View-Controller) в комбинации с шаблоном Observer. В качестве View использованы файлы на языке разметки FXML – созданном специально для JavaFX и основанном на одном из самых популярных языков разметки XML, - позволяющие удобно описывать компоненты приложения, их расположение относительно друг друга.

Рассмотрим реализацию шаблона проектирования MVC в проекте на примере клиентской части (добавление питомца, запись на приём). Реализация MVC состоит из класса-контроллера Controller.Client, класса-модели Model.Client, представления client.fxml.

Контроллер Client реализует интерфейс Initializable, позволяющий ему переопределить метод initialize (Рисунок 9), который вызывается сразу после отображения представления. В этом методе контроллер устанавливает связь между View и Model с помощью методов bindBidirectional, setItems, addListener, setCellValueFactory. Эти методы ставят в соответствие компонента представления (например таблица записей на приём для авторизованного пользователя) текущее состояние модели (осуществляемое посредством ObservableList – Рисунок 10). Как только меняются элементы ObservableList – обновляется соответствующий элемент представления.

```
1 usage  ▲ ssushnost
public void onClickLogout() {
    model.LogOut();
}

▲ ssushnost
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    welcomeLabel.textProperty().bindBidirectional(model.welcomeText);
    petComboBox.setItems(model.pets);
    petComboBox.valueProperty().addListener((observable, oldValue, newValue) -> model.setPet(newValue));
    doctorComboBox.setItems(model.doctors);
    doctorComboBox.valueProperty().addListener((observable, oldValue, newValue) -> model.setDoctor(newValue));
    petColumn.setCellValueFactory(new PropertyValueFactory<>("petName"));
    doctorColumn.setCellValueFactory(new PropertyValueFactory<>("doctorName"));
    dateColumn.setCellValueFactory(new PropertyValueFactory<>("dateString"));
    appointmentsTableView.setItems(model.appointments);
    datePicker.setDayCellFactory(model::disablePastDates);
    datePicker.valueProperty().addListener((observable, oldValue, newValue) -> model.setDate(newValue));
}

1 usage  ▲ ssushnost
public void onClickSubmit() {
    model.addAppointment();
}

1 usage  ▲ ssushnost
public void onClickPetMenuItem() {
    model.setPetView();
}
```

Рисунок 9. Фрагмент класса-контроллера Client

```

public class Client {
    4 usages
    private final AuthenticatedUser authenticatedUser = AuthenticatedUser.getInstance();
    1 usage
    public StringProperty welcomeText = new SimpleStringProperty("Welcome, " + authenticatedUser.getWelcomeName());
    public ObjectProperty<Other.Pet> pet = new SimpleObjectProperty<>();
    2 usages
    public ObjectProperty<Person> doctor = new SimpleObjectProperty<>();
    2 usages
    public ObjectProperty<LocalDate> date = new SimpleObjectProperty<>();
    4 usages
    public ObservableList<Appointment> appointments;
    3 usages
    public ObservableList<Person> doctors;
    public ObservableList<Other.Pet> pets;
    1 usage
    Database database = Database.getInstance();
    4 usages
    Connection connection = database.getConnection();
}

```

Рисунок 10. Фрагмент класса-модели Client

Таким образом представление отображает состояние модели, а контроллер связывает модель и представление, а также обрабатывает события представления (например нажатие кнопки «sing in»(войти в систему)).

2.6 Класс App

Класс App (Рисунок 11) является ключевым классом, так как является точкой входа в программу. Он наследует класс `javafx.application.Application` и переопределяет его метод `start()`, с помощью которого в JavaFX инициализируется и отображается окно приложения.

Метод `call()`: Это статический метод, который вызывает метод `launch()` из класса `Application`, запуская приложение JavaFX.

Метод `setRoot(String fxml)`: Этот метод устанавливает корневой элемент сцены, основываясь на переданном имени файла FXML. Он загружает FXML-файл с помощью метода `loadFXML()` и устанавливает его в качестве корневого элемента сцены.

Метод `start(Stage stage)`: Этот метод переопределяет метод `start()` из класса `Application` и выполняется при запуске приложения JavaFX. Он создает новую сцену и устанавливает ее размеры. Затем он загружает FXML-файл "connecting", в котором содержится описание представления, отвечающего за

отображение процесса и результата подключения к базе данных, с помощью метода loadFXML() и устанавливает его в качестве корневого элемента сцены.

Метод loadFXML(String fxml): Этот метод загружает FXML-файл, основываясь на переданном имени файла. Он создает экземпляр класса FXMLLoader и использует его для загрузки FXML-файла, связывая его с классом App. Затем он вызывает метод load() для загрузки файла и возвращает корневой элемент, который был создан из FXML-файла.

```
ssushnost
public class App extends Application {

    3 usages
    private static Scene scene;

    1 usage  ssushnost
    public static void call() {
        launch();
    }

    10 usages  ssushnost
    public static void setRoot(String fxml) throws IOException {
        scene.setRoot(loadFXML(fxml));
    }

    ssushnost
    @Override
    public void start(Stage stage) throws IOException {
        scene = new Scene(loadFXML("connecting"), v: 700, v1: 600);
        stage.setScene(scene);
        stage.show();
    }

    2 usages  ssushnost
    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource("name: fxml + ".fxml"));
        return fxmlLoader.load();
    }
}
```

Рисунок 11. Класс App – точка входа в программу

После того как программа подключилась к базе данных загружается окно авторизации (authorization.fxml), представленное на Рисунок 12, предназначенное для ввода имени пользователя (username), пароля (password). Оно предоставляет пользователю возможность войти в систему, проверяет правильность введенных данных с данными, хранящимися в базе данных.

Хэширование паролей обеспечивает безопасность пользовательских данных, защищая аккаунт от несанкционированного доступа и повышает общий уровень безопасности системы.

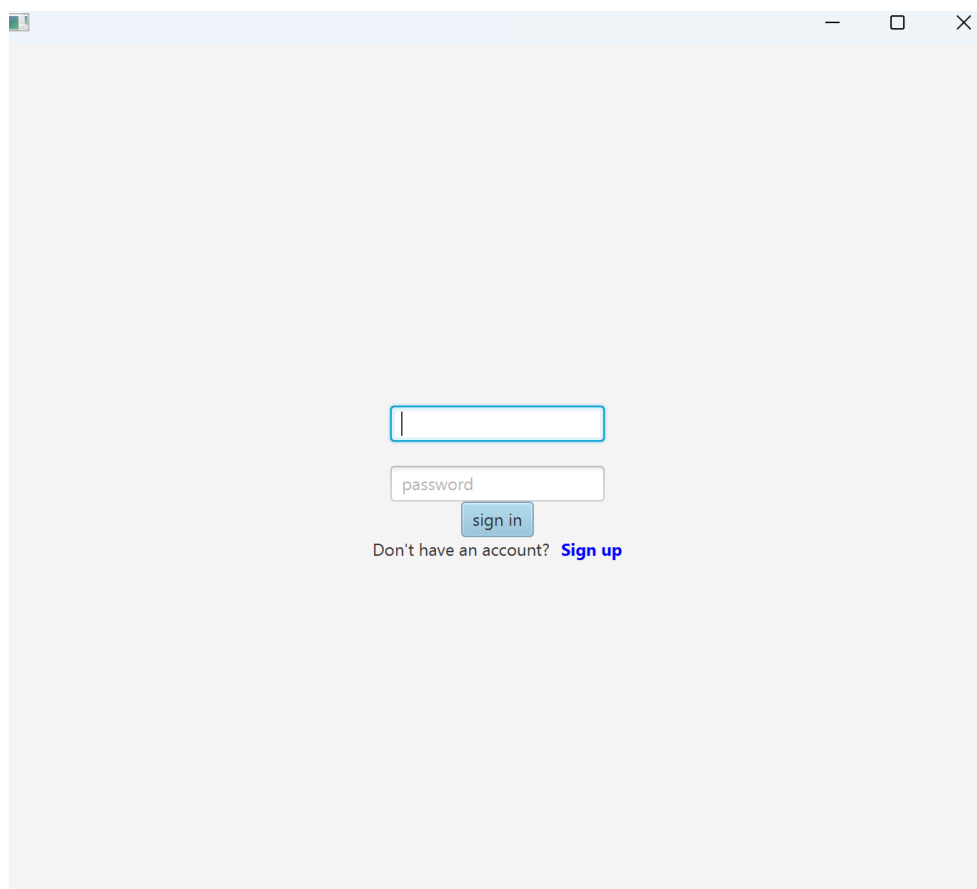
A screenshot of a web application's login interface. It features a light gray background with a centered login form. The form consists of two input fields: the top one is empty, and the bottom one is labeled 'password'. Below the password field is a blue 'sign in' button. Underneath the button, the text 'Don't have an account?' is followed by a blue 'Sign up' link. The entire interface is contained within a window with standard OS window controls (minimize, maximize, close) at the top right.

Рисунок 12. Представление авторизации

С окна авторизации можно перейти на вкладку регистрации (Рисунок 13) посредством нажатия кнопки «Sign up» (зарегистрироваться).

Только после заполнения всех необходимых полей пользователь может нажать на кнопку «sign up» (зарегистрироваться), тем самым внося себя в базу данных и завершив процесс регистрации, создавать новую учетную запись в системе.

После регистрации пользователь перенаправляется на окно авторизации, где может войти в систему, используя данные указанные при регистрации.

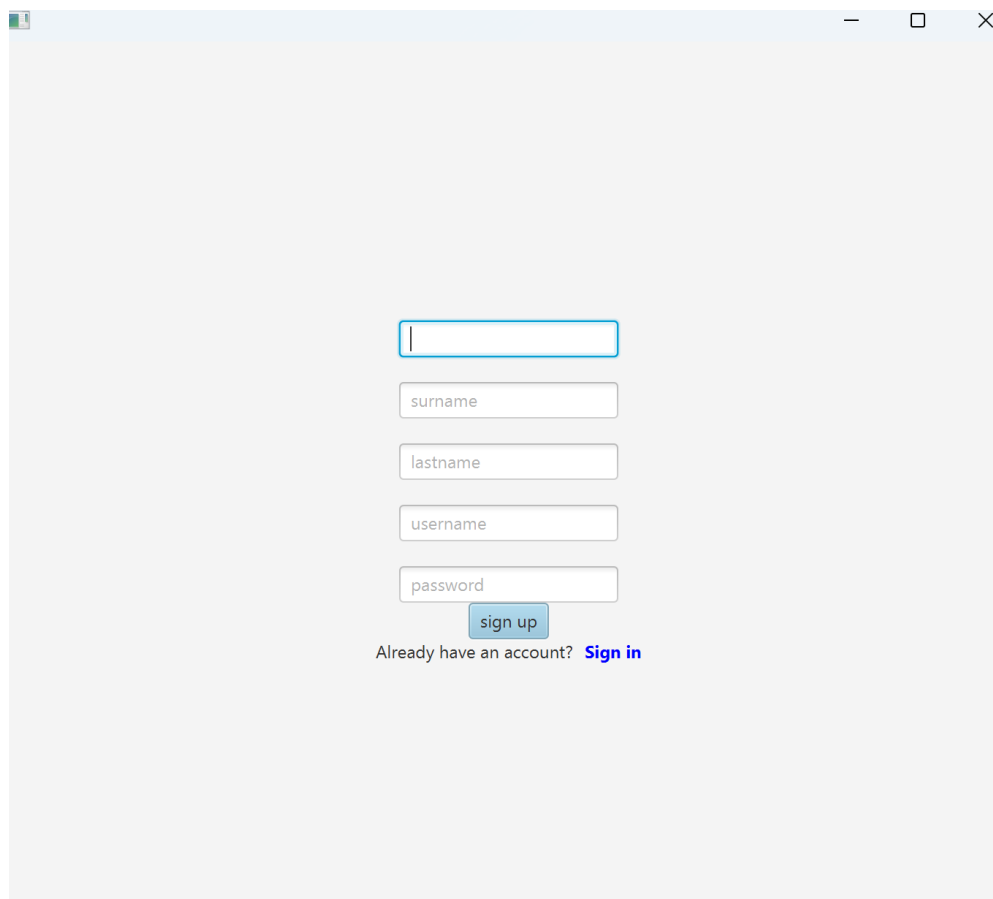
A screenshot of a web registration form. It features a light gray background with a central form area. The form includes five input fields: a first field (likely for email) with a blue border, followed by 'surname', 'lastname', 'username', and 'password'. Below these is a blue 'sign up' button. At the bottom, there is a link that says 'Already have an account? Sign in'.

Рисунок 13. Представление регистрации

Класс Database (Рисунок 14) является ключевым в приложении, так как он отвечает за подключение к базе данных и без него приложение не являлось бы клиент-серверным.

Основные методы и функции класса Database:

`getInstance()`: Статический метод, который возвращает единственный экземпляр класса Database с использованием шаблона проектирования Singleton. Если экземпляр еще не создан, метод создает его и устанавливает соединение с базой данных.

`connect()`: Метод, который устанавливает соединение с базой данных MySQL. Он использует драйвер JDBC для загрузки соответствующего драйвера базы данных и создания объекта Connection. В этом методе указываются

параметры подключения к базе данных, такие как URL базы данных, имя пользователя и пароль.

`getConnection()`: Метод, который возвращает объект `Connection`, предоставляющий доступ к базе данных. Этот объект может быть использован для выполнения запросов SQL и получения результатов из базы данных.

```
private Database() {
    try {
        Class.forName( className: "com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException exception) {
        System.out.println("Database Connection Creation Failed : " + exception.getMessage());
    }
}

1 usage  ssushnost
public static boolean connect() {
    System.out.println("connecting to database...");
    try {
        connection = DriverManager.getConnection(url, user, password);
    } catch (SQLException e) {
        System.out.println("can't connect to database");
        return false;
    }
    System.out.println("successfully connected to database");
    return true;
}

ssushnost
public static Database getInstance() {
    try {
        if (instance == null)
            instance = new Database();
        else if (connection.isClosed())
            instance = new Database();
        return instance;
    } catch (SQLException exception) {
        throw new RuntimeException(exception);
    }
}
```

Рисунок 14. Класс Database

Проект реализует концепцию CRUD. CRUD - это аббревиатура, которая обозначает четыре основные операции, используемые при взаимодействии с базой данных: Create (Создание), Read (Чтение), Update (Обновление) и Delete (Удаление). Эти операции представляют собой основные функциональности,

необходимые для управления данными в базе данных. Рассмотрим их реализацию на примере класса Pet:

Операции CRUD представлены в виде кнопок на окне добавление питомца для клиента, изображенного на Рисунок 15. Так Create – кнопка Submit, добавляющая питомца, Read – происходит автоматически при добавлении, удалении, обновлении данных питомца, Update – кнопка Update, обновляющая поля питомца на указанные в поле данных, выпадающем списке, Delete – кнопка Delete, удаляющая выбранного питомца из базы данных.

Menu

Welcome, Никита Евгеньевич

[Log out](#)

Add your pet

Breed Пудель

My pets

name	breed
ouae	Пудель

Рисунок 15. Представление добавление питомца в части клиента

1. Create (Создание):

- Метод `addPet()`, изображенный на Рисунок 16, отвечает за создание новой записи о питомце в базе данных.

- Перед добавлением питомца, метод проверяет, что выбрана порода (`breed.get() != null`) и имя питомца не пустое (`!name.get().isBlank()`).
- Затем создается SQL-запрос
- После выполнения запроса метод обновляет список `pets`, считывая данные из базы данных с помощью метода `readPetsFromDatabase()`.

```
public void addPet() {
    if (breed.get() == null || name.get().isBlank())
        return;
    String statement = "insert into pets (name, breed_id, owner_id) value (?, ?, ?)";
    try (PreparedStatement preparedStatement = connection.prepareStatement(statement)) {
        preparedStatement.setString(1, name.get());
        preparedStatement.setInt(2, breed.get().getId());
        preparedStatement.setInt(3, authenticatedUser.getId());
        preparedStatement.executeUpdate();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    readPetsFromDatabase();
}
```

Рисунок 16. Метод `addPet`, реализующий операцию *Create*

2. Read (Чтение):

- Метод `readPetsFromDatabase()`, изображенный на Рисунок 17, отвечает за чтение всех питомцев из базы данных.
- Создается SQL-запрос с использованием параметра `owner_id`, чтобы выбрать только питомцев, принадлежащих текущему аутентифицированному пользователю.
- Параметр `owner_id` устанавливается с помощью метода `setInt()`.
- Затем выполняется запрос, и результаты выборки сохраняются в список `pets`, созданный с помощью `ObservableList`.

```

public void readPetsFromDatabase() {
    pets.clear();
    String statement = "select * from pets where owner_id = ?";
    try (PreparedStatement preparedStatement = connection.prepareStatement(statement)) {
        preparedStatement.setInt(1, authenticatedUser.getId());
        ResultSet resultSet = preparedStatement.executeQuery();
        while (resultSet.next()) {
            Other.Pet pet = Other.Pet.fromResultSet(resultSet);
            pets.add(pet);
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

```

Рисунок 17. Метод `readPetsFromDatabase`, реализующий операцию *Read*

3. Update (Обновление):

- Метод `updatePet(Other.Pet pet)`, изображенный на Рисунок 18: метод отвечает за обновление данных о питомце в базе данных.
- Он принимает объект `pet` типа `Other.Pet`, который содержит информацию о питомце, которую нужно обновить (выбирается из таблицы).
- Создается SQL-запрос
- Запрос обновляет имя питомца и его породу на указанные в текстовом поле и выпадающем списке.
- Параметры запроса устанавливаются с помощью методов `setString()`, `setInt()` и `setInt()`.
- Затем SQL-запрос выполняется с помощью метода `executeUpdate()`, и данные о питомце обновляются в базе данных.
- После обновления данных о питомце метод вызывает `readPetsFromDatabase()`, чтобы обновить список питомцев.

```

1 usage  ssushnost
public void updatePet(Other.Pet pet) {
    String statement = "update pets set name = ?, breed_id = ? where id = ?";
    try (PreparedStatement preparedStatement = connection.prepareStatement(statement)) {
        preparedStatement.setString( parameterIndex: 1, name.get());
        preparedStatement.setInt( parameterIndex: 2, breed.get().getId());
        preparedStatement.setInt( parameterIndex: 3, pet.getId());
        preparedStatement.executeUpdate();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    readPetsFromDatabase();
}

```

Рисунок 18 Метод updatePet, реализующий операцию Update

4. Delete (Удаление):

- Метод deletePet(Other.Pet pet), изображенный на Рисунок 19, отвечает за удаление записи о питомце из базы данных.
- Создается SQL-запрос с использованием параметра id питомца, полученного из выбранной строки таблицы с питомцами, которого необходимо удалить.
- После выполнения запроса на удаление метод удаляет питомца из списка pets с помощью метода remove(), для того что бы он пропал из таблицы (так как вид синхронизирован с моделью).

```

1 usage  ssushnost
public void deletePet(Other.Pet pet) {
    String statement = "delete from pets where id = ?";
    try (PreparedStatement preparedStatement = connection.prepareStatement(statement)) {
        preparedStatement.setInt( parameterIndex: 1, pet.getId());
        preparedStatement.executeUpdate();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    pets.remove(pet);
}

```

Рисунок 19. Метод deletePet, реализующий операцию Delete

В рамках данного проекта было принято решение использовать параметризованные запросы для взаимодействия с базой данных с целью предотвращения SQL-инъекций.

Основное преимущество использования параметризованных запросов заключается в предотвращении SQL-инъекций. Использование параметров вместо включения значений прямо в запрос позволяет базе данных правильно обрабатывать и экранировать значения, тем самым предотвращая возможность внедрения злонамеренного кода.

Кроме того, использование параметризованных запросов может повысить производительность. При использовании параметризованных запросов, база данных может скомпилировать запрос один раз и затем повторно использовать его с различными значениями параметров. Это снижает расходы на компиляцию запроса и может значительно улучшить производительность.

В результате использования параметризованных запросов (пример параметрического запроса, используемого в проекте представлен на Рисунок 20), приложение было обеспечено безопасностью обработки данных, предотвращены возможные атаки типа «SQL-инъекция», повышена производительность.

```
String statement = "insert into appointments (pet_id, person_id, doctor_id, date) value (?, ?, ?, ?)";
try (PreparedStatement preparedStatement = connection.prepareStatement(statement)) {
    preparedStatement.setInt(1, pet.getId());
    preparedStatement.setInt(2, authenticatedUser.getId());
    preparedStatement.setInt(3, doctor.getId());
    preparedStatement.setDate(4, Date.valueOf(date.get()));
    preparedStatement.executeUpdate();
} catch (SQLException e) {
    throw new RuntimeException(e);
}
```

Рисунок 20. . Пример параметрического запроса используемого в проекте

Упаковка приложения в исполняемый JAR-файл является распространенным способом предоставления и развертывания программного обеспечения. Вот некоторые особенности и преимущества его использования:

1. Сборка и зависимости: Упаковка приложения в JAR-файл позволяет включить все необходимые компоненты и зависимости, такие как библиотеки или другие внешние модули, в один файл. Это облегчает установку и развертывание приложения на других компьютерах или серверах без необходимости устанавливать каждую зависимость отдельно.
2. Переносимость: Исполняемый JAR-файл является платформонезависимым и может выполняться на любой платформе, где установлена Java Virtual Machine (JVM). Это означает, что вы можете разработать приложение на одной платформе и запускать его на других платформах без необходимости перекомпиляции или внесения изменений в исходный код.
3. Упрощенное развертывание: Исполняемый JAR-файл содержит все необходимые файлы и ресурсы, чтобы приложение могло быть самостоятельным. Пользователи могут просто запустить JAR-файл, чтобы запустить приложение, без необходимости настройки окружения или установки дополнительных компонентов.
4. Управление версиями: Используя JAR-файлы, можно управлять версиями приложения. Каждый JAR-файл может содержать информацию о версии приложения, что облегчает контроль версий и управление обновлениями.

На Рисунок 21 изображен результат запуска проекта, используя скомпилированный JAR-файл.

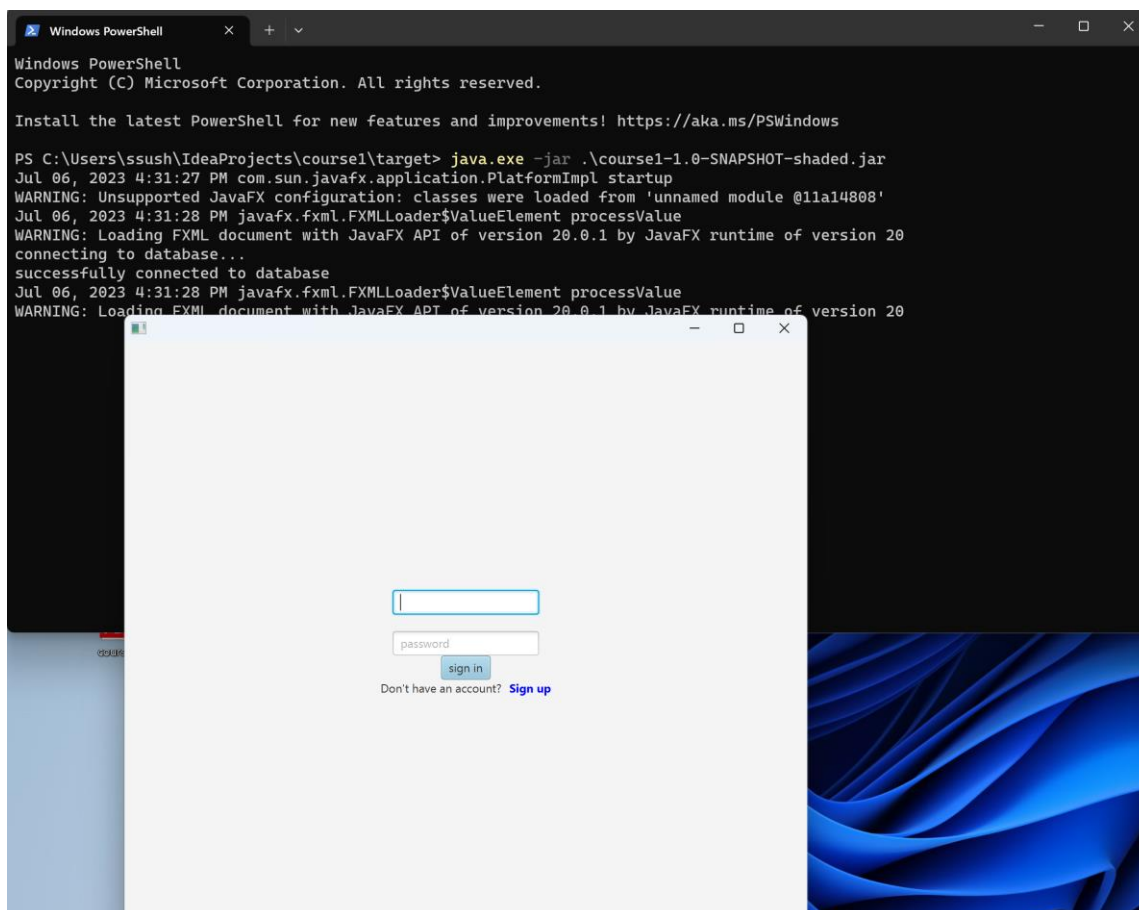


Рисунок 21. Запуск приложения, через JAR-файл

3 Тестирование и сценарии работы в приложении

Тест 1: Вход в систему используя неправильные данные.

Шаг 1: В окне аутентификации ввести правильное имя пользователя.

Шаг 2: Ввести неправильный пароль

Шаг 3: Нажать «Sing in» (Войти)

Ожидаемый результат: программа не даст пользователю войти

Фактический результат: программа не дала пользователю войти в систему и оповестила его о неправильно введенных входных данных (Рисунок 22)

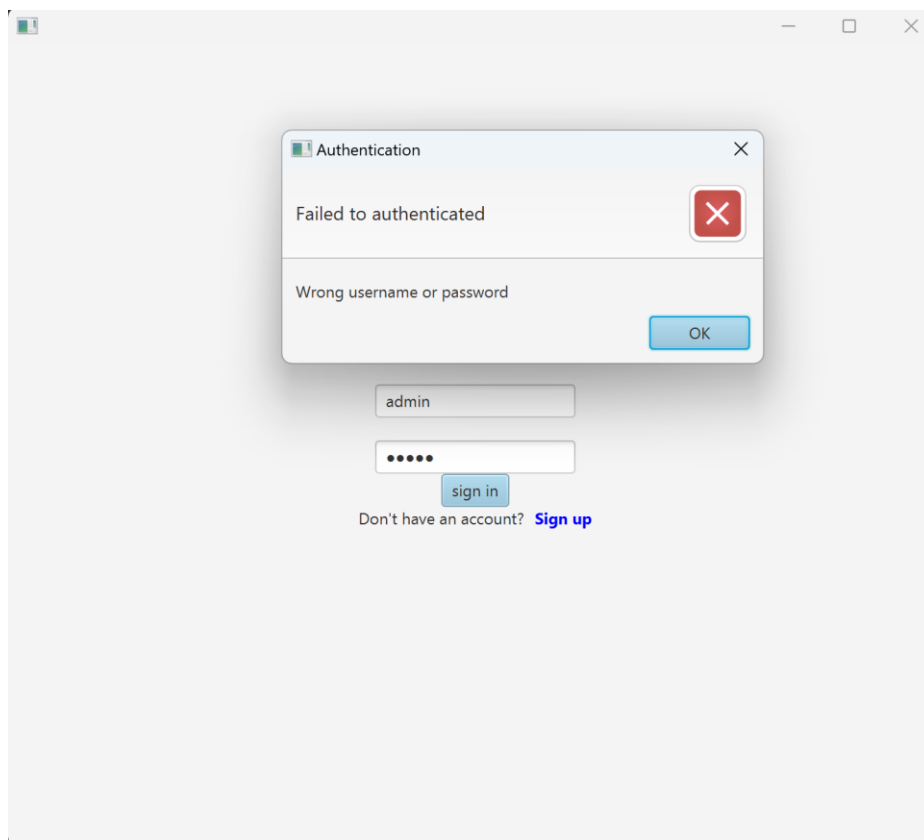


Рисунок 22. Фактический результат теста 1

Тест 2: Вход с пустым полем пароля.

Шаг 1: В окне аутентификации ввести имя пользователя.

Шаг 2: Поле пароля оставить пустым

Шаг 3: Нажать «Sing in» (Войти)

Ожидаемый результат: программа не даст пользователю войти

Фактический результат: программа не дала пользователю войти в систему и оповестила его о том, что он не ввёл пароль (Рисунок 23)

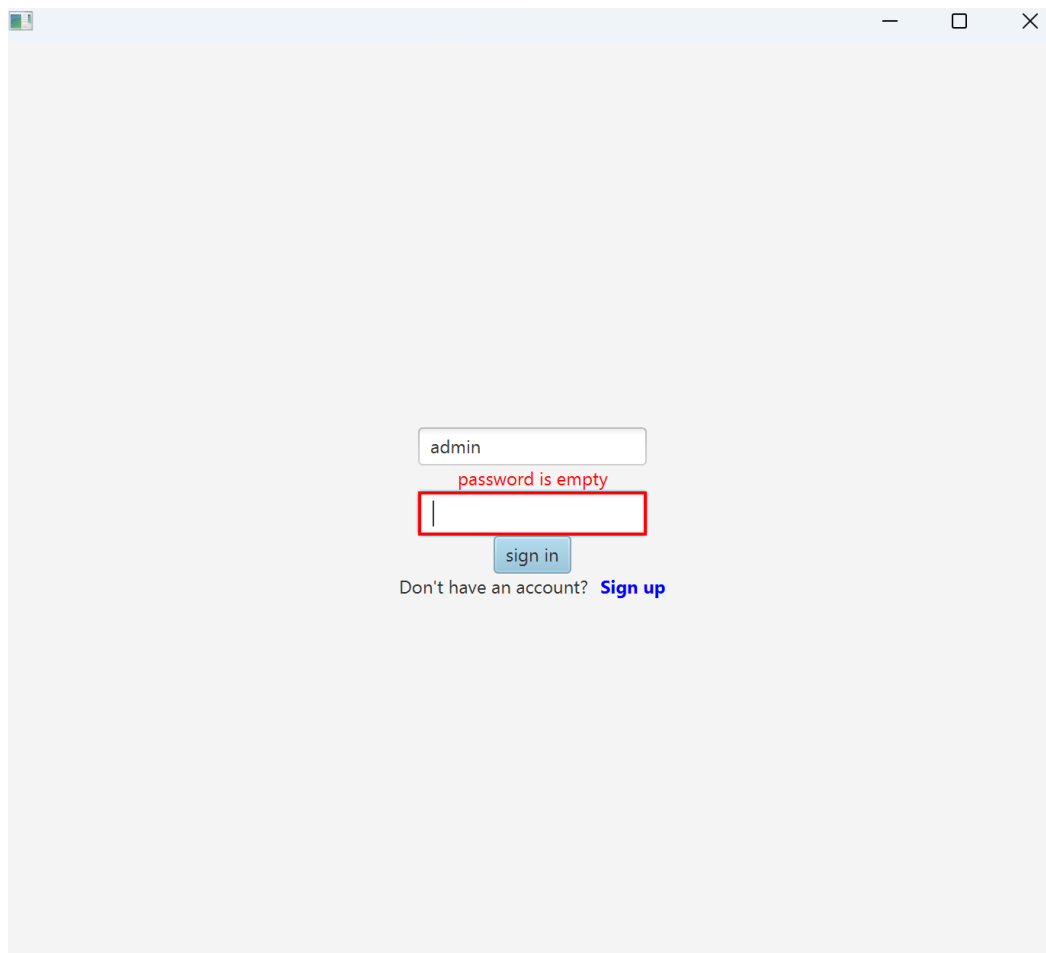


Рисунок 23. Фактический результат теста 2

Тест 3: Клиент добавляет питомца.

Шаг 1: Зарегистрироваться если не зарегистрированы.

Шаг 2: Авторизоваться используя учётные данные.

Шаг 3: Перейти во вкладку «My pets» используя меню.

Шаг 4: Ввести имя питомца в поле name.

Шаг 5: Выбрать породу питомца из выпадающего списка.

Шаг 6: Нажать «add»(добавить).

Ожидаемый результат: программа добавит питомца в базу данных и отобразит добавление

Фактический результат: программа добавила питомца в базу данных и отобразила его в таблице питомцев (Рисунок 23)

Menu

Log out

Welcome, Никита Евгеньевич

Add your pet

Жучка

Breed Корги

Submit

My pets

name	breed
Жучка	Корги

Рисунок 24. Фактический результат теста 3

Заключение

Результатом данного проекта является клиент-серверное desktop-приложение, которое способствует автоматизации процессов ветеринарной клиники, улучшению качества предоставляемых услуг и эффективности работы персонала. Приложение обладает интуитивным пользовательским интерфейсом, обеспечивает безопасность данных, удовлетворяет требованиям ролевой модели доступа и обеспечивает защиту от потенциальных уязвимостей.

Реализация данного проекта позволит ветеринарным клиникам оптимизировать свои бизнес-процессы, улучшить взаимодействие между сотрудниками и клиентами, а также повысить качество предоставляемых услуг. Отметим, что разработанное приложение может быть дальнейшей доработано и расширено в соответствии с потребностями конкретной клиники.

В процессе разработки данного проекта мы приобрели ценный опыт в области объектно-ориентированного программирования, проектирования баз данных и разработки графического интерфейса. Работа над проектом позволила углубиться в проблематику ветеринарной медицины и понять важность автоматизации бизнес-процессов для повышения эффективности работы ветеринарных клиник.

Список литературы и интернет-ресурсов

1. Документация JavaFX [Электронный ресурс]. URL: <https://www.oracle.com/java/technologies/javase/javafx-overview.html>
2. Kolade Chris CRUD Operations – What is CRUD? [Электронный ресурс]. URL: <https://www.freecodecamp.org/news/crud-operations-explained/>
3. Васильев А. А. "Java. Библиотека профессионала. Том 2. Расширенные средства программирования". Санкт-Петербург: БХВ-Петербург, 2019.
4. Сьерра К., Бейтс Б. "Изучаем Java". Санкт-Петербург: ДМК Пресс, 2018.
5. Хорстманн К., Корнелл Г. "Java. Библиотека профессионала. Том 1. Основы". Санкт-Петербург: Питер, 2019.
6. Фаулер М. "Архитектура корпоративных программных приложений". Москва: ДМК Пресс, 2017.
7. Блох Дж. "Java. Эффективное программирование". Санкт-Петербург: Питер, 2020.