Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет инженерно-экономический

Кафедра экономической информатики

Дисциплина «Распределенные системы обработки информации»

|  |  |
| --- | --- |
|  | «К ЗАЩИТЕ ДОПУСТИТЬ» |
|  | Руководитель курсового проекта  ассистент кафедры экономической информатики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. П. Лыщик |
|  | \_\_\_.\_\_\_\_.2023 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«РАЗРАБОТКА АвтоматизированнОЙ системЫ Работы МУЗЕЯ»**

|  |  |
| --- | --- |
|  | Выполнил студент группы 074002  Шейграсов Сергей Александрович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2023  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2023

|  |  |  |  |
| --- | --- | --- | --- |
| Перечень условных обозначений, символов и терминов………………... | | | 3 |
| Введение.………………………………………………………..…………... | | | 5 |
| 1 | Анализ и моделирование предметной области системы работы музея……………………………….…….……………….. | | 7 |
|  | 1.1 | Описание предметной области…………………………………... | 7 |
|  | 1.2 | Функциональное моделирование процесса работы музея на основе стандарта IDEF0………………......................................... | 9 |
|  | 1.3 | Информационная модель системы……………………………………………………………. | 13 |
|  | 1.4 | Модели представлений автоматизированной системы работы музея……………………………………………………………… | 15 |
| 2 | Проектирование и конструирование программного средства……….. | | 24 |
|  | 2.1 | Постановка задачи……………………………………………....... | 24 |
|  | 2.2 | Обоснование выбора компонентов и технологий для реализации системы работы музея……………………..……….. | 25 |
|  | 2.3 | Архитектурные решения…………………………......................... | 27 |
|  | 2.4 | Описание алгоритмов, реализующих ключевую бизнес-логику системы …………………………………….……………………... | 29 |
|  | 2.5 | Проектирование пользовательского интерфейса.......................... | 31 |
| 3 | Руководство развертывания и использования системы работы музея……………………………………………………… | | 33 |
| Заключение………………………………………………………………...... | | | 41 |
| Список использованных источников ……………………………............... | | | 42 |
| Приложение А Отчет о проверке на заимствования в системе «Антиплагиат»……………………………………………………………… | | | 43 |
| Приложение Б Листинг кода алгоритмов, реализующих бизнес-логику…........................................................................................................... | | | 44 |

# **ВВЕДЕНИЕ**

Информационные технологии в современном мире интенсивно развиваются и применяются повсеместно. Наиболее широко применяемые средства в наше время – это сотовая связь и интернет, мобильные телефоны и компьютеры. Интернет в особенности становится все более удобной площадкой для продвижения и удобного взаимодействия с общественностью. Большинство передовых отраслей активно развивают свои системы с помощью этих современных технологий.

Но помимо вышеперечисленных отраслей, маленькие отрасли и организации также нуждаются в соответствующем программном обеспечении. Все больше компаний стремятся предоставить свои товары и услуги в онлайн-среде.

Музей — учреждение, занимающееся сбором, изучением, хранением и экспонированием предметов — памятников естественной истории, материальной и духовной культуры, а также просветительской и популяризаторской деятельностью. Сейчас данные учреждения активно проводит экскурсии, активно знакомя людей со своим достоянием. Для этого музеям необходимо предоставить своим потенциальным посетителям всю необходимую информацию о проводимых экскурсиях, тех кто их проводит, времени и предоставляемых экспонатах. Помимо этого, необходимо обеспечить удобных формат взаимодействия с посетителями.

Таким образом, музеи также нуждаются в оптимизации своей деятельности посредством использования современных средств. В связи с этим необходимо создание веб-приложения, которое позволило бы упростить взаимодействие пользователей с организацией, а также упростить работу работников непосредственно организации.

Цель курсового проекта – совершенствование организации работы музея, прежде всего упрощения взаимодействия потенциальных посетителей с системой данного музея и оптимизация деятельности работников данного учреждения посредством создания соответствующего веб-приложения.

Из выше представленной цели вытекают следующие задачи:

* описать процессы работы музея;
* определить необходимые требования;
* разработать модели представления системы;
* создать базу данных для отслеживания деятельности организации;
* разработать для пользователя простой и удобный интерфейс приложения;
* разработать и описать руководство пользователя.

Объект исследования – процесс работы музея.

Предмет исследования – принципы создания и функционирования автоматизированной системы работы музея.

|  |  |  |
| --- | --- | --- |
| **1** | **АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ СИСТЕМЫ РАБОТЫ МУЗЕЯ** | |
| **1.1** | | **Описание предметной области** |

Музей -учреждение, занимающееся собиранием, изучением, хранением и экспонированием предметов: памятников естественной истории, материальной и духовной культуры, а также просветительской и популяризаторской деятельностью. Музеи, являясь банками информации огромной ценности, призваны использовать ее в социокультурных и экономических целях, обеспечить самоокупаемость и предложить обществу новую стратегию развития культуры. Для этого требуется положить в основу перспективного плана принципы бизнеса, не разрушая традиционные функции музея.

В настоящее время знакомство с коллекциями и изучение их отдельных экспонатов приобретает все большую популярность среди различных социальных групп общества. Музей как социально-культурное учреждение завоевывает все более крепкие позиции в системе культурного просвещения посетителей. Сейчас музей становится объединяющим звеном для совершенно различных по своим направлениям культурных интересов людей.

Участие музеев в социально значимых общественных процессах требует применения в музейной практике новых современных информационных технологий, постоянного обновления мультимедийного оборудования и насыщения рынка музейных услуг, что позволило бы расширить аудиторию и увеличить число новых пользователей. При решении этой задачи актуализируется издательская деятельность, создаются и пополняются web-приложения, осуществляется компьютерная каталогизация фондов (книжных и предметных).

Прежде всего все это касается такой функции музея как экспозиция. Экспозиция — основная форма музейной коммуникации, образовательные и воспитательные цели которой осуществляются путём демонстрации музейных экспонатов, организованных, объяснённых и размещённых в соответствии с разработанной музеем научной концепцией и современными принципами архитектурно-художественных решений. Данный вид деятельности прежде всего нацелен на взаимодействия с посетителями музея.

На протяжении столетий общественная значимость музея определялась качеством его экспозиции, с 1970-х годов не менее важным показателем стал уровень проводимых им акций, мероприятий и программ. В последние годы все более существенным фактором становится обращенность музея в открытое информационное пространство. Пользователи желают получить всю необходимую информацию, относительно доступных экскурсий, времени их проведения, экскурсоводов, которые могут их провести. Выход организации в открытое информационное пространство может решить эти вопросы.

В связи с этим музеи прибегают к созданию собственных web-приложений. Данное web-приложение позволяет дать пользователю доступ ко всей интересующей его информации в удобном формате. Пользователи могут эффективно взаимодействовать с системой музея, выбирая необходимые категории экскурсий, конкретную экскурсию, экскурсоводов и время с датой, которые больше всего им подходит.

Так, стоит выделить несколько существенных моментов:

– Веб-приложение музея в некоторых случаях может быть единственной возможностью получить исчерпывающую информацию о самом музее и его коллекциях потенциальному посетителю, который в реальности не сможет посетить данный музей по каким-либо объективным причинам (к примеру, посетитель с ограниченными возможностями или сам музей находится в недоступном для посетителя месте).

– Наполнение музейного веб-приложения может предоставить исчерпывающую информацию о коллекции музея, о выставках и образовательных мероприятиях.

Но помимо весомой пользы посетителям музея, веб-приложения музея позволяет значительно упростить организационную работу. Так руководитель музея может достаточно просто добавлять новые категории экскурсий и их конкретные виды. Музеи получаю достаточно данных для определения своей дальнейшей стратегии. Экскурсоводы в свою очередь могут следить за своей работой и вносить необходимые изменения в зависимости от существующего у них статуса в данной системе.

Помимо этих основных функций, многие музеи используют web-приложение для размещения научных статей и других материалов чтобы уменьшить затраты на публикации. Также, музейное веб-приложение даёт возможность открыть доступ к новым источникам финансирования посредствам электронной коммерции.

Появляется новая возможность для взаимодействия музея (сотрудников музея) и его посетителей. В данном случае музейный веб-приложение, по сути, играет роль интерфейса, посредством которого происходит взаимодействие посетителя и аудитории.

|  |  |
| --- | --- |
| **1.2** | Функциональное моделирование процесса онлайн заказа еды фастфуд-ресторана на основе стандарта IDEF0 |

Проведённый анализ предметной области даёт возможность разработать функциональную модель процесса оказания платной услуги пациенту в поликлинике на основе методологии IDEF0.

IDEF (I-CAM DEFinition или Integrated DEFinition) — методологии семейства ICAM (Integrated Computer-Aided Manufacturing) для решения задач моделирования сложных систем, позволяют отображать и анализировать модели деятельности широкого спектра сложных систем в различных разрезах. При этом широта и глубина обследования процессов в системе определяется самим разработчиком, что позволяет не перегружать создаваемую модель излишними данными.

На рисунке 1.1 представлена контекстная диаграмма верхнего уровня. На входе в диаграмму представлены аккаунт пользователя, каталог экскурсий (из которых можно выбрать) и список экскурсоводов, которые могут провести экскурсию. Эти входные данные становятся предстоящей экскурсией. Выполняется этот процесс в соответствии с уставом музеем и законом о защите прав потребителей. Необходимыми для этого ресурсами же являются клиент музея и экскурсовод.

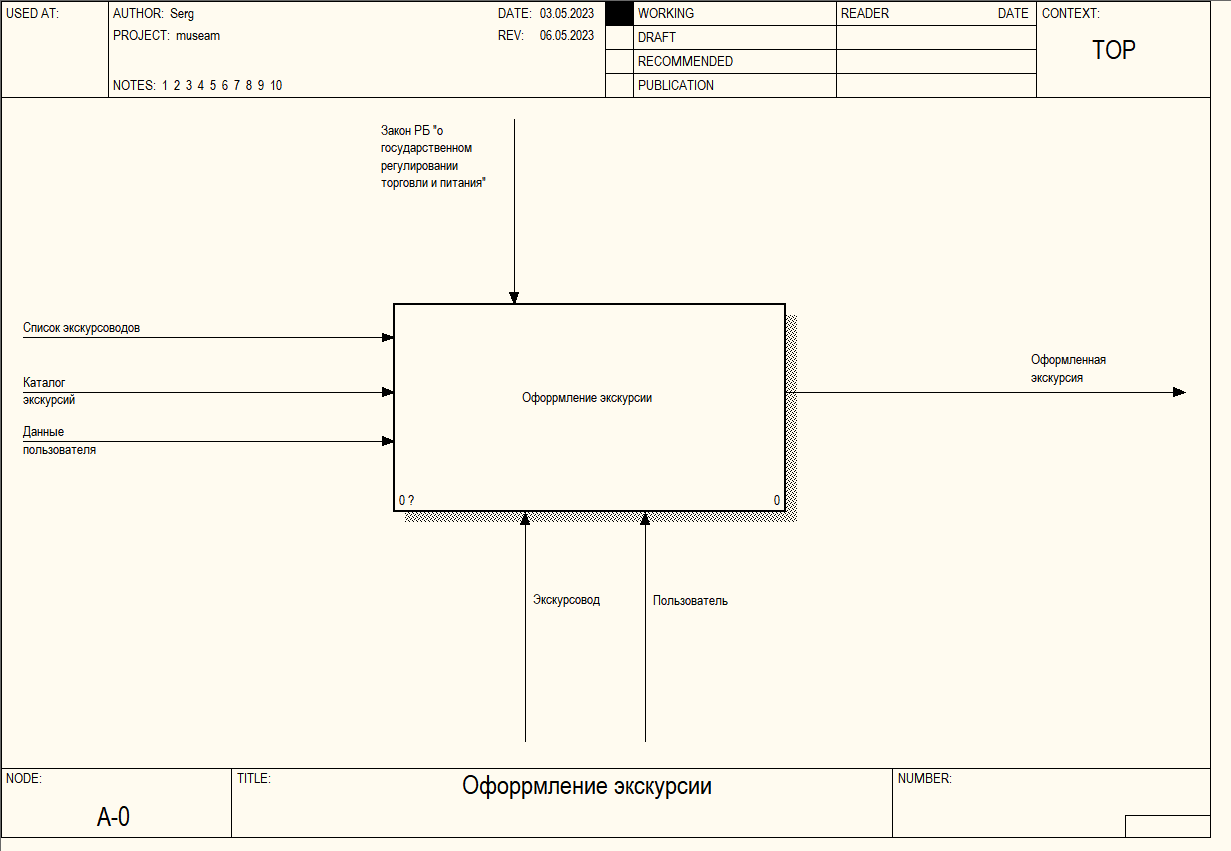


Рисунок 1.1 – Контекстная диаграмма верхнего уровня

Декомпозиция контекстной диаграммы, представленная на рисунке 1.2, описывает разделение основного процесса на подпроцессы. Этот процесс разделяется на 4 этапа:

* выбрать экскурсию;
* выбрать время и экскурсовода;
* подтвердить экскурсию;
* подготовиться к экскурсии;

После того, как экскурсовод получает заявку от клиента на экскурсию, он может ее одобрить и тем самым ее подтвердить. В дальнейшем он должен к ней подготовиться.

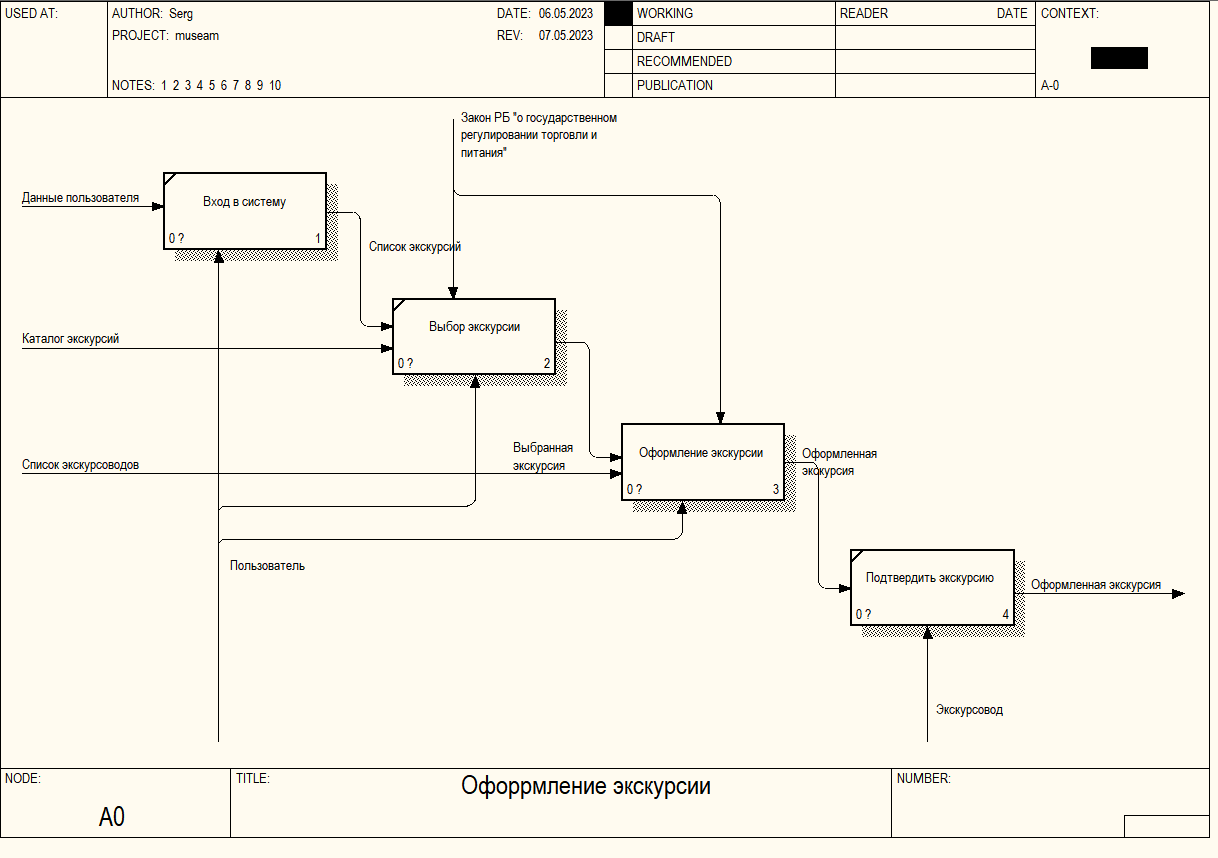


Рисунок 1.2 – Декомпозиция контекстной диаграммы верхнего уровня

В дальнейшем на рисунке 1.3 представлена диаграмма декомпозиции блока «Подтвердить экскурсию». Этот блок состоит из этапов:

- проверить лист с экскурсиями

-проверить расписание и подтвердить экскурсию;

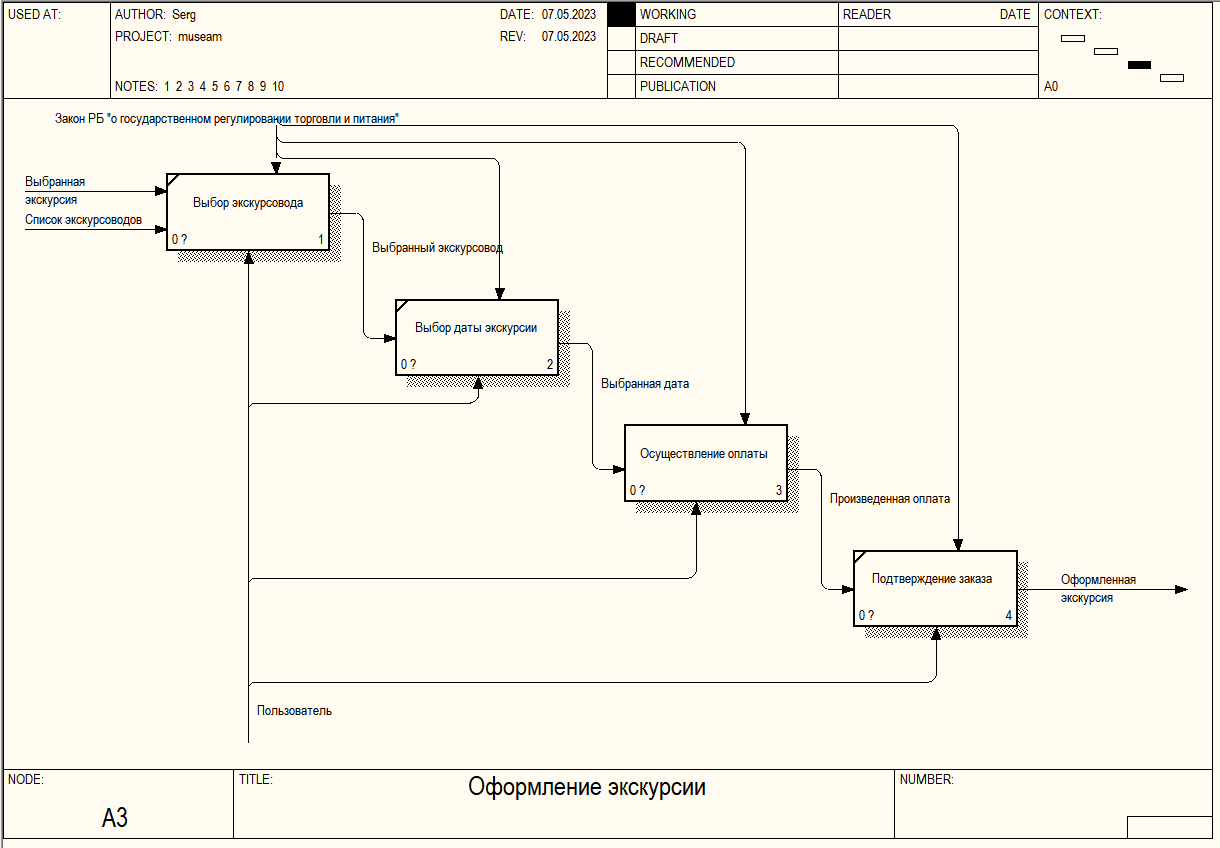


Рисунок 1.3 –Диаграмма декомпозиции блока «Подтвердить экскурсию»

В дальнейшем, для описания процесса, необходимо использовать IDEF3. DEF3 — это метод моделирования бизнес-процессов, дополняющий IDEF0. Метод IDEF3 — это управляемый сценарием метод сбора описания потока процессов, предназначенный для сбора знаний о том, как работает конкретная система.

С помощью данного метода можно рассмотреть подробнее процесс осуществление оплаты. Соответствующая декомпозиция представлена на рисунке 1.4.

Как видно из представленной ниже диаграммы, первым этапом является ввод данных банковской карты. В дальнейшем, пользователю предоставлен выбор между оплатой сразу и отложенным платежом, который будет взыматься после прошествия заказанной экскурсии. В конце пользователь подтверждает оплату выбранной экскурсии.

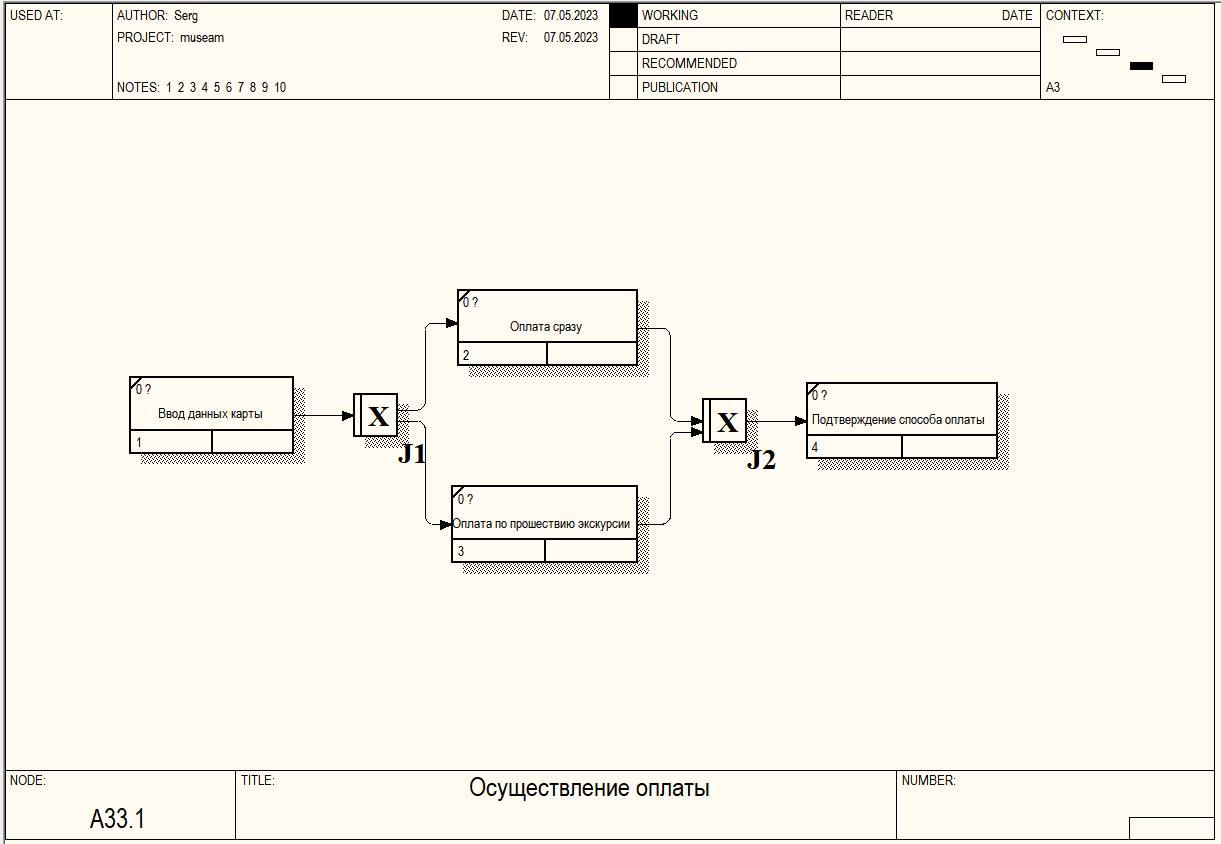


Рисунок 1.4 –Диаграмма декомпозиции блока «Осуществление оплаты»

|  |  |
| --- | --- |
| **1.3** | **Информационная модель системы работы музея** |

Информационная система – это коммуникационная и вычислительная система по сбору, хранению, обработке и передаче информации, снаб­жающая работников различного ранга той информацией, которая необходима им для реализации функций управления.

Так же информационная модель системы отражает информацию о предметной области, так называемой части реального мира, данные о которой должны храниться в проектируемой базе данных. В данном случае предметной областью является музей. При этом информация, необходимая для описания информационной системы, зависит от типа самой инфраструктуры.

В области разработки программного обеспечения и [моделирования данных](https://translated.turbopages.org/proxy_u/en-ru.ru.08902d62-64595708-8a4a6aca-74722d776562/https/en.wikipedia.org/wiki/Data_modeling) информационная модель обычно представляет собой абстрактное, формальное представление типов сущностей, которое может включать их свойства, взаимосвязи и операции, которые могут быть выполнены над ними. Типами объектов в модели могут быть объекты реального мира, такие как устройства в сети, или события, или они сами могут быть абстрактными, например, для объектов, используемых в биллинговой системе. Как правило, они используются для моделирования ограниченной области, которая может быть описана замкнутым набором типов сущностей, свойств, взаимосвязей и операций.

Для хранения и работы с данными разрабатываемого приложения была база данных PostgresSQL. Схема данных приведена на рисунке 1.5.

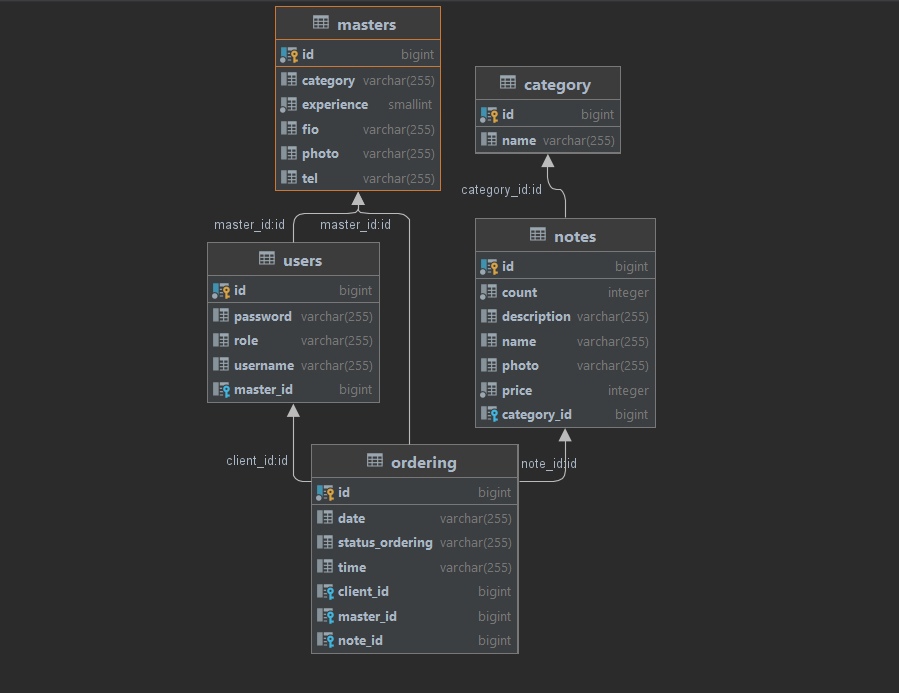
****

Рисунок 1.5 – Информационная модель базы данных

В данной модели представлены следующие сущности:

* Заказ (ordering);
* Записи (notes);
* Пользователи (users);
* Экскурсоводы (masters);
* Категории (category);

Наиболее важной сущностью является заказ(ordering). Соответствующая таблица содержит следующие элементы:

* Id;
* Date (дата экскурсии);
* Status ordering (статус заказа);
* Time (время соответствующей экскурсии);
* Client\_id (id клиента);
* Master\_id (id экскурсовода);
* Note\_id (id записи);

|  |  |
| --- | --- |
| **1.4** | **Модели представлений автоматизированной системы работы музея** |

В дальнейшем представлена диаграмма вариантов использования. Она представлена на рисунке 1.6. Она предназначены для упрощения взаимодействия с будущими пользователями системы, с клиентами, и особенно пригодятся для определения необходимых характеристик системы. Другими словами, диаграммы вариантов использования говорят о том, что система должна делать, не указывая сами применяемые методы.

Вариант использования описывает, с точки зрения действующего лица, группу действий в системе, которые приводят к конкретному результату. Варианты использования являются описаниями типичных взаимодействий между пользователями системы и самой системой. Они отображают внешний интерфейс системы и указывают форму того, что система должна сделать.

В качестве актеров здесь выступают клиент (то есть обычный пользователь), администратор и экскурсовод.

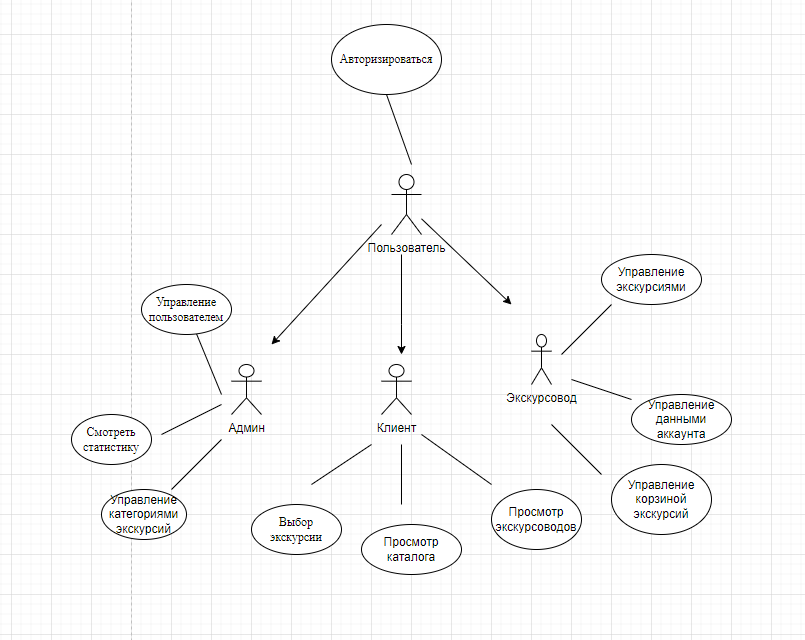


Рисунок 1.6 –Диаграмма вариантов использования

Каждому пользователю, в независимости от роли, может авторизироваться. Клиенту доступно взаимодействие с каталогом доступных экскурсий. Также он может выбрать экскурсию из предложенных категорий и представленных в этих категориях вариантов. После выбора он может выбрать подходящие критерии экскурсии: экскурсоводов, время и дату. Также клиенту доступна раздел с информацией об экскурсоводах.

Администраторы в свою очередь имеют самый важный функционал. Он может просматривать статистику, управлять категориями экскурсий, а именно добавлять категории и редактировать их. Помимо этого, он ответственен за управлением пользователями. Он может давать им определенные категории.

Экскурсоводы, в свою очередь, редактируют информации о себе в соответствующей категории. Также он добавляет, удаляет и редактирует экскурсиями в конкретных категориях. После того, как пользователь выбрал экскурсию, экскурсовод утверждает или отменяет экскурсию.

Далее приведены диаграммы классов. Диаграмма классов — структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов, методов, интерфейсов и взаимосвязей между ними.

На рисунке 1.7 представлена диаграмма классов пакета controller. Он представляет собой верхний слой архитектуры. Controller получает ввод пользователя, обрабатывает его и посылает обратно результат обработки.

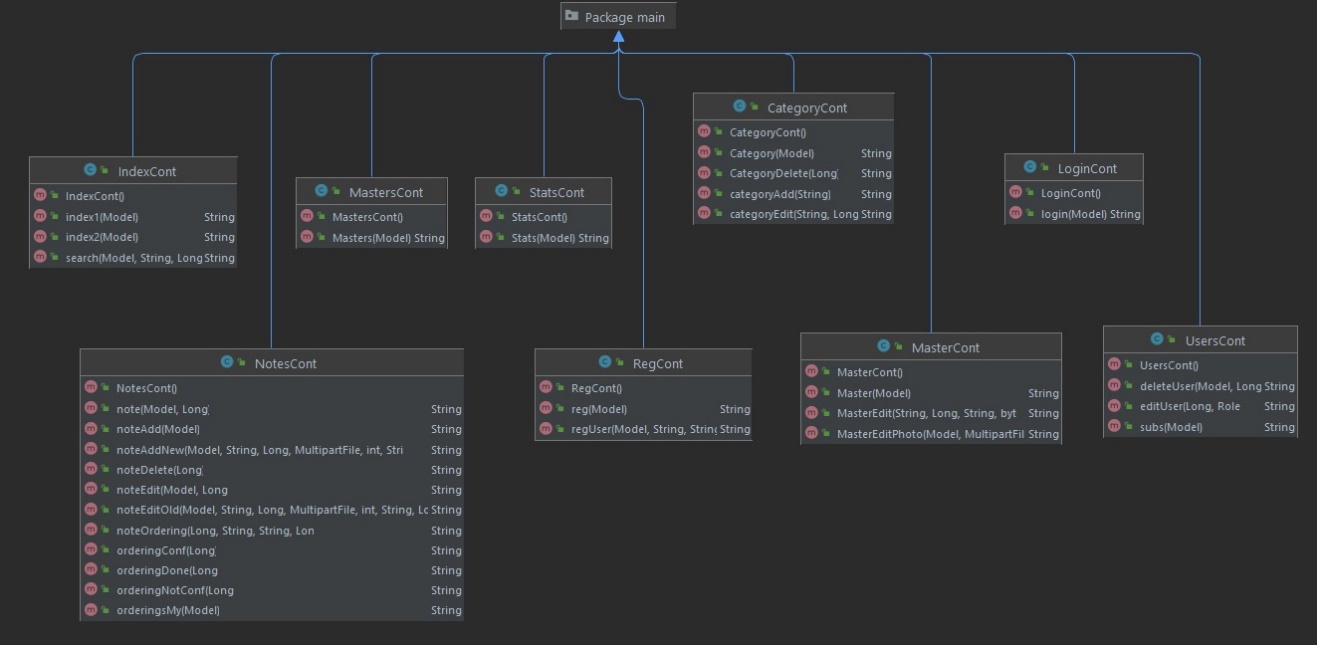


Рисунок 1.7 –Диаграмма классов пакета controller

На рисунке 1.8 представлена диаграмма класса пакета repo. Пакет repo содержит классы, инкапсулирующие логику работы с сущностями базы данных.

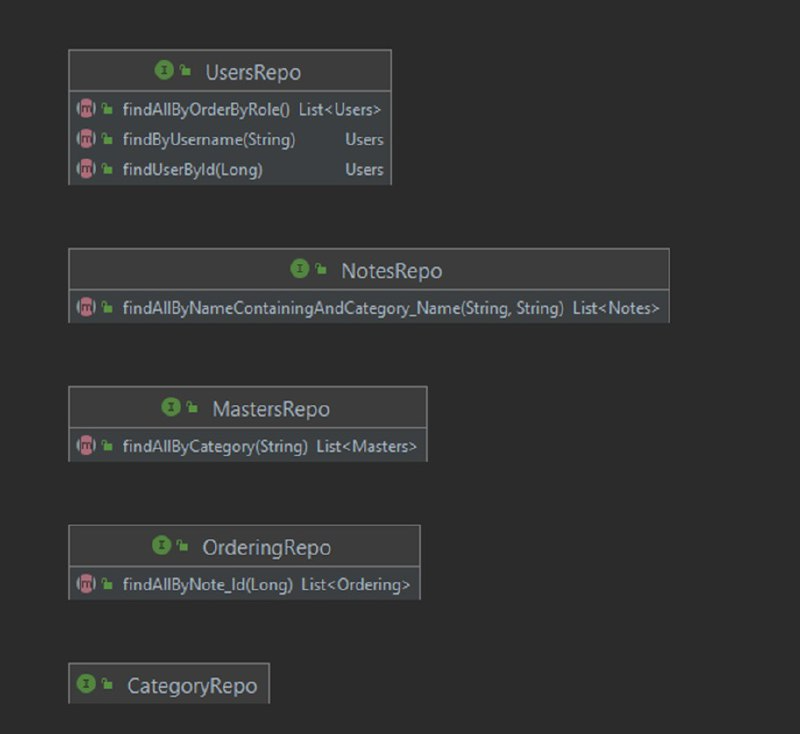


Рисунок 1.8–Диаграмма классов пакета repo

На рисунке 1.9 диаграмма пакета model. представляют собой базу данных и взаимодействие с системой управления базы данных.

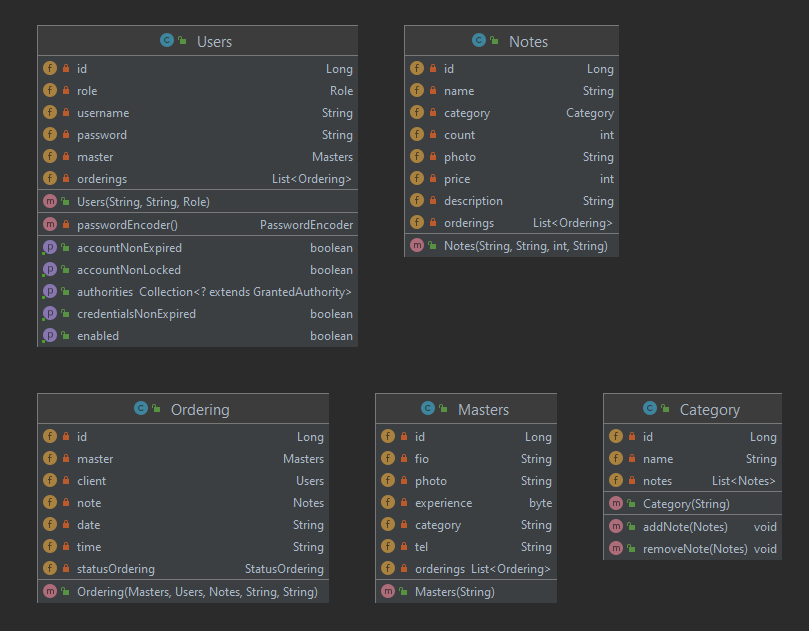


Рисунок 1.9 –Диаграмма классов пакета model

В дальнейшем необходимо представить диаграмму развертывания. Диаграмма развертывания – это тип UML-диаграммы, которая показывает архитектуру исполнения системы, включая такие узлы, как аппаратные или программные среды исполнения, а также промежуточное программное обеспечение, соединяющее их. Диаграммы развертывания используются для визуализации аппаратных процессоров/узлов/устройств системы, каналов связи между ними и размещения программных файлов на этом аппаратном обеспечении.

Представлена данная диаграмма на рисунке 1.10 Сервер использует сборщик проектов Maven, Spring Boot необходим для запуска и управления.



Рисунок 1.10 –Диаграмма развертывания

Диаграммы компонентов используются для визуализации организации компонентов системы и зависимостей между ними. Они позволяют получить высокоуровневое представление о компонентах системы. Представлена данная диаграмма на рисунке 1.11.

Компонент Database включает в себя базу данных, соединение с сервером происходит посредством использования JDBC драйвера. Сервер имеет Database control classes необходимый для подключения и работы с базой данных, control classes служит в качестве соединяющего звена реализации бизнес-логики. Сервер соединен с клиентом, передача данных происходит с помощью TCP/IP. На клиенте организована структура пользовательского интерфейса на основе HTML/CSS.

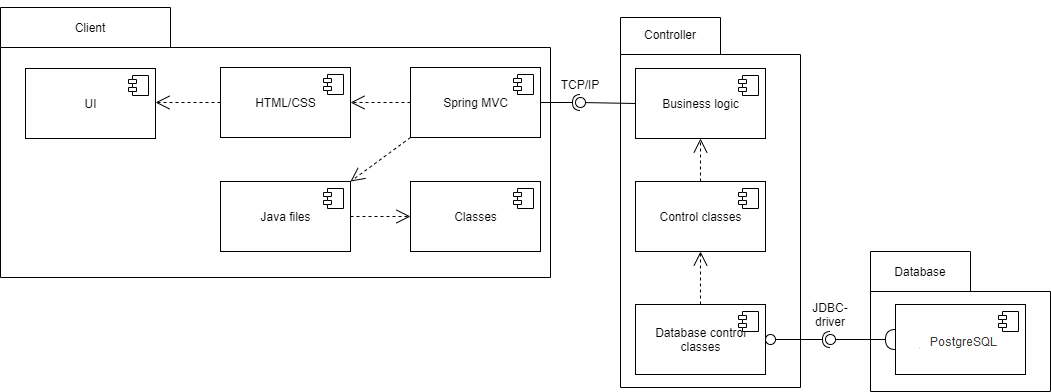


Рисунок 1.11 –Диаграмма компонентов

Диаграмма состояний – это тип диаграммы, используемый для описания поведения систем. Диаграммы состояний требуют, чтобы описываемая система состояла из конечного числа [состояний](https://translated.turbopages.org/proxy_u/en-ru.ru.497f5846-645a3a58-5c52407a-74722d776562/https/en.wikipedia.org/wiki/State_(computer_science)); иногда это действительно так, в то время как в других случаях это разумная [абстракция](https://translated.turbopages.org/proxy_u/en-ru.ru.497f5846-645a3a58-5c52407a-74722d776562/https/en.wikipedia.org/wiki/Abstraction_(computer_science)). Существует множество форм диаграмм состояний, которые немного отличаются и имеют разную [семантику](https://translated.turbopages.org/proxy_u/en-ru.ru.497f5846-645a3a58-5c52407a-74722d776562/https/en.wikipedia.org/wiki/Semantics#Computer_science). Диаграммы состояний используются для того, чтобы дать абстрактное описание [поведения](https://translated.turbopages.org/proxy_u/en-ru.ru.497f5846-645a3a58-5c52407a-74722d776562/https/en.wikipedia.org/wiki/Behavior) [системы](https://translated.turbopages.org/proxy_u/en-ru.ru.497f5846-645a3a58-5c52407a-74722d776562/https/en.wikipedia.org/wiki/System). Это поведение анализируется и представляется серией событий, которые могут произойти в одном или нескольких возможных состояниях. Таким образом, "каждая диаграмма обычно представляет объекты одного класса и отслеживает различные состояния своих объектов в системе". Среди основных компонентов данной диаграммы можно выделить состояние и переход. Состояние– это ситуация в жизни объекта, на протяжении которой он удовлетворяет некоторому условию, осуществляет определенную деятельность или ожидает какого–то события. Переход– это отношение между двумя состояниями, показывающее, что объект, находящийся в первом состоянии, должен выполнить некоторые действия и перейти во второе состояние, как только произойдет определенное событие и будут выполнены заданные условия. Данная диаграмма представлена на рисунке 1.12.



Рисунок 1.12 – Диаграмма состояний

В дальнейшем необходимо построить диаграмму последовательности. Эта диаграмма является видом диаграмм взаимодействия, которые описывают отношения объектов в различных условиях. Диаграммы последовательностей обычно содержат объекты, которые взаимодействуют в рамках сценария, сообщения, которыми они обмениваются, и возвращаемые результаты, связанные с сообщениями. Диаграмма последовательности процесса поиска необходимых экскурсий представлена на рисунке 1.13.

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники), вертикальные линии (линии жизни), отображающие течение времени при деятельности объекта, и стрелки, показывающие выполнение действий объектами. На рассматриваемой диаграмме объекты располагаются слева направо. В верхней части представлен сценарий, в ходе которого будет найдена экскурсия. В низу находится, исход при котором диаграмма не будет найдена.

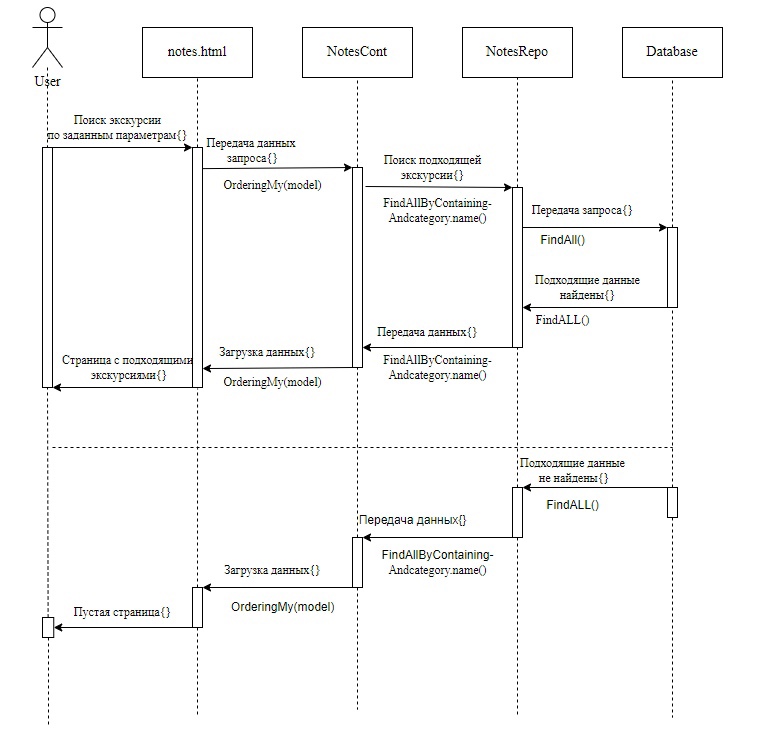


Рисунок 1.13 –Диаграмма последовательности

|  |  |  |
| --- | --- | --- |
| **2** | **ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА** | |
| **2.1** | | **Постановка задачи** |

Для достижения поставленной в курсовом проекте цели необходимо разработать автоматизированную систему работы музея. Автоматизированная система, в свою очередь, представляет собой организационно-техническую систему, обеспечивающую выработку решений на основе автоматизации информационных процессов в различных сферах деятельности (управление, проектирование, производство и тому подобное) или их сочетаниях.

Для этого необходимо разработать такое приложение, которое предоставит возможность многим пользователям вносить и просматривать необходимую информацию. Пользоваться им смогут различные сотрудники музея. Например, директор музея сможет добавлять экскурсии и давать категории отдельным пользователям. Экскурсоводы смогут просматривать свои экскурсии и подтверждать.

Для повышения рентабельности музея необходимо автоматизировать прежде всего процесс бронирования билетов на экскурсии. Осуществить это необходимо для каждого клиента, который пользуется данной услугой. Выбор экскурсий осуществляется с помощью специальных информационных технологий. Для корректного бронирования необходима предоплата и соответствующие личные данные пассажира. Предоплата будет зависеть от конкретной экскурсии и конкретного экскурсовода.

Данный процесс тесно связан с регистрацией пользователя. Следовательно, и данный процесс подлежит автоматизации. При бронировании билета пользователем, ему необходимо авторизоваться в системе. Это является процессом, который содержит конфиденциальную информацию пользователя. Поэтому очень важно осуществить автоматизацию данного этапа.

Это приложение должно производить обработку следующих запросов:

* авторизация пользователя;
* регистрация пользователя;
* получение списка пользователей;
* получение списка экскурсий;
* получение списка экскурсоводов;
* получение статистики о проведённых экскурсиях;
* добавление новых записей в базу данных;
* редактирование существующих записей;
* удаление записей из базы данных;
* получение отчёта о сумме дохода за определённый промежуток времени;
* формирование текстового отчёта.

Работников музея, в свою очередь, можно подразделить на простых пользователей, экскурсоводов, директора. Каждая из перечисленных категорий работников имеет уникальные атрибуты-характеристики, определяемые профессиональной направленностью.

|  |  |
| --- | --- |
| **2.2** | **Обоснование выбора компонентов и технологий для реализации системы онлайн заказа фастфуд-ресторана** |

Для написания back-части приложения был выбран язык Java. Java - это [высокоуровневый](https://translated.turbopages.org/proxy_u/en-ru.ru.7209af29-6458cdb9-72d6d111-74722d776562/https/en.wikipedia.org/wiki/High-level_programming_language), основанный на классах, объектно-ориентированный [язык](https://translated.turbopages.org/proxy_u/en-ru.ru.7209af29-6458cdb9-72d6d111-74722d776562/https/en.wikipedia.org/wiki/Programming_language)программирования[, который разработан таким образом, чтобы иметь как можно меньше](https://translated.turbopages.org/proxy_u/en-ru.ru.7209af29-6458cdb9-72d6d111-74722d776562/https/en.wikipedia.org/wiki/Dependency_(computer_science))зависимостей, связанных с реализацией. Это один из наиболее известных и популярных языков.

Среди основных особенностей, как упоминалось выше, языка можно выделить: мультифункциональность, строгая сильная типизация и объектно-ориентированность.

 Java не привязан ни к одной из популярных платформ. С одинаковым успехом его можно использовать на Windows или iOS, Linux или Android. Сильная типизация не позволяет смешивать в выражениях различные типы и не выполняет автоматически неявные преобразования.

Java создан по модели объектно-ориентированного программирования. Здесь есть классы и объекты. Классы — типы данных, а объекты — представители классов. Разработчик создает их сам, дает названия и присваивает им свойства и операции, которые можно выполнять с ними. Это напоминает конструктор, позволяющий построить то, что хочется.

К основным достоинствамязыка можно отнести:

* мультифункциональность;
* достаточно простой синтаксис;
* независимость (код сможет работать на любой платформе, поддерживающей Java);
* надежность (благодаря строгой статической типизации);
* возможность параллельной разработки;
* хорошую организацию кода и его легкую поддержку;
* универсальность использования различных классов.

В качестве среды разработки выбрана IntelliJ IDEA. Интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains. Первая версия появилась в январе 2001 года и быстро приобрела популярность как первая среда для Java с широким набором интегрированных инструментов для рефакторинга, которые позволяли программистам быстро реорганизовывать исходные тексты программ.

Преимущества:

* автодополнение кода и качественная отладка;
* удобная навигация;
* безопасный рефакторинг – применить изменения во всем проекте можно за пару кликов;
* функция Live Edit позволяет мгновенно посмотреть все изменения в браузере;
* интерфейс будет понятен даже новичкам.

Данные преимущества делают данную среду разработки оптимальной для создания в ней курсового проекта.

Проектирование информационной модели, хранящей данные о работе музея реализовано на PostgreSQL. PostgreSQL является одной из наиболее популярных систем управления базами данных.

Среди ее достоинств можно выделить:

* подерживает функцию управления параллелизмом нескольких версий.;
* полная совместимость с SQL и ACID.;
* поддерживает JSON и другие функции NoSQL.;
* ннадёжность и целостность данных (например, в ней сложно разрушить таблицы).;
* есть возможность сохранения произведенных процедур, за счет чего возможно расширение опций.;
* функции не ограничивают размер БД, даже если она исчисляется петабайтами.;

Оптимальность данной СУБД для построения сложных систем послужило основной причиной ее выбора.

|  |  |
| --- | --- |
| **2.3** | **Архитектурные решения** |

Архитектура программного обеспечения заключается в принятии фундаментальных структурных решений, которые после внедрения требуют дорогостоящих изменений. Выбор архитектуры программного обеспечения включает в себя конкретные структурные параметры из возможностей проектирования программного обеспечения

При проектировании системы был использован конструкционный шаблон MVC (Model-View-Controller). Этот шаблон разделяет работу приложения на три отдельные функциональные роли: модель данных (Model), пользовательский интерфейс (View) и управляющую логику (Controller). Таким образом, изменения, вносимые в один из компонентов, оказывают минимально возможное воздействие на другие компоненты. В данном паттерне модель не зависит от представления или управляющей логики, что делает возможным проектирование модели как независимого компонента.

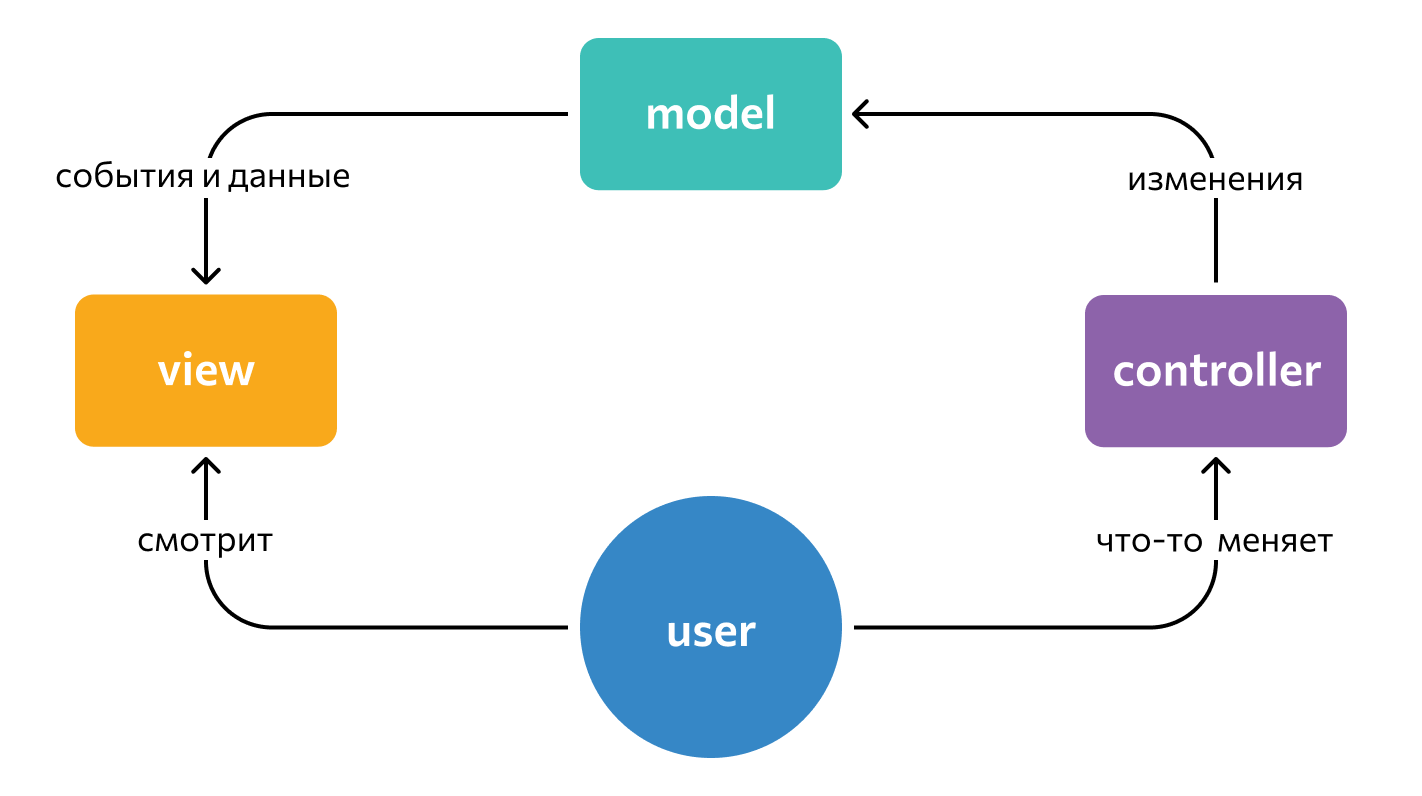


Рисунок 2.1 –Схема шаблона MVS

Первая часть содержит всю бизнес-логику приложения. Эта часть называется Модель (Model). В ней содержится код, который делает все то, для чего приложение создавалось. В данном проекте такой моделью является класс DataObject.

Вторая часть содержит все, что касается отображения данных пользователю. Она называется Вид (View). В ней содержится код, который управляет показом окон и сообщений. В данном проекте эту часть реализовывают FXML-файлы, в которых описаны все визуальные элементы приложения.

Третья часть содержит код, который занимается обработкой действий пользователя. Любые действия пользователя, направленные на изменения модели, должны обрабатываться в этой части. Такая часть называется Controller. Для каждого FXML-файла в данном проекте реализован свой контроллер.

Spring Boot – это популярный фреймворк для создания веб-приложений с использованием Java. Это часть фреймворка Spring, которая представляет собой набор инструментов и библиотек для создания приложений корпоративного уровня. Spring Boot упрощает создание автономных приложений производственного уровня, которые можно легко развернуть и запустить с минимальной конфигурацией. Данный фреймворк позволяет автоматически настраивать ваше приложение на основе зависимостей, которые вы добавили в свой проект. Ещё одним преимуществом Spring Boot является его поддержка широкого спектра технологий веб-приложений.

REST – это архитектурный стиль взаимодействия компонентов распределенной системы в компьютерной сети. Проще говоря, REST определяет стиль взаимодействия (обмена данными) между разными компонентами системы, каждая из которых может физически располагаться в разных местах. В общем случае этот происходит посредством запросов-ответов. Компоненту, которая отправляет запрос называют **клиентом**; компоненту, которая обрабатывает запрос и отправляет клиенту ответ, называют **сервером**. Запросы и ответы, чаще всего, отправляются по протоколу HTTP. Как правило сервер — это некое веб-приложение. Клиентом же может довольно многое. Например, мобильное приложение, которое запрашивает у сервера данные. Либо браузер, который отправляет запросы с веб-страницы на сервер для загрузки данных.

В целом, из основных преимуществ данного архитектурного стиля можно выделить:

- производительность (за счёт использования кэша);

- масштабируемость;

- прозрачность системы взаимодействия;

- простота интерфейсов;

- портативность компонентов;

- лёгкость внесения изменений;

- способность эволюционировать, приспосабливаясь к новым требованиям.

Для взаимодействия с базами данных, в свою очередь, был использован стандарт JDBC. Java Database Connectivity (JDBC) — это стандарт Java, предоставляющий интерфейс для подключения Java к реляционным базам данных. JDBC в своей основе имеет концепцию драйверов. Driver позволяет получать соединение с БД. Для реализации поставленной задачи задействуют специальные URL-адреса.

Основными преимуществами считаются:

* Лёгкость разработки: разработчик может не знать специфики базы данных, с которой работает;
* Код практически не меняется, если переходить на другую базу данных;
* Не нужно устанавливать громоздкую клиентскую программу;
* К любой базе можно подсоединиться через легко описываемый URL;

|  |  |
| --- | --- |
| **2.4** | Описание алгоритмов, реализующих ключевую бизнес-логику системы |

Бизнес-логика — совокупность правил, принципов, зависимостей поведения объектов предметной области (области человеческой деятельности, которую система поддерживает). Иначе можно сказать, что бизнес-логика — это реализация правил и ограничений автоматизируемых операций. Алгоритм бизнес-логики представляет собой последовательность определенных шагов, которые приводят к желаемому результату. На рисунке 2. Представлено графическое представление основного бизнес-процесса в виде блок схемы.

Первым шагом, пользователь вводит необходимые для поиска нужной экскурсии данные. Затем происходит процесс поиска соответствующей экскурсии. Если подходящие экскурсия найдена, то пользователь может перейти к ее оформлению. Он выбирает подходящее время для данной экскурсии. Затем идет отправка соответствующих данных на сервер. После пользователь приступает к процессу оплаты. Он вводит необходимые личные данные. После этого, у пользователя есть выбор, когда списать сумму. Он может оплатить сразу или выбрать автоматический платеж по прошествию экскурсии. Затем пользователь подтверждает оплату.

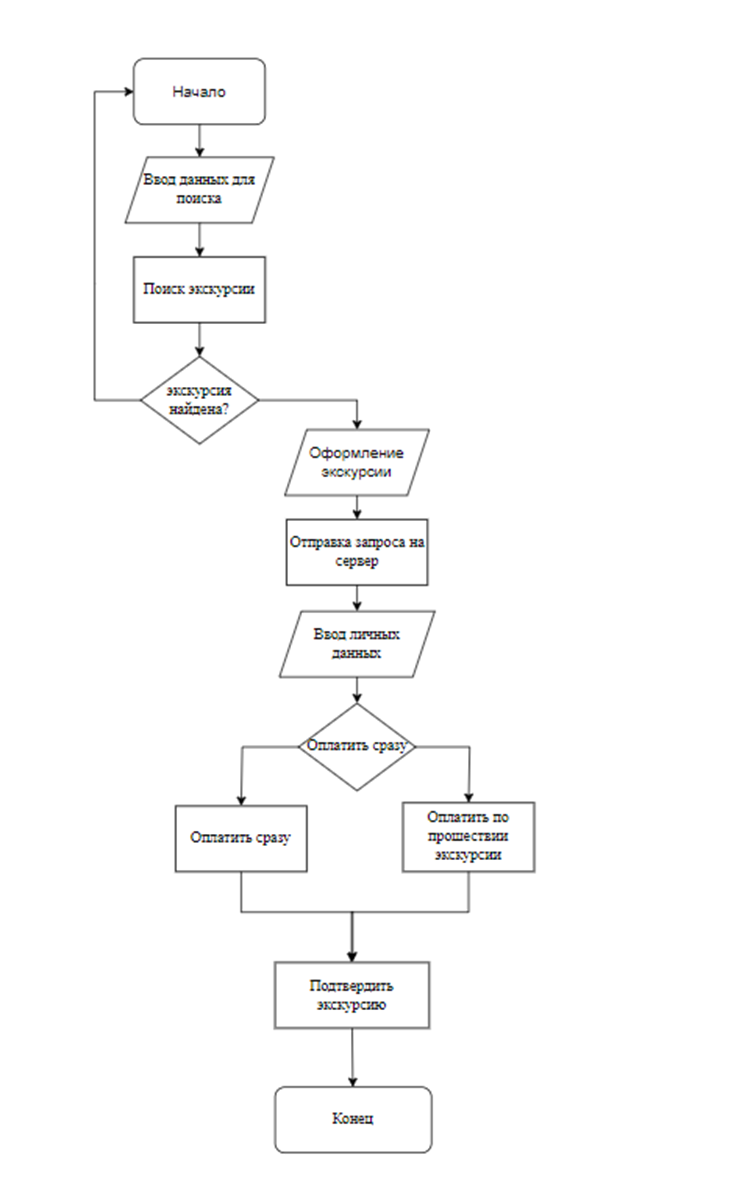


Рисунок 2.2 –Диаграмма состояний

|  |  |
| --- | --- |
| **2.5** | Проектирование пользовательского интерфейса |

Качество пользователь­ского интерфейса является самостоятельной характеристикой программного про­дукта, сопоставимой по значимости с такими его показателями, как надежность и эффективность использования вычислительных ресурсов. Среди основных требований к интерфейсу можно выделить: естественность, простота и эстетическая привлекательность.

Естественный интерфейс — такой, который не вынуждает пользователя суще­ственно изменять привычные для него способы решения задачи. Это, в частности, означает, что сообщения и результаты, выдаваемые приложением, не должны тре­бовать дополнительных пояснений.

Интерфейс должен быть простым. Это означает не упрощенчество, а обеспечение легкости в его изучении и в использовании. Кроме того, он должен предоставлять доступ ко всему перечню функциональных возможностей, предус­мотренных данным приложением. Реализация доступа к широким функциональ­ным возможностям и обеспечение простоты работы противоречат друг другу. Раз­работка эффективного интерфейса призвана сбалансировать эти цели.

Корректное визуальное представление используемых объектов, в свою очередь, обеспечивает передачу весьма важной дополнительных сведений о поведении и взаимодействии различных объектов. В то же время следует помнить, что каждый визуальный элемент, который появляется на экране, потенциально требует внимания пользователя, которое, как известно, не безгранич­но.

На рисунке 2.3 Представлена главная страница разрабатываемого веб-приложения. Естественность интерфейса представлена в виде довольно распространенного меню в “шапке” приложения. В нем должны располагаться основные разделы, доступные пользователю. В этом же воплощена и простота интерфейса. Пользователю доступны все необходимые разделы. В визуальном стиле, в качестве цвета “шапки” сайта, представлен голубой цвет, как достаточно сдержанный и холодный цвет, намекающий на серьезный подход к изучению, предоставляемого музеем материала.

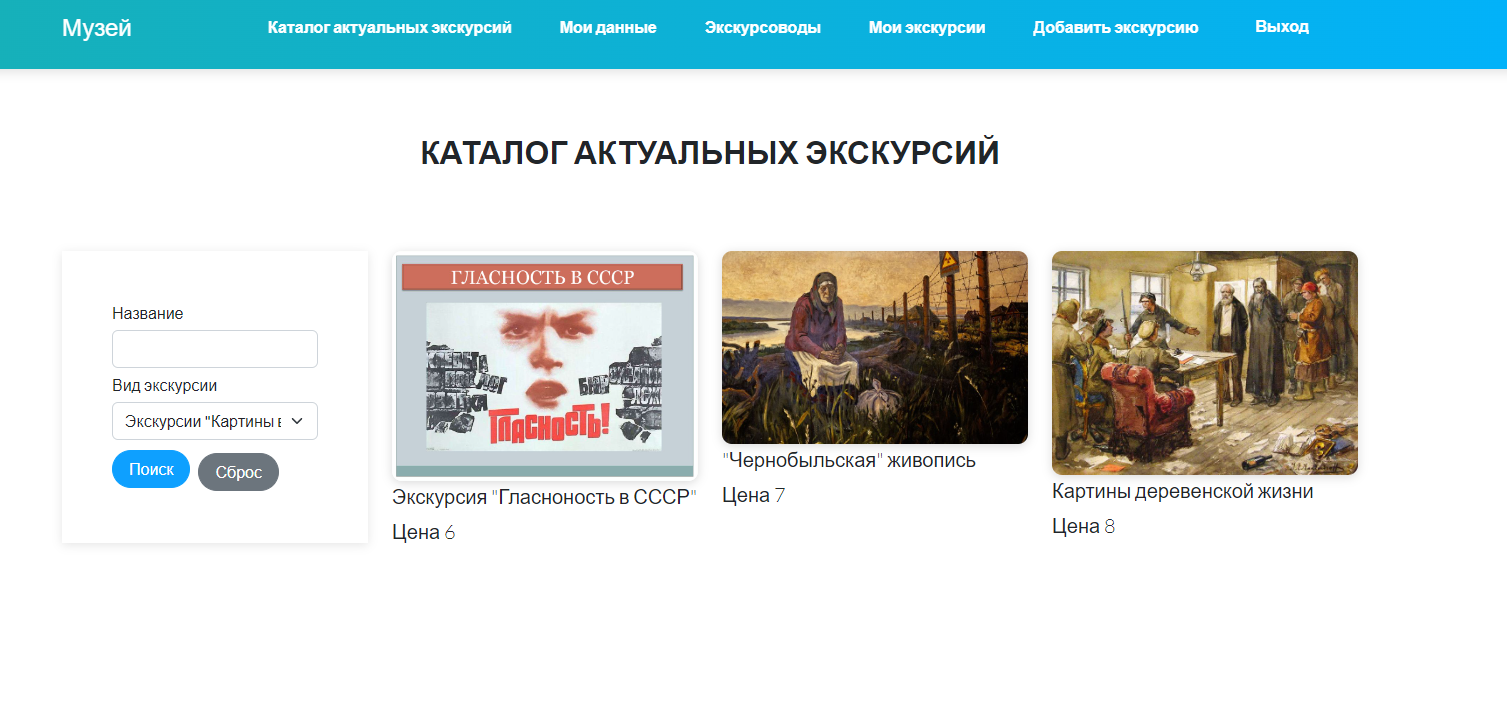


Рисунок 2.3 – главная страница приложения

|  |  |
| --- | --- |
| **3** | **РУКОВОДСТВО РАЗВЕРТЫВАНИЯ И ИСПОЛЬЗОВАНИЯ СИСТЕМЫ РАБОТЫ МУЗЕЯ** |

После запуска приложения перед пользователем появится страница Регистрации (Рисунок 3.1). Пользователь должен будет ввести свой логин и пароль, под которыми он в дальнейшем сможет заходить. После этого пользователь может зайти под этими данными в систему. Если же пользователь вводит неправильный логин или пароль он получит сообщение о некорректных введенных данных (Рисунок 3.2)

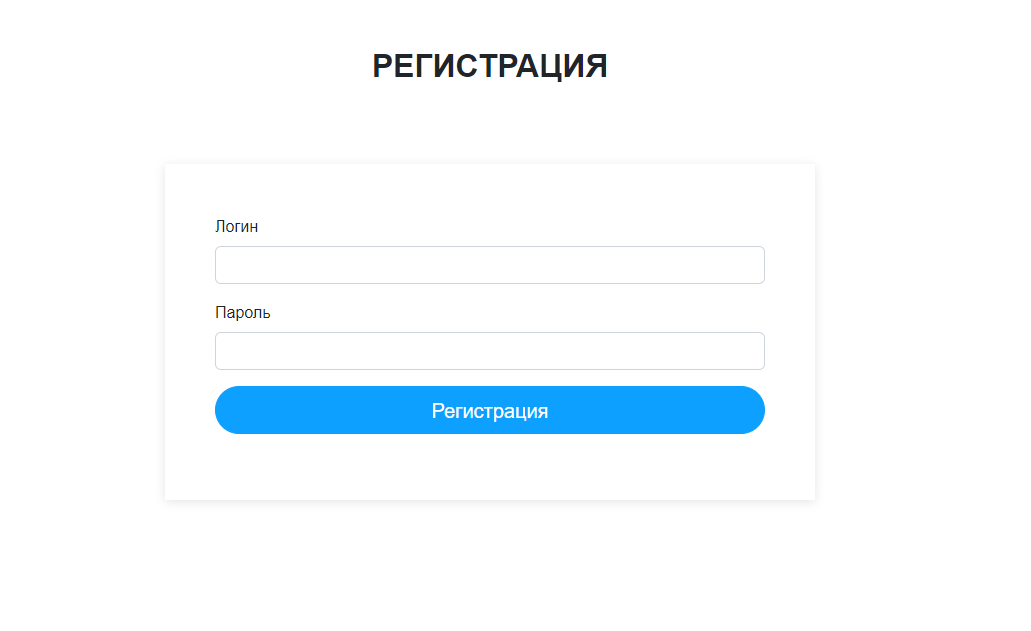


Рисунок 3.1 – страница регистрации

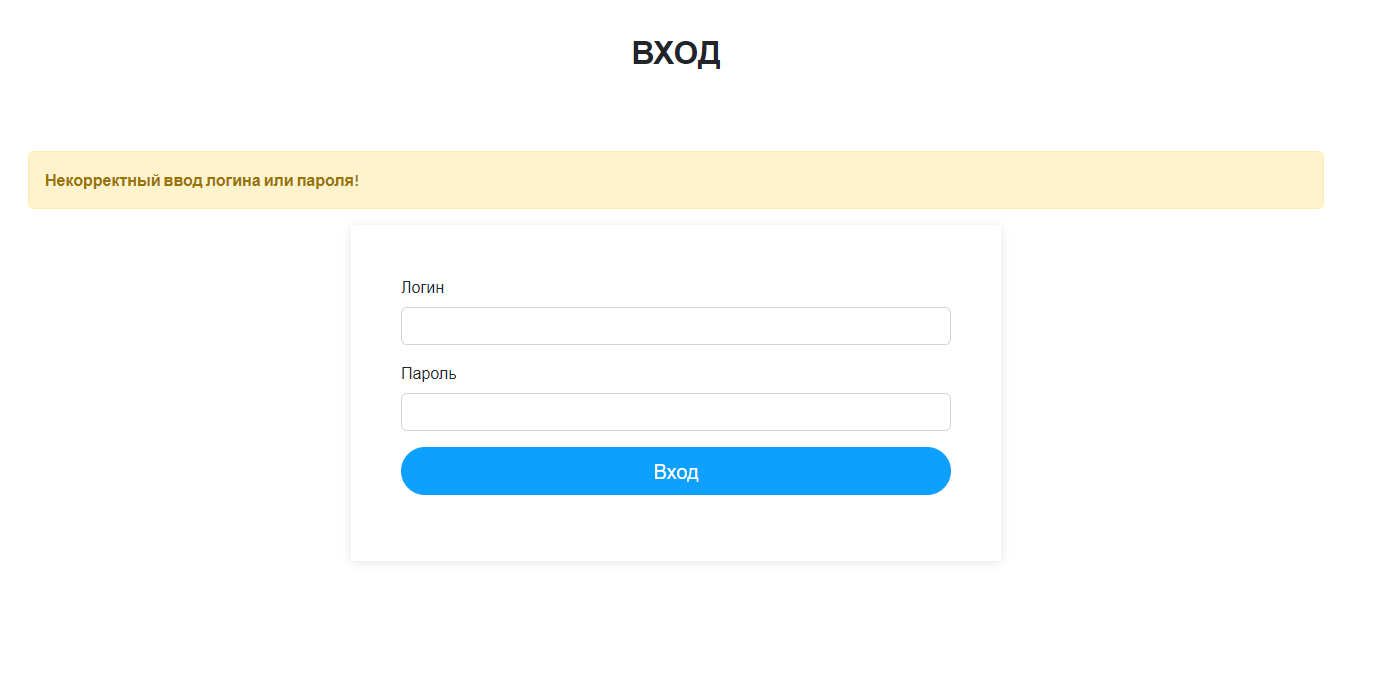


Рисунок 3.2 – Сообщение о некорректных данных

После входа в систему пользователь видит главную страницу (Рисунок 3.3). На ней представлен каталог актуальных экскурсий. В левой половине страницы располагается окно поиска. Пользователь может задать вид экскурсии или задать ее конкретное название. После нажатия кнопки «поиск» пользователю будет представлен список экскурсии искомой категории или конкретная искомая экскурсия. Все экскурсии сопровождены названиями и ценами. В дальнейшем пользователь сможет выбрать нужную ему экскурсию.

После этого пользователь оказывается на странице оформления заказа (Рисунок 3.4). Он должен выбрать доступного для данной экскурсии экскурсовода и время.

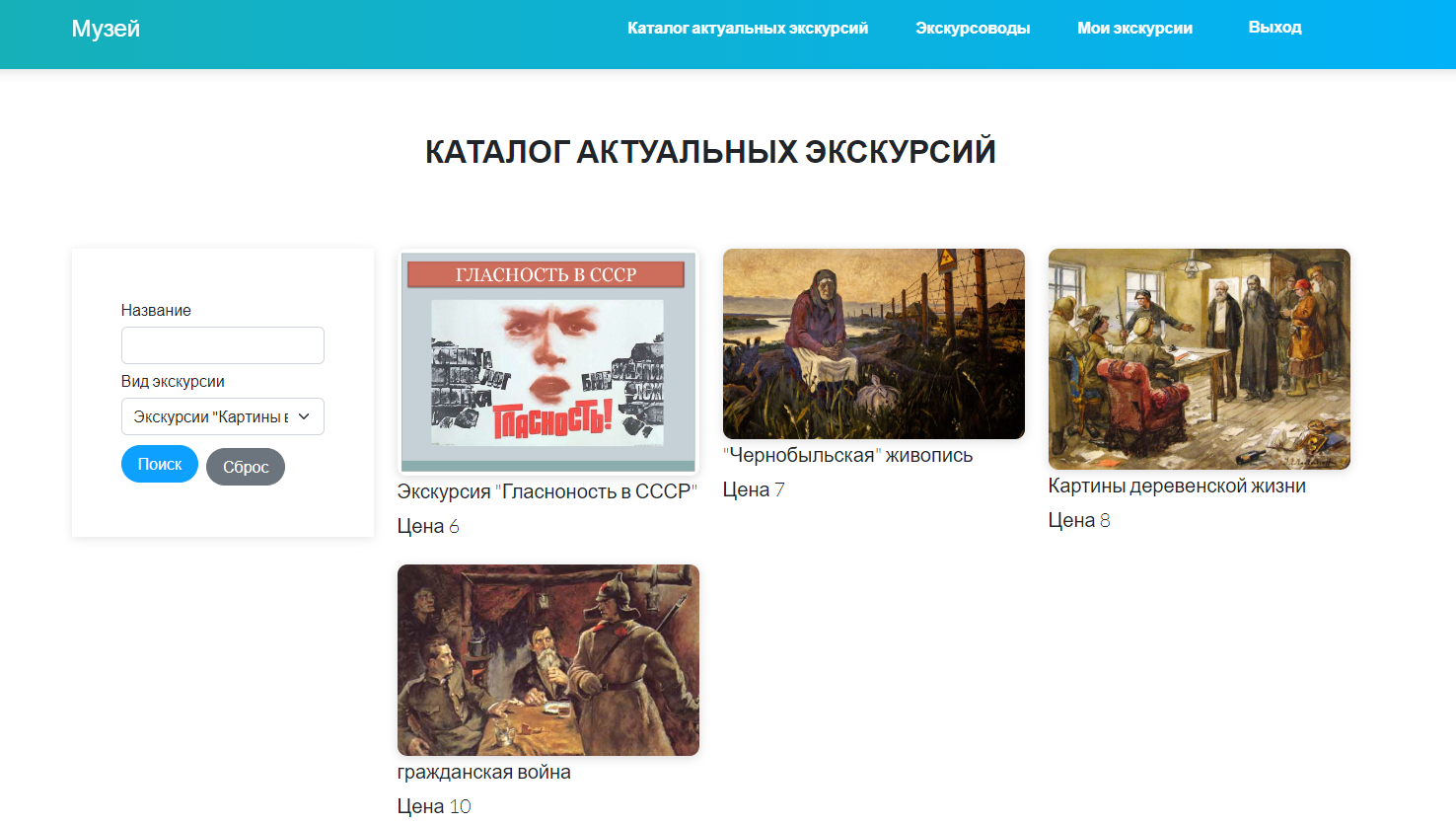


Рисунок 3.3 – Главная страница

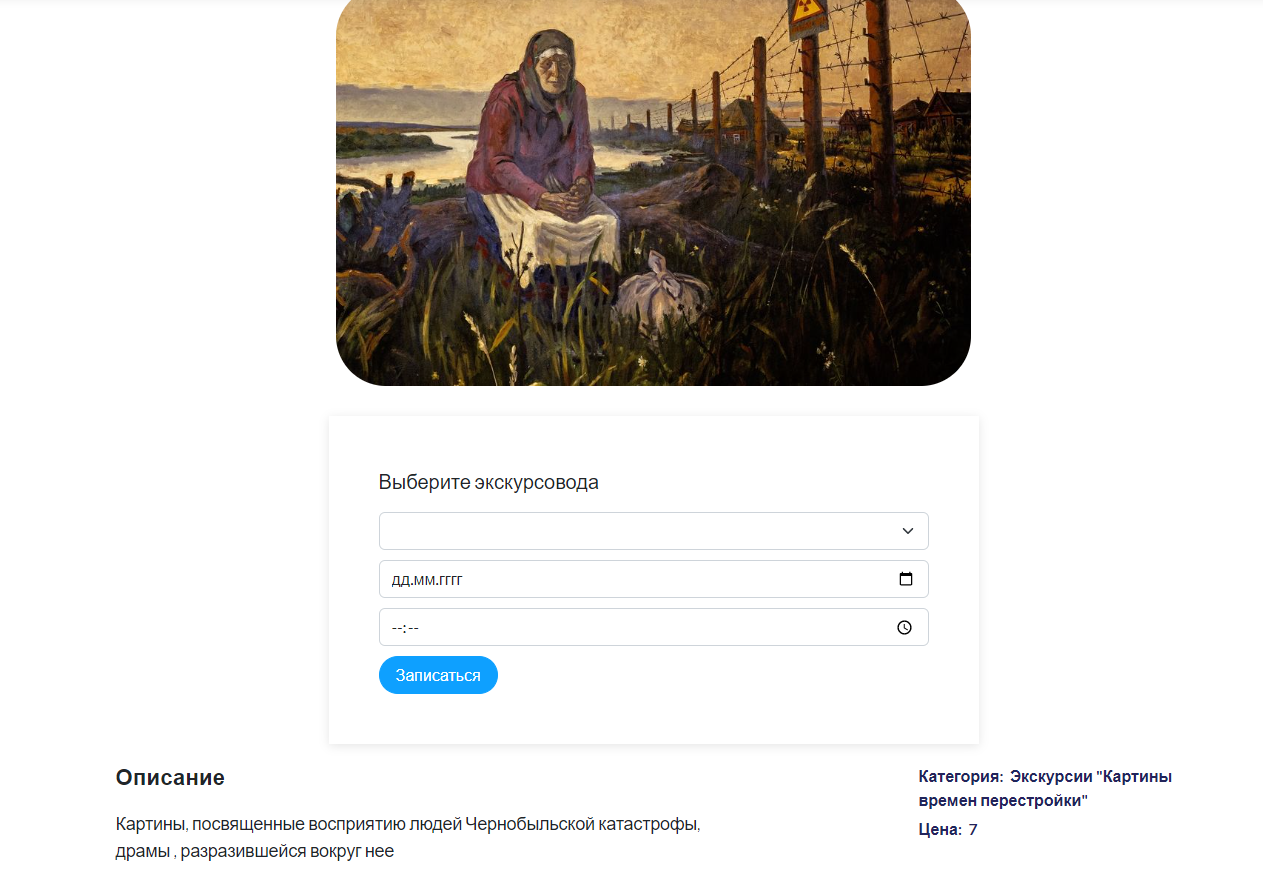


Рисунок 3.4 – Оформление экскурсии

После окончания процесса, пользователь может просмотреть список в добавленных экскурсии в разделе «Мои экскурсии» (Рисунок 3.5). Здесь представлена вся базовая информация и статус.

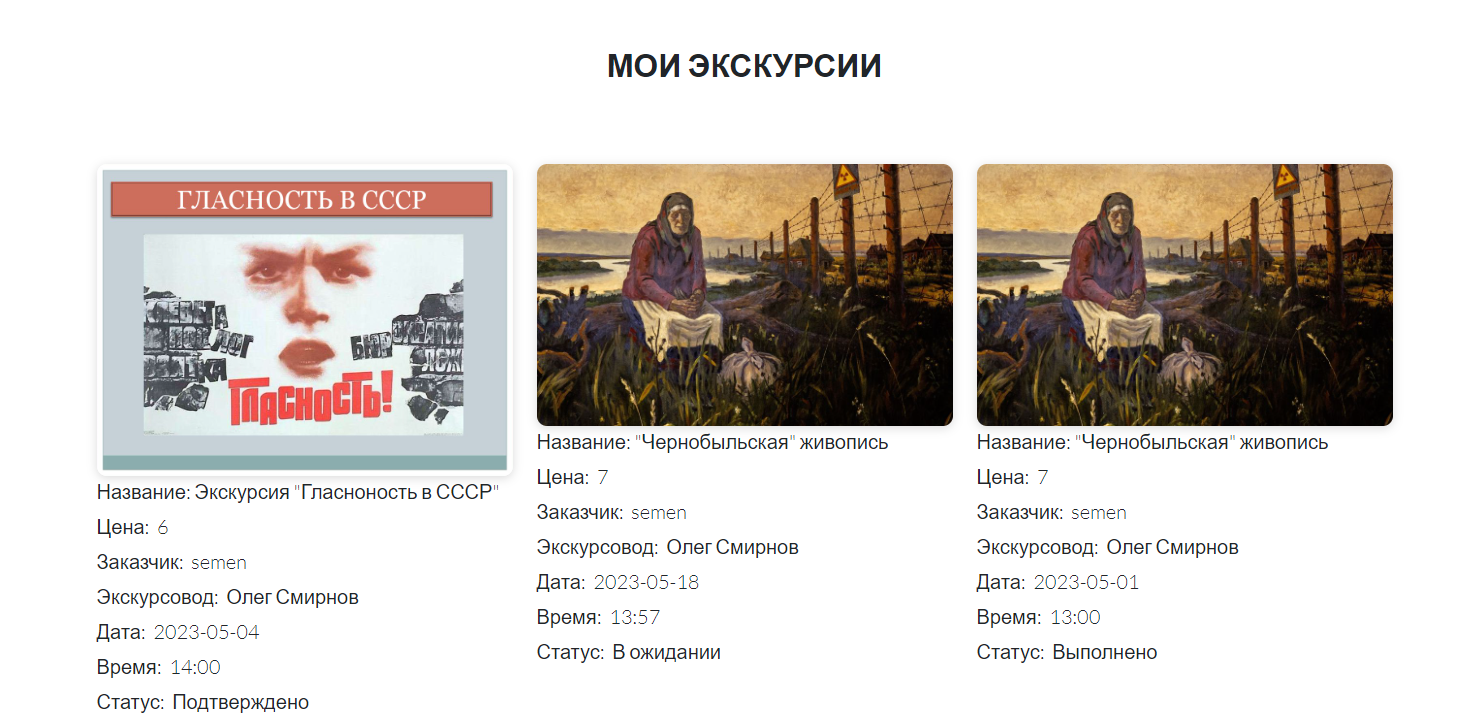


Рисунок 3.5 – Раздел «Мои экскурсии»

Администратор, обладает основными возможностями в системе. Администратором становится первый зарегистрировавшийся пользователь. Он может добавлять, обновлять и удалять категории экскурсий (Рисунок 3.6).

Администратор ответственен за управление пользователями (Рисунок 3.7). Он присуждает пользователям одно из трех доступных категорий (администратор, экскурсовод и клиент) и удаляет их, если потребуется. Также администратору доступен просмотр аналитика деятельности музея. В данном разделе представлены прошедшие экскурсии, их цена и прибыль, которую они принесли.

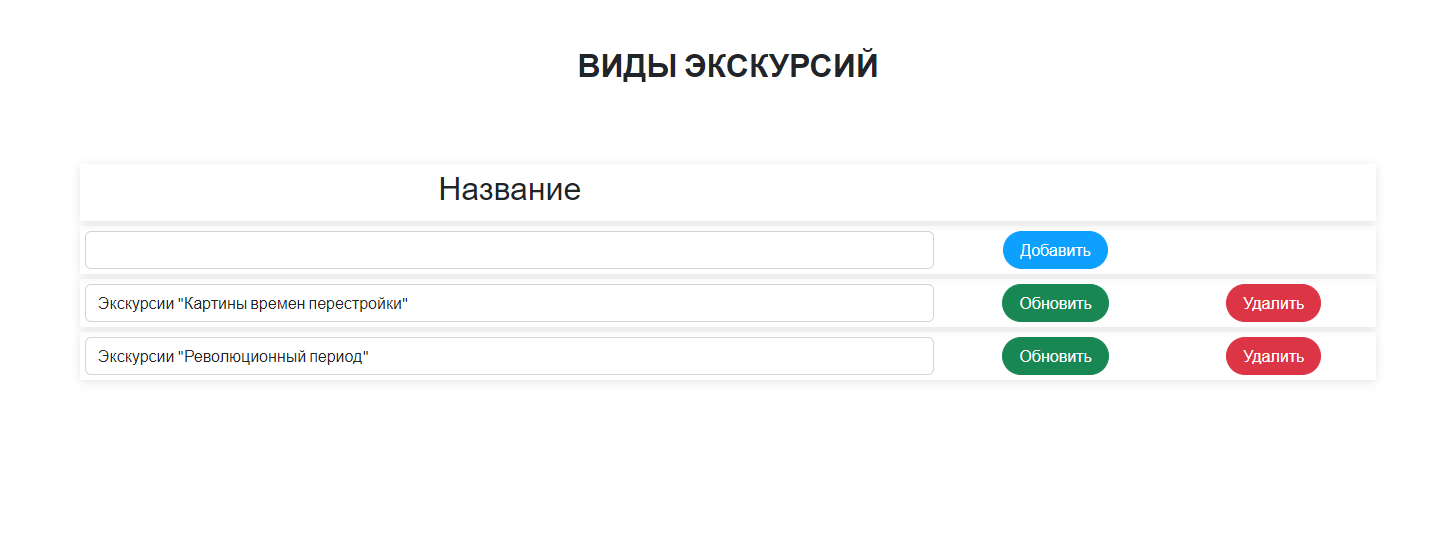


Рисунок 3.6 – Управление категориями экскурсий

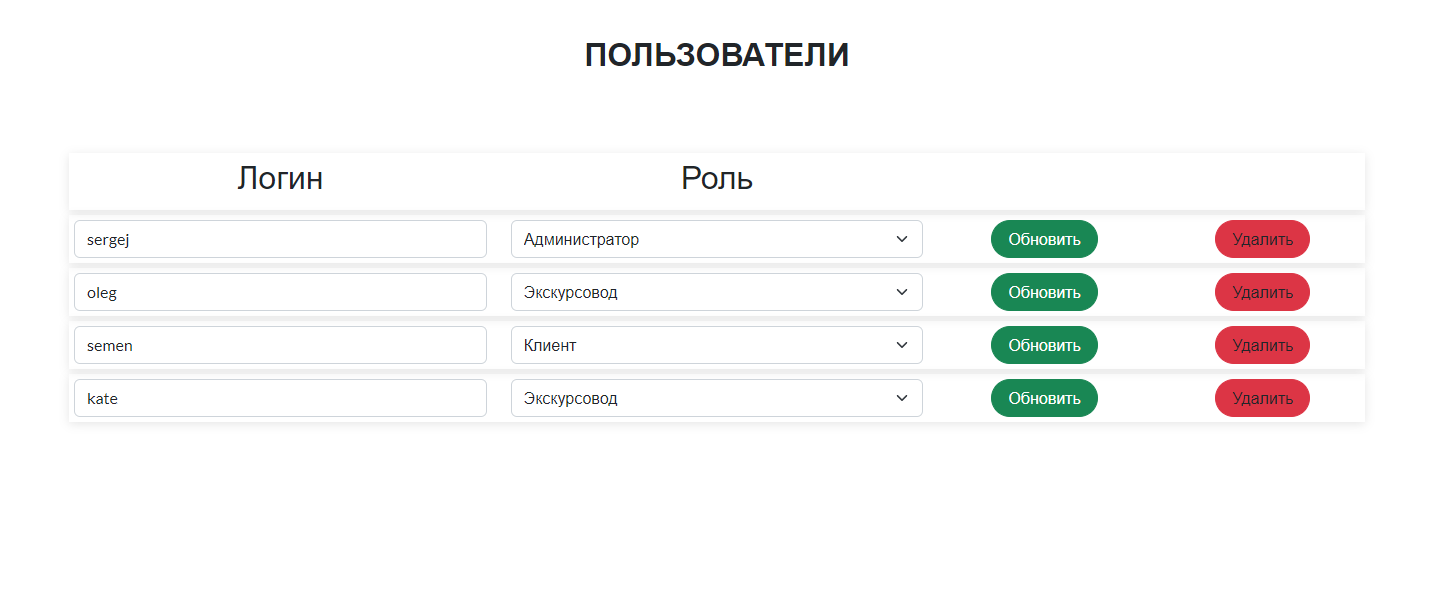


Рисунок 3.7 – Управление пользователями

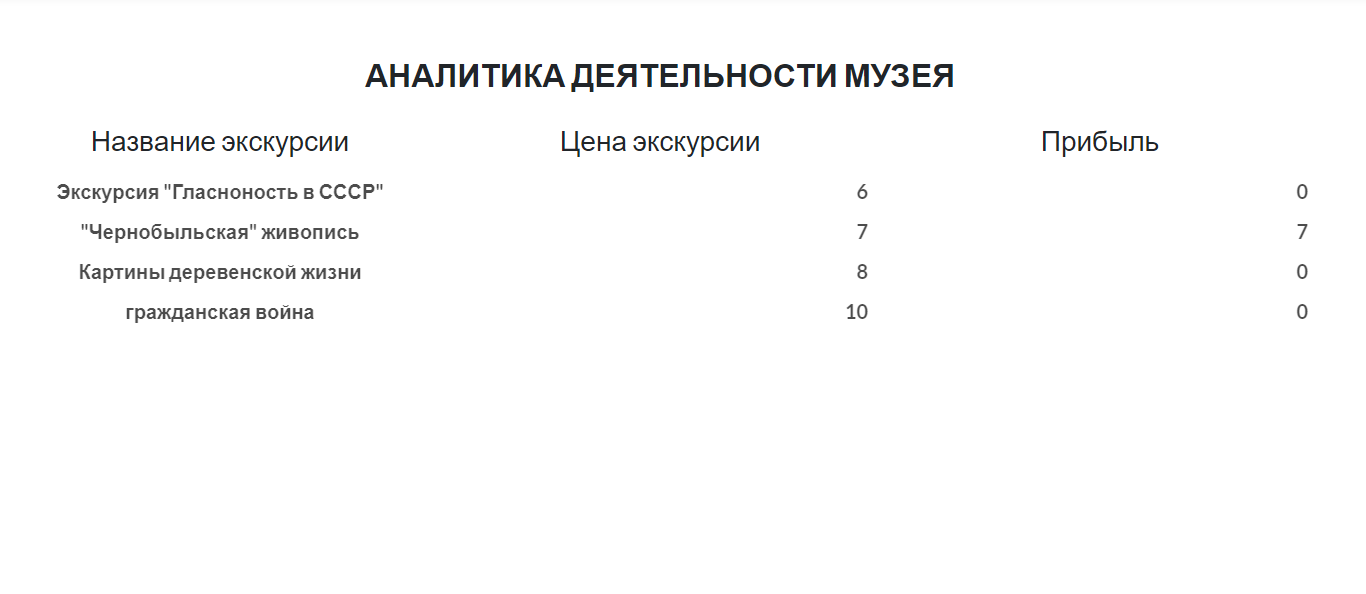


Рисунок 3.7 – Аналитика деятельности музея

Экскурсовод имеет свои специфические возможности. Как уже упоминалось выше, статус экскурсовода предоставляется пользователю администратором. Первоначально экскурсовод может заполнить раздел «Мои данные» (Рисунок 3.8). Он должен ввести свое имя, вид экскурсии, мобильный телефон и опыт работы. Также он может загрузить свое фото. Это необходимо, чтобы пользователи в дальнейшем смогли просматривать информацию об экскурсоводах.

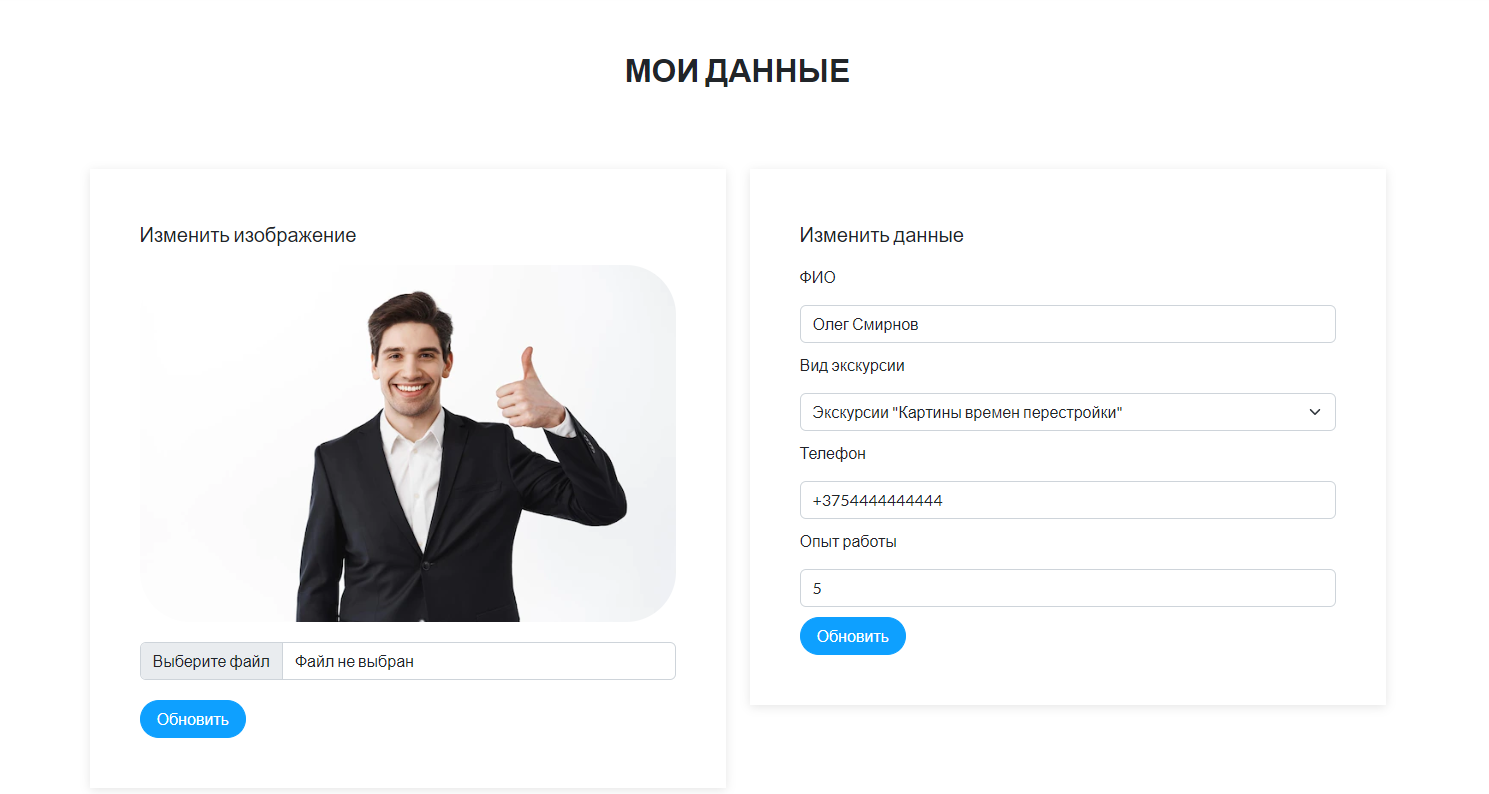


Рисунок 3.8 – Раздел «Мои данные»

Экскурсовод может добавлять новые экскурсии (Рисунок 3.9). Он выбирает определенную категорию, затем пишет название, добавляет фото, цену и описание экскурсии. После чего необходимо нажать кнопку добавить.

Также он может очистить поля в случае ошибки.

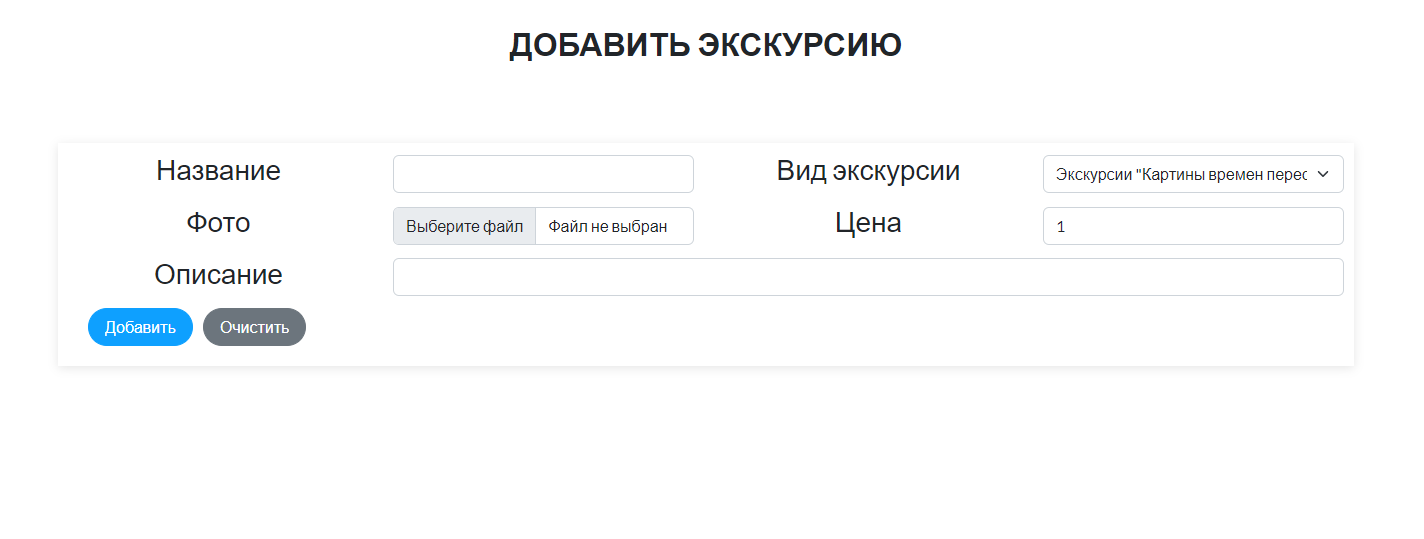


Рисунок 3.9 – Добавление новых экскурсий

Также пользователь может подтверждать или не подтверждать заказанные экскурсии (Рисунок 3.10). После проведения экскурсии он может изменить статус экскурсии на «выполнено». Все прошедшие экскурсии сохраняются в данном разделе, поэтому экскурсовод всегда может их просмотреть при необходимости.

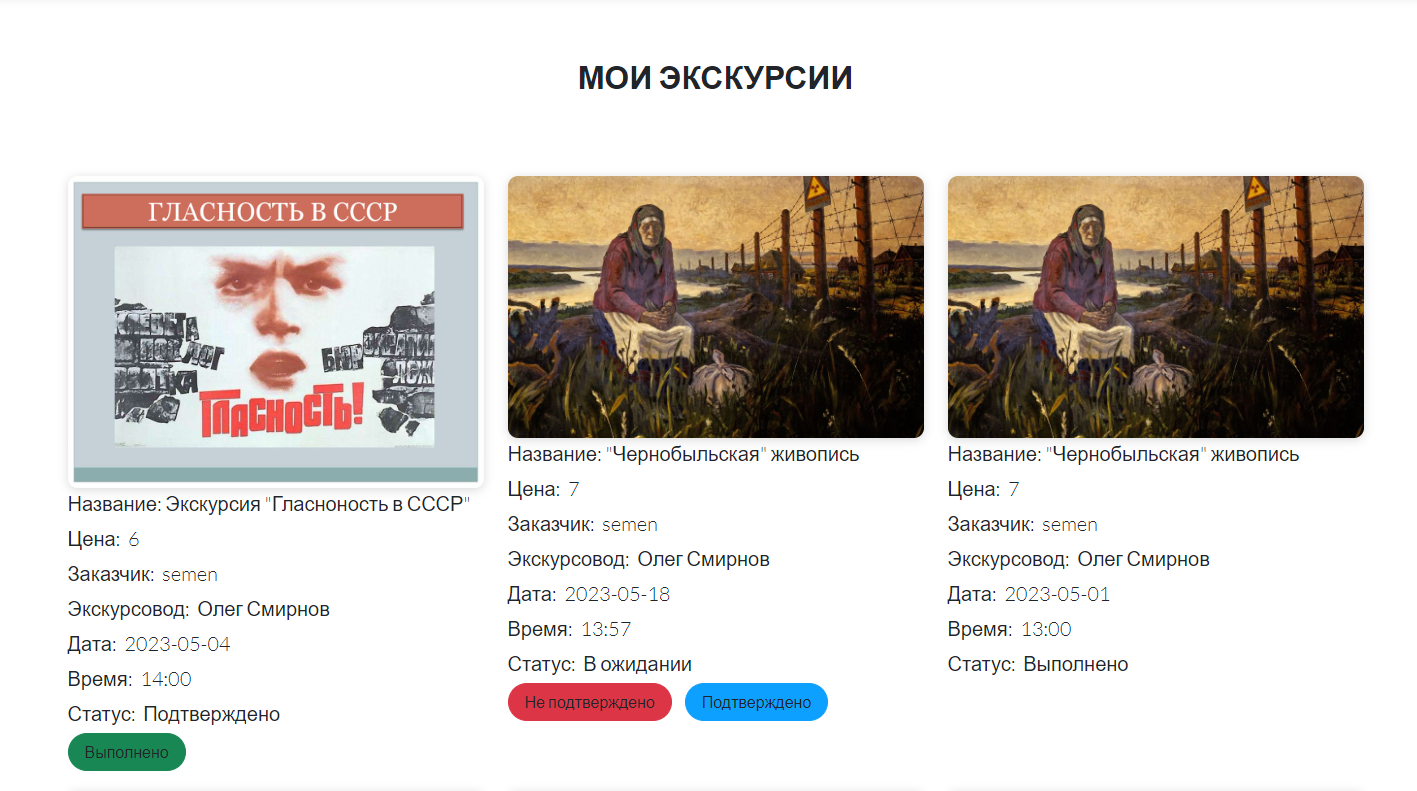


Рисунок 3.10 – работа с экскурсиями

Заключение

При выполнении курсовой работы был произведен анализ предметной области, реализовано программное и информационное обеспечение. Была исследована область и специфика работы музея для более точного понимания постеленной задачи. Были показаны и описаны диаграммы UML, с помощью которых было выполнено проектирование системы. Также была рассмотрена архитектура созданного программного средства. Помимо этого, в ходе выполнения проекта было составлено руководство пользователю, где понятным и доступным языком описывается принцип работы программы.

Приложение имеет дружественный пользовательский интерфейс, описанный в руководстве пользователя. В приложении реализована работа с базой данных PostgreSQL при помощи фреймворка Spring Data.

Проектирование UML-диаграмм позволило сформулировать основные требования к информационной системе, а также обеспечить объективность в выработке требований к проектированию системы. Данные модели описания требований к информационной системе преобразуются в систему моделей, описывающих концептуальный проект информационной системы. При этом сформировались модели архитектуры информационной системы, требования к программному обеспечению и информационному обеспечению

Основные функции программного средства реализованы в соответствии с выявленными особенностями предметной области. Был разработан довольно широкий функционал для работы с информацией, которая содержится в базе данных. Стиль интерфейса программы создавался с упором на массовость потребления и использования, который позволит любому пользователю легко и просто использовать данное программное средство без лишних временных затрат.

Созданное приложение может использоваться для автоматизированного процесса работы музея и, соответственно, оптимизацию деятельности данного учреждения

В будущем возможно рассмотрение вопроса о расширении функционала программы или же усовершенствования имеющегося. Это обеспечит расширение спектра применения разработанного программного средства.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, И. Якобсон. – М. : ДМК Пресс, 2006. – 496 с.
2. Диаграммы классов [Электронный ресурс]. – Режим доступа : <https://flexberry.github.io/ru/gpg_class-diagram.html>

[3] Построение контекстной диаграммы [Электронный ресурс]. – Режим доступа: [file:///D:/pz\_1.html](file:///D:\pz_1.html)

[4] Черемных, С. Моделирование и анализ систем. IDEF-технологии / Черемных С.В., Семенов И.О., Ручкин В.С, Москва: Россия. – 2006. – 192 с.

[5] Батин, Н.В. Проектирование баз данных / Батин Н.В., Минск: Беларусь -2007. – 56 с.

[6] Базы данных [Электронный ресурс]. – Режим доступа: [file:///D:/EORD\_bazy\_dannykh.html](file:///D:\EORD_bazy_dannykh.html)

[6] Национальный правовой интернет-портал Республики Беларусь [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://pravo.by/document/?guid=3871&p0=v19302435>.

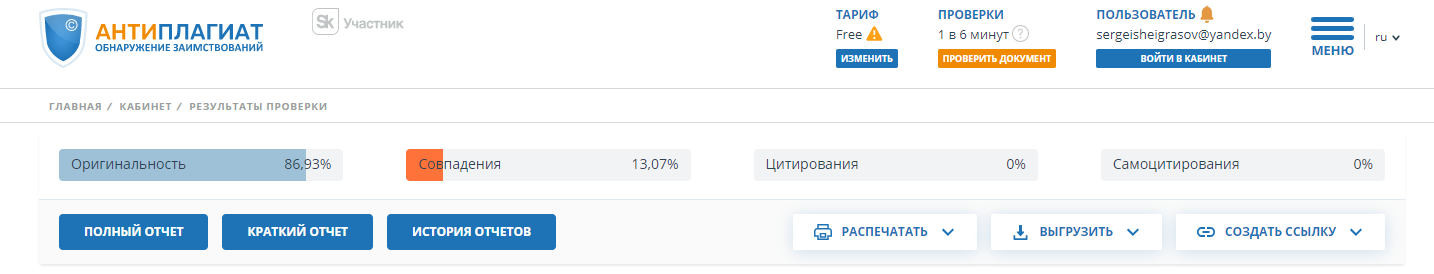
[7] Академик – словари и энциклопедии [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://dic.academic.ru/dic.nsf/business/18231>

[8] Основы предметной области [Электронный ресурс]. – Режим доступа: <https://studfile.net/preview/5659491/page:3/>

[9] Определение термина REST [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/REST

[10] JavaRush [Электронный ресурс]. – Электронные данные. – Режим доступа: https://javarush.ru.

**Приложение А (обязательное) - Отчет о проверке на заимствования в системе «Антиплагиат»**



**Приложение Б (обязательное) - Листинг кода алгоритмов, реализующих бизнес-логику**

**NoteCont**

package com.mus.controller;  
  
import com.mus.controller.main.Attributes;  
import com.mus.model.\*;  
import com.mus.model.enums.StatusOrdering;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.multipart.MultipartFile;  
  
import java.io.File;  
import java.util.List;  
import java.util.Objects;  
import java.util.UUID;  
  
@Controller  
@RequestMapping("/notes")  
public class NotesCont extends Attributes {  
  
 @GetMapping("/add")  
 public String noteAdd(Model model) {  
 AddAttributesNoteAdd(model);  
 return "noteAdd";  
 }  
  
 @GetMapping("/my")  
 public String orderingsMy(Model model) {  
 AddAttributesNotesMy(model);  
 return "notesMy";  
 }  
  
 @GetMapping("/notconf/{id}")  
 public String orderingNotConf(@PathVariable Long id) {  
 Ordering ordering = orderingRepo.getReferenceById(id);  
 ordering.setStatusOrdering(StatusOrdering.*NOT\_CONF*);  
 orderingRepo.save(ordering);  
 return "redirect:/notes/my";  
 }  
  
 @GetMapping("/conf/{id}")  
 public String orderingConf(@PathVariable Long id) {  
 Ordering ordering = orderingRepo.getReferenceById(id);  
 ordering.setStatusOrdering(StatusOrdering.*CONF*);  
 orderingRepo.save(ordering);  
 return "redirect:/notes/my";  
 }  
  
 @GetMapping("/done/{id}")  
 public String orderingDone(@PathVariable Long id) {  
 Ordering ordering = orderingRepo.getReferenceById(id);  
 ordering.setStatusOrdering(StatusOrdering.*DONE*);  
 ordering.getNote().setCount(ordering.getNote().getCount() + 1);  
 orderingRepo.save(ordering);  
 return "redirect:/notes/my";  
 }  
  
 @GetMapping("/{id}")  
 public String note(Model model, @PathVariable Long id) {  
 AddAttributesNote(model, id);  
 return "note";  
 }  
  
 @GetMapping("/edit/{id}")  
 public String noteEdit(Model model, @PathVariable Long id) {  
 AddAttributesNoteEdit(model, id);  
 return "noteEdit";  
 }  
  
 @GetMapping("/delete/{id}")  
 public String noteDelete(@PathVariable Long id) {  
 List<Ordering> orderings = orderingRepo.findAllByNote\_Id(id);  
 for (Ordering i : orderings) {  
 orderingRepo.deleteById(i.getId());  
 }  
 notesRepo.deleteById(id);  
 return "redirect:/";  
 }  
  
 @PostMapping("/ordering/{noteId}")  
 public String noteOrdering(@RequestParam Long masterId, @RequestParam String date, @RequestParam String time, @PathVariable Long noteId) {  
 Masters master = mastersRepo.getReferenceById(masterId);  
 Users client = getUser();  
 Notes note = notesRepo.getReferenceById(noteId);  
  
 Ordering ordering = new Ordering(master, client, note, date, time);  
  
 orderingRepo.save(ordering);  
 return "redirect:/notes/{noteId}";  
 }  
  
 @PostMapping("/add")  
 public String noteAddNew(Model model, @RequestParam String name, @RequestParam Long categoryId, @RequestParam MultipartFile photo, @RequestParam int price, @RequestParam String description) {  
 String res = "";  
 if (photo != null && !Objects.*requireNonNull*(photo.getOriginalFilename()).isEmpty()) {  
 String uuidFile = UUID.*randomUUID*().toString();  
 boolean createDir = true;  
 try {  
 File uploadDir = new File(uploadImg);  
 if (!uploadDir.exists()) createDir = uploadDir.mkdir();  
 if (createDir) {  
 res = "notes/" + uuidFile + "\_" + photo.getOriginalFilename();  
 photo.transferTo(new File(uploadImg + "/" + res));  
 }  
 } catch (Exception e) {  
 model.addAttribute("message", "Некорректный данные!");  
 AddAttributesNoteAdd(model);  
 return "noteAdd";  
 }  
 }  
  
 Notes note = new Notes(name, res, price, description);  
  
 Category category = categoryRepo.getReferenceById(categoryId);  
  
 category.addNote(note);  
  
 categoryRepo.save(category);  
  
 return "redirect:/notes/add";  
 }  
  
 @PostMapping("/edit/{id}")  
 public String noteEditOld(Model model, @RequestParam String name, @RequestParam Long categoryId, @RequestParam MultipartFile photo, @RequestParam int price, @RequestParam String description, @PathVariable Long id) {  
 Notes note = notesRepo.getReferenceById(id);  
 if (photo != null && !Objects.*requireNonNull*(photo.getOriginalFilename()).isEmpty()) {  
 String res = "";  
 String uuidFile = UUID.*randomUUID*().toString();  
 boolean createDir = true;  
 try {  
 File uploadDir = new File(uploadImg);  
 if (!uploadDir.exists()) createDir = uploadDir.mkdir();  
 if (createDir) {  
 res = "notes/" + uuidFile + "\_" + photo.getOriginalFilename();  
 photo.transferTo(new File(uploadImg + "/" + res));  
 }  
 } catch (Exception e) {  
 model.addAttribute("message", "Некорректный данные!");  
 AddAttributesNoteEdit(model, id);  
 return "noteEdit";  
 }  
 note.setPhoto(res);  
 }  
  
 note.setName(name);  
 note.setPrice(price);  
 note.setDescription(description);  
  
 Category categoryNew = categoryRepo.getReferenceById(categoryId);  
 if ((!categoryNew.getId().equals(note.getCategory().getId()))) {  
 Category categoryOld = note.getCategory();  
 categoryOld.removeNote(note);  
 categoryNew.addNote(note);  
 }  
 notesRepo.save(note);  
  
 return "redirect:/";  
 }  
}

**UsersCont.java**

package com.mus.controller;  
  
import com.mus.controller.main.Attributes;  
import com.mus.model.Masters;  
import com.mus.model.Users;  
import com.mus.model.enums.Role;  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.\*;  
  
@Controller  
@RequestMapping("/users")  
public class UsersCont extends Attributes {  
 @GetMapping  
 public String subs(Model model) {  
 AddAttributesUsers(model);  
 return "users";  
 }  
  
 @PostMapping("/edit/{id}")  
 public String editUser(@PathVariable Long id, @RequestParam Role role) {  
 Users user = usersRepo.getReferenceById(id);  
  
 if (role != Role.*MASTER* && user.getRole() == Role.*MASTER*) {  
 Masters master = user.getMaster();  
 user.setMaster(null);  
 mastersRepo.deleteById(master.getId());  
 }  
 if (role == Role.*MASTER* && user.getRole() != Role.*MASTER*) {  
 Masters master = new Masters(defPhoto);  
 mastersRepo.save(master);  
 user.setMaster(master);  
 }  
  
 user.setRole(role);  
 usersRepo.save(user);  
 return "redirect:/users";  
 }  
  
 @GetMapping("/delete/{id}")  
 public String deleteUser(Model model, @PathVariable Long id) {  
 Users user = usersRepo.getReferenceById(id);  
 if (user == getUser()) {  
 AddAttributesUsers(model);  
 model.addAttribute("message", "Вы не можете удалить свой профиль!");  
 return "users";  
 }  
 usersRepo.deleteById(id);  
 return "redirect:/users";  
 }  
}

**Ordering.java**

package com.mus.model;  
  
import com.mus.model.enums.StatusOrdering;  
import lombok.AccessLevel;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
  
import javax.persistence.\*;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@Entity  
public class Ordering {  
 @Setter(AccessLevel.*NONE*)  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(nullable = false, updatable = false)  
 private Long id;  
  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 private Masters master;  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 private Users client;  
 @ManyToOne(fetch = FetchType.*LAZY*)  
 private Notes note;  
 private String date;  
 private String time;  
 @Enumerated(EnumType.*STRING*)  
 private StatusOrdering statusOrdering;  
  
 public Ordering(Masters master, Users client, Notes note, String date, String time) {  
 this.master = master;  
 this.client = client;  
 this.note = note;  
 this.date = date;  
 this.time = time;  
 this.statusOrdering = StatusOrdering.*WAITING*;  
 }  
}

**Users.java**

package com.mus.model;  
  
import com.mus.model.enums.Role;  
import lombok.AccessLevel;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
  
import javax.persistence.\*;  
import java.util.Collection;  
import java.util.Collections;  
import java.util.List;  
  
@Getter  
@Setter  
@NoArgsConstructor  
@Entity  
public class Users implements UserDetails {  
 @Setter(AccessLevel.*NONE*)  
 @Id  
 @GeneratedValue(strategy = GenerationType.*AUTO*)  
 @Column(nullable = false, updatable = false)  
 private Long id;  
 @Enumerated(EnumType.*STRING*)  
 private Role role;  
 private String username;  
 private String password;  
  
 @OneToOne(fetch = FetchType.*LAZY*)  
 private Masters master;  
 @OneToMany(mappedBy = "client", fetch = FetchType.*LAZY*, orphanRemoval = true, cascade = CascadeType.*ALL*)  
 private List<Ordering> orderings;  
 public Users(String username, String password, Role role) {  
 this.role = role;  
 this.username = username;  
 this.password = passwordEncoder().encode(password);  
 }  
  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
  
 @Override  
 public boolean isEnabled() {  
 return true;  
 }  
  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return Collections.*singleton*(getRole());  
 }  
  
 private PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder(12);  
 }  
}