Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Инженерно-экономический факультет

Кафедра экономики

Дисциплина: Распределенные системы обработки информации

Курсовая работа

на тему

«Автоматизированная система работы груминг салона»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент 3-го курса группы 074002 специальности «Электронный маркетинг» |  |  |  | Рощина Ирина Сергеевна |
|  |  | *(подпись и дата)* |  |  |
| Руководитель: |  |  |  | Лыщик Артём Петрович |
|  |  | *(подпись и дата)* |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Минск, 2023

**СОДЕРЖАНИЕ**

[СОДЕРЖАНИЕ 4](#_Toc131204028)

[ВВЕДЕНИЕ 4](#_Toc131204029)

[1 АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА 6](#_Toc131204030)

[1.1 Описание предметной области 6](#_Toc131204031)

[1.2 Разработка функциональной модели предметной области 7](#_Toc131204032)

[1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований 10](#_Toc131204033)

[1.4 Разработка информационной модели предметной области 12](#_Toc131204034)

[1.5 UML-модели представления программного средства и их описания 14](#_Toc131204035)

[2 ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 16](#_Toc131204036)

[2.1 Постановка задачи 16](#_Toc131204037)

[2.2 Обоснование выбора компонентов и технологий для реализации программного средства 16](#_Toc131204038)

[2.3 Архитектурные решения 20](#_Toc131204039)

[2.4 Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства 22](#_Toc131204040)

[2.5 Проектирование пользовательского интерфейса 23](#_Toc131204041)

[ЗАКЛЮЧЕНИЕ 25](#_Toc131204042)

[Список использованных источников 26](#_Toc131204043)

[ПРИЛОЖЕНИЕ А 27](#_Toc131204044)

[ПРИЛОЖЕНИЕ Б 27](#_Toc131204045)

[ПРИЛОЖЕНИе В 27](#_Toc131204046)

**ВВЕДЕНИЕ**

# **1 АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА**

## **Описание предметной области**

Необходимость создания веб-приложения обуславливается необходимостью автоматизации существующих информационных процессов. Перед созданием веб-приложения необходимо поставить перед собой ряд вопросов: для достижения каких целей необходима разработка, насколько быстро нужно разработать систему, а также, какова величина затрат для того, чтобы осуществить его проектирование.

В интернете существует немало коммерческих подходов. В нем можно рекламировать стандартные услуги и продавать товары, он открывает реальные перспективы электронной коммерции. На современном этапе развития можно выделить два основных направления использования интернета в бизнесе: технологии интернета для бизнеса и бизнес в интернет-пространстве.

Салон красоты для животных предоставляет услуги, которые позволяют удовлетворить спрос людей, на уходовые процедуры для их питомцев.

Целевая аудитория салона для животных – люди, которые ухаживают за своими домашними животными и хотят обеспечить им качественный уход и удобства.

Чаще всего целевая аудитория включает людей, которые ищут профессиональные услуги для своих животных, такие как стрижка, мытье, уход за зубами и когтями, ветеринарные услуги и т.д.

Веб-приложение позволяет автоматизировать и структурировать необходимые для целевой аудитории услуги, обеспечить возможность быстрой записи на необходимую услугу, предоставить все возможный спектр услуг с их кратким описанием, также предоставить информацию о стоимости услуги и информацию о грумерах с отзывами клиентов. Помимо прочего веб-приложение выполняет информационно-ознакомительную функцию: приложение проводит ознакомление потенциальных клиентов с предоставляемыми услугами, контактными данными сотрудников. Веб-приложение представляет посетителям сайта основную информацию, позволяет хранить и удалять данные о грумерах и клиентах.

Можно выделить следующие услуги, предоставляемые веб-приложением:

- стрижка, данный комплекс включает в себя набор из нескольких услуг, таких как вычесывание колтунов, мойка, сушка, стрижка, этот комплекс рекомендуется посещать не реже чем раз в два месяца;

- мойка и сушка, данный комплекс отличается от предыдущего отсутствием стрижки, этот комплекс является более повседневным и рекомендуется к посещению раз в месяц;

- подстригание когтей, данная процедура подразумевает очистку лап от грязи и стрижку когтей, рекомендуется к посещению раз в месяц или чаще, если это необходимо;

- чистка зубов, данная процедура подразумевает гигиеническую чистку зубов, а также удаление зубного камня и налета с помощью ультразвука, если необходимо.

Организационная структура распределения рабочих процессов компании представлена на рисунке 1.1:

Рисунок 1.1 – Организационная структура салона красоты для животных

Схема взаимодействия клиентов и мастеров салона посредством связи через администрацию компании представлена на рисунке 1.2:

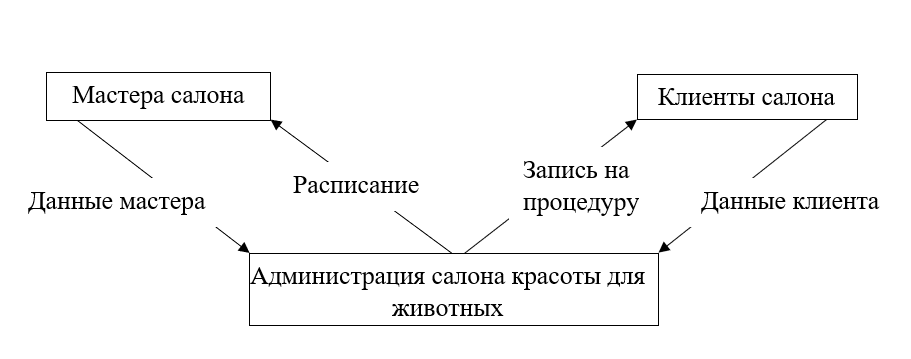


Рисунок 1.2 – Схема взаимодействий в салоне красоты для животных

## **Разработка функциональной модели предметной области**

Разработка функциональной модели предметной области – процесс создания описания того, как система или процесс должны функционировать в рамках определенной предметной области. Функциональная модель обычно состоит из блоков, которые описывают различные функции, процессы и задачи, выполняемые в предметной области.

В результате создания функциональной модели предметной области, получается диаграмма, которая показывает, как должна работать система или процесс в рамках данной предметной области. Это может быть полезно при проектировании системы или процесса, а также при оценке возможных изменений в них.

На контекстной диаграмме верхнего уровня представлена функциональная модель «Деятельность салона» (рисунок 1.3).

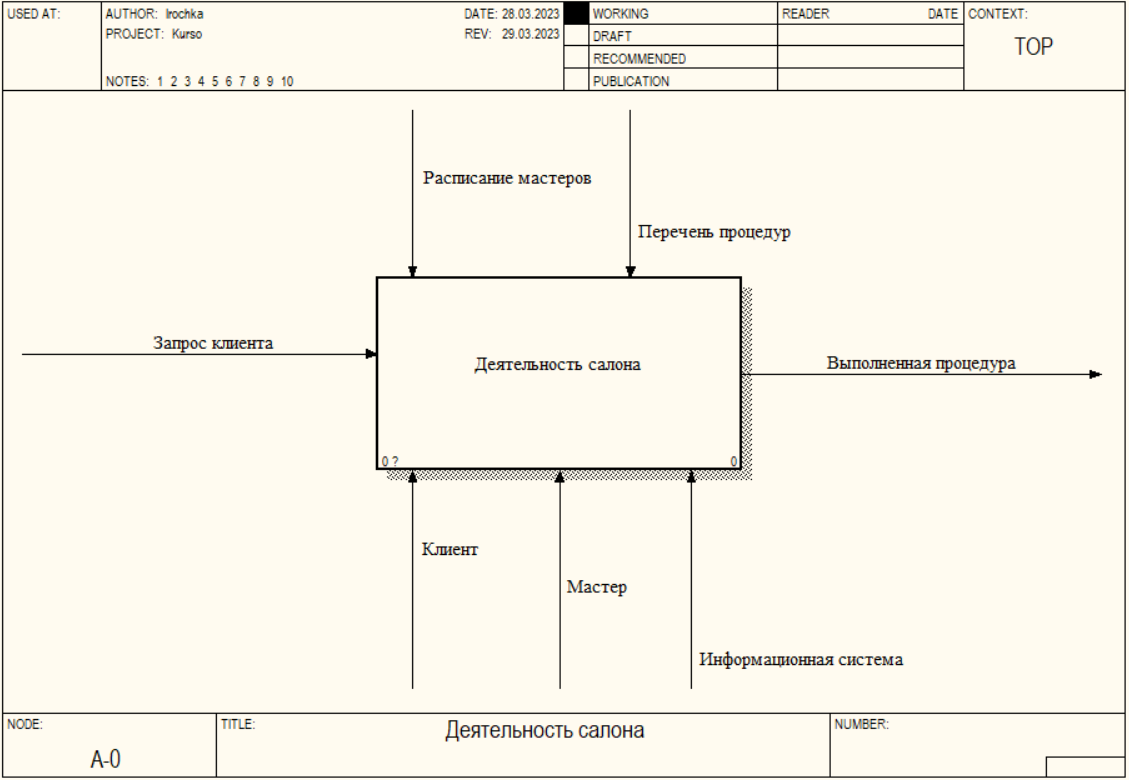


Рисунок 1.3 – Контекстная диаграмма «Деятельность салона»

Были определены входные и выходные данные, механизмы и управление.

Входными данными является запрос клиента; механизмами являются информационная система, клиент и мастер; управлением являются расписание мастеров, перечень процедур. Главной целью бизнес-процесса является выполненная процедура.

Для контекстной диаграммы можно выделить следующие подпроцессы (рисунок 1.4):

- создать заказ;

- принять заказ;

- выполнить услугу;

- подтвердить выполнение услуги.

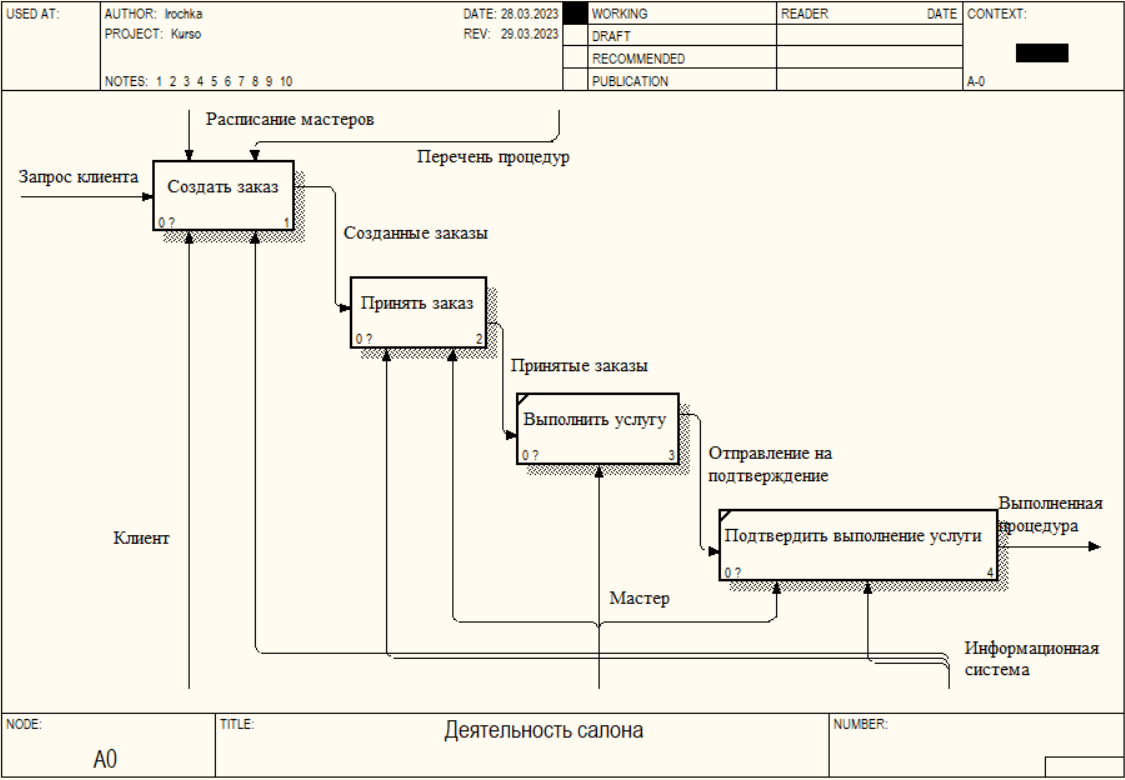


Рисунок 1.4 – Декомпозиция контекстной диаграммы «Деятельность салона»

Процесс начинается с создания заказа клиентом. Здесь мастер узнает, что необходимо клиенту согласно перечню предлагаемых услуг, и принимает заказ. Далее мастеру необходимо выполнить услугу и подтвердить ее выполнение, на этом процесс заканчивается.

Декомпозиция блока создания заказа представлена на рисунке 1.5.

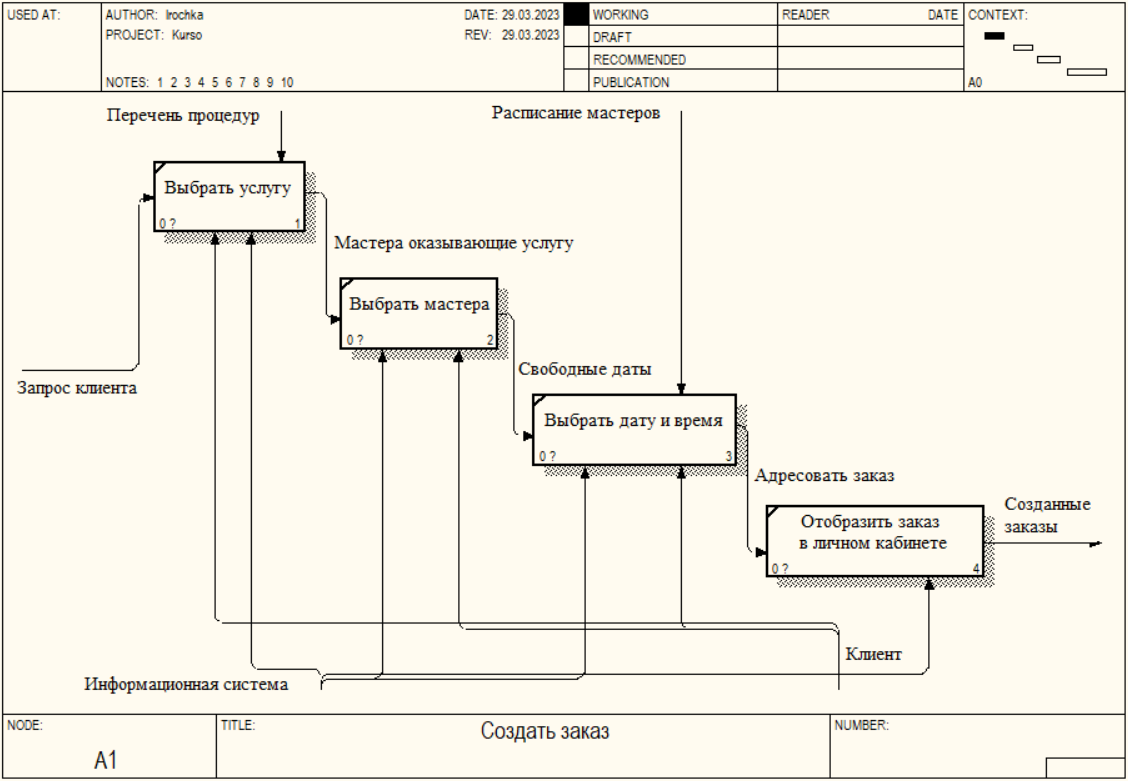


Рисунок 1.5 – Декомпозиция блока «Создать заказ»

Данная декомпозиция состоит из следующих этапов: «Выбрать услугу», «Выбрать мастера», «Выбрать дату и время», «Отобразить заказ в личном кабинете».

Анализируя запросы клиентов, составляется база запросов, которые необходимо будет принять мастерам для удовлетворения своих клиентов.

Вторым блоком декомпозиции главного процесса является блок «Принять заказ» (рисунок 1.6).

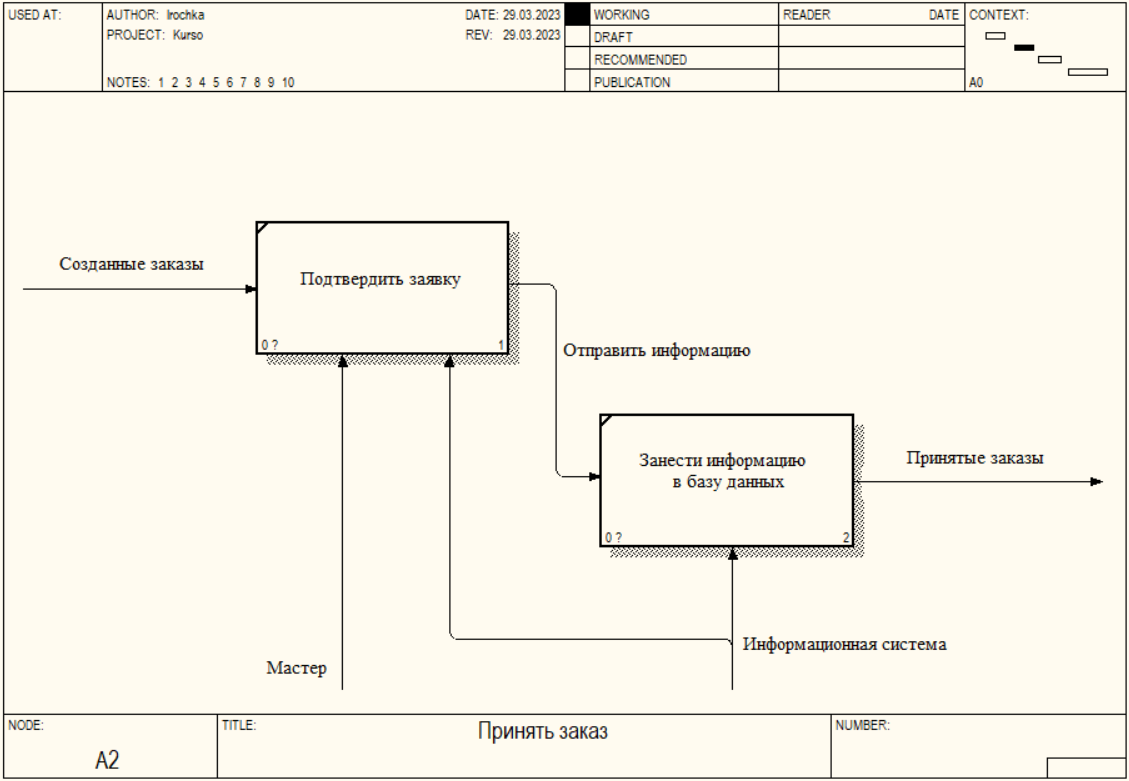


Рисунок 1.6 – Декомпозиция блока «Принять заказ»

Данная декомпозиция состоит из следующих блоков: «Подтвердить заявку», «Занести информацию в базу данных».

В результате мастер выполняет процедуру и отмечает ее как выполненную, что является подтверждением того, что процесс работы с клиентом является завершенным.

|  |  |
| --- | --- |
| **Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований** |  |

С целью анализа требований к разрабатываемому программному средству будет представлена диаграмма вариантов использования и ее описание.

Диаграмма вариантов использования (Use Case Diagram) – графическая модель, которая показывает отношения между актерами (пользователями) и функциональностью системы в виде взаимодействий. Она часто используется в процессе проектирования программного обеспечения и позволяет лучше понимать, как система будет использоваться пользователем.

Диаграмма вариантов использования состоит из актеров, вариантов использования и связей между ними. Актеры – внешние сущности, которые взаимодействуют с системой. Варианты использования – конкретные действия или задачи, которые пользователь может выполнить с помощью системы.

Связи между актерами и вариантами отображаются с использованием отношений четырех видов:

- ассоциаций – служит для обозначения взаимодействия актера с вариантом использования;

- обобщения – служит для указания того факта, что некоторая сущность А может быть обобщена до сущности В;

- включения (зависимость со стереотипом «include») – указывает, что некоторое заданное поведение одного варианта использования обязательно включается в качестве составного компонента в последовательность поведения другого варианта использования;

- расширения (зависимость со стереотипом «extend») – определяет потенциальную возможность включения поведения одного варианта использования в состав другого.

В курсовой работе есть 3 актера: мастер, клиент и администратор (рисунок 1.7)

Клиент сайта салона может:

- просматривать услуги;

- записываться на услугу;

- просматривать свои записи;

- просматривать профили мастеров (грумеров).

Мастер салона может:

- просматривать услуги;

- изменять свой профиль;

- добавлять услуги;

- подтверждать/отклонять записи;

- выполнять записи;

- просматривать свои записи;

- просматривать профили мастеров (грумеров).

Администратор сайта может:

- просматривать услуги;

- просматривать статистику;

- просматривать профили мастеров (грумеров);

- просматривать пользователей;

- добавлять категории услуг.

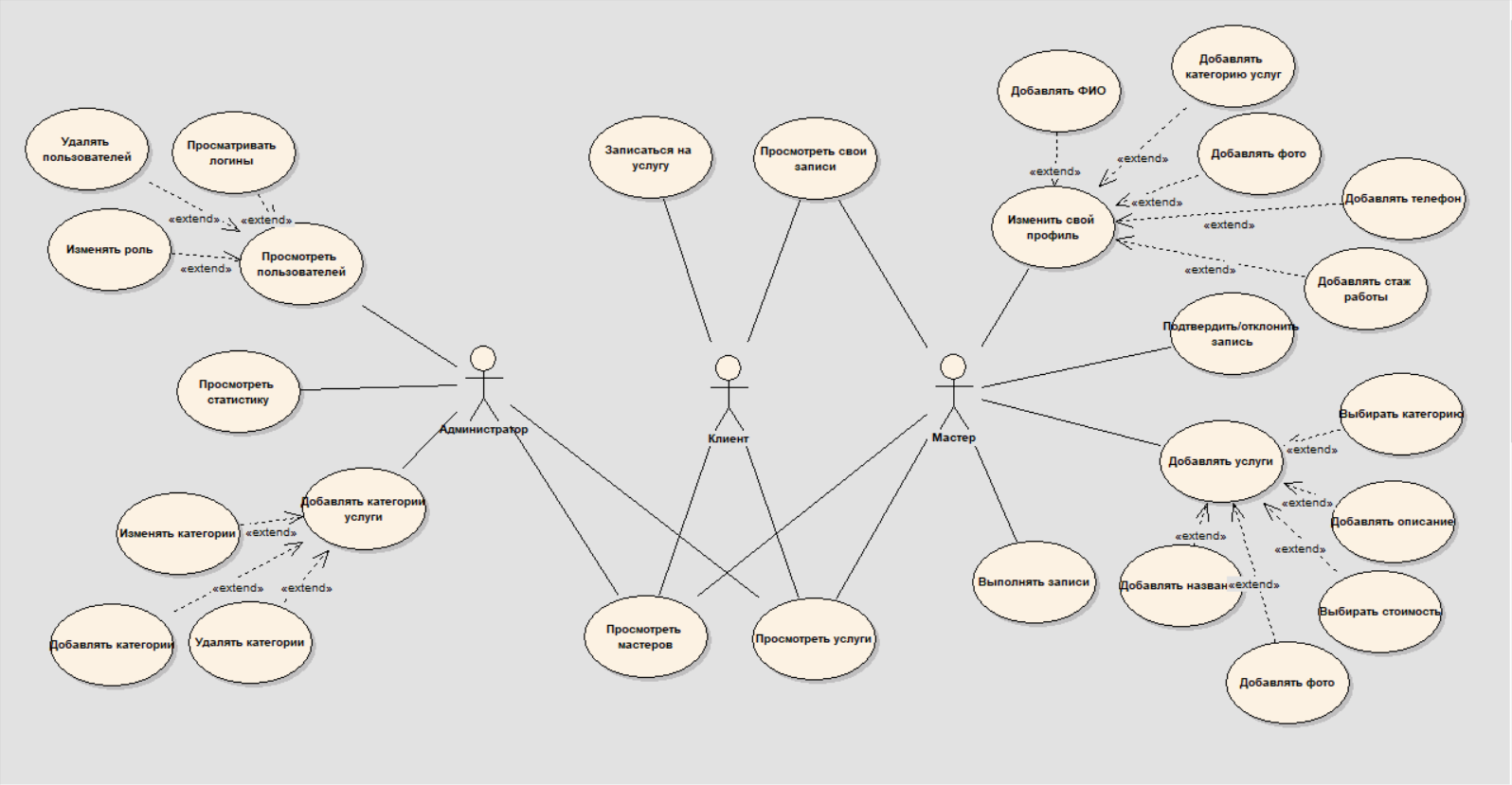


Рисунок 1.7 – Диаграмма вариантов использования

|  |  |
| --- | --- |
| **Разработка информационной модели предметной области** |  |

Информационная модель предметной области – формализованное описание предметной области, которое отражает основные объекты, свойства и взаимосвязи между ними, а также операции, которые могут выполняться над этими объектами.

Информационная модель может быть представлена в виде графической схемы, таблицы или другого формата. В процессе разработки информационной модели проводится анализ предметной области, определяются основные элементы и структуры, выявляются свойства объектов и взаимосвязи между ними.

Основной целью создания информационной модели является обеспечение полного и точного понимания предметной области и создание основы для создания базы данных. Информационная модель позволяет описать структуру данных, которые будут храниться в базе данных, а также определить, как эти данные будут использоваться и обрабатываться в рамках различных бизнес-процессов.

В процессе разработки системы была сформирована информационная модель системы (см. рисунок 1.8). В ходе моделирования были выделены следующие сущности:

- category;

- notes;

- ordering;

- users;

- masters.

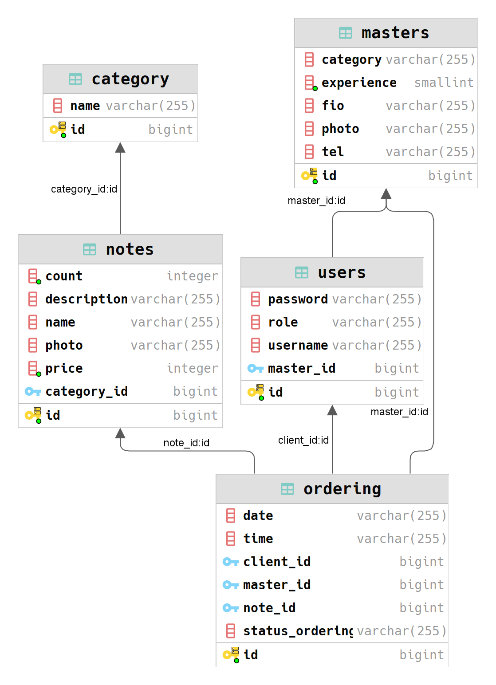


Рисунок 1.8 – Информационная модель системы

Сущность «category» включает в себя следующие атрибуты:

- «name» – название категории (направления);

- «id» – первичный ключ таблицы.

Сущность «notes» включает в себя следующие атрибуты:

- «discription» – описание услуги;

- «name» – название услуги;

- «photo» – имя картинки;

- «price» – стоимость услуги;

- «count» – сумма проданных процедур;

- «category\_id» – внешний ключ связанной таблицы;

- «id» – первичный ключ таблицы.

Сущность «ordering» включает в себя следующие атрибуты:

- «date» – дата записи;

- «time» – время записи;

- «client\_id» – внешний ключ связанной таблицы;

- «master\_id» – внешний ключ связанной таблицы;

- «note\_id» – внешний ключ связанной таблицы;

- «status\_ordering» – статус записи;

- «id» – первичный ключ таблицы.

Сущность «users» включает в себя следующие атрибуты:

- «password» – пароли пользователей;

- «role» – роль пользователя;

- «username» – логины пользователей;

- «master\_id» – внешний ключ связанной таблицы;

- «id» – первичный ключ таблицы.

Сущность «masters» включает в себя следующие атрибуты:

- «category» – категория (направление) мастера;

- «experience» – стаж;

- «fio» – фио мастера;

- «photo» – имя картинки;

- «tel» – телефон мастера;

- «id» – первичный ключ таблицы.

|  |  |
| --- | --- |
| **UML-модели представления программного средства и их описания** |  |

Диаграмма последовательности (Sequence diagram) – графическое представление взаимодействия объектов в рамках определенного сценария. Это один из самых распространенных типов диаграмм UML (Unified Modeling Language) и используется для моделирования динамических аспектов системы.

Диаграмма последовательности показывает последовательность сообщений, отправляемых между объектами, и временные интервалы, в течение которых эти сообщения передаются. Каждый объект на диаграмме представлен вертикальной линией, называемой "жизненным циклом объекта". Сообщения представляются стрелками, направленными от одного объекта к другому, и сопровождаются названием операции и аргументами.

Диаграмма последовательности используется для представления различных сценариев взаимодействия между объектами, таких как выполнение методов, вызовы функций, передача параметров и т.д. Она также может быть использована для анализа производительности и оптимизации процессов.

Диаграмма последовательности курсовой работы представлена на рисунке 1.9.

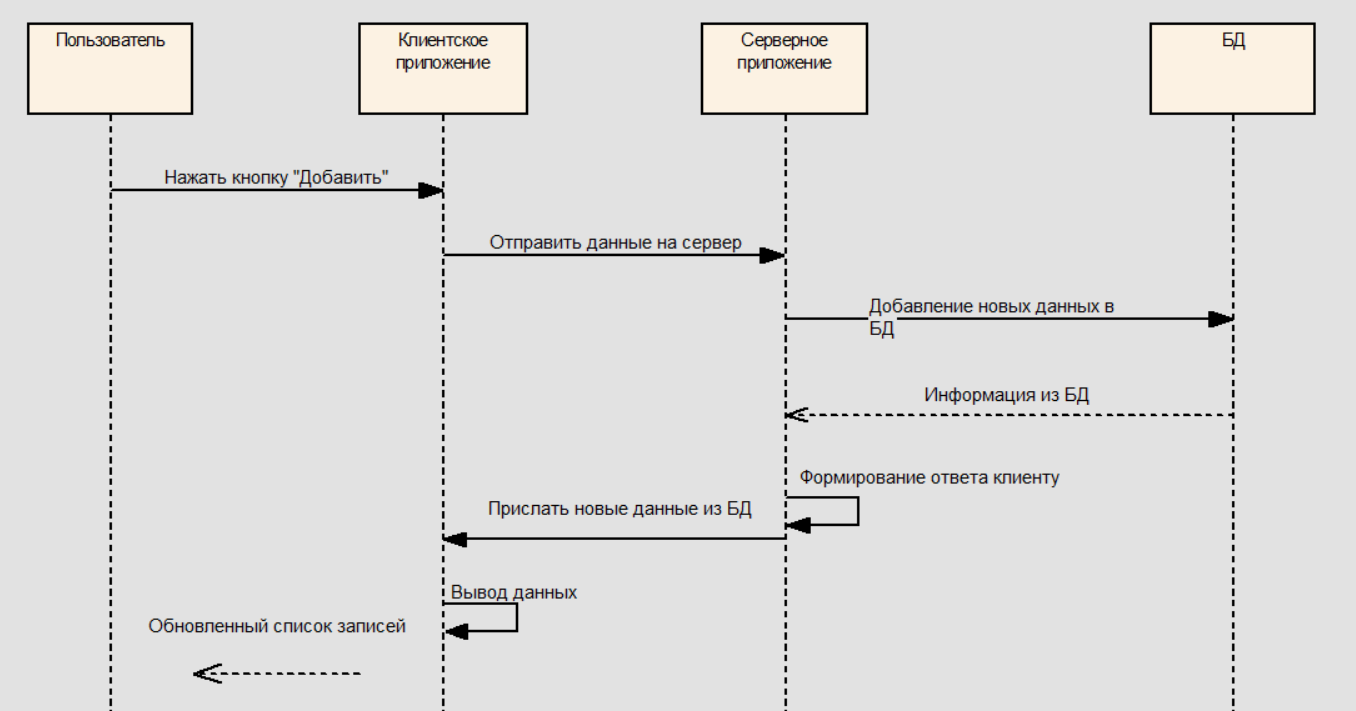


Рисунок 1.9 – Диаграмма последовательности

Диаграмма состояний курсовой работы представлена на рисунке 1.10.

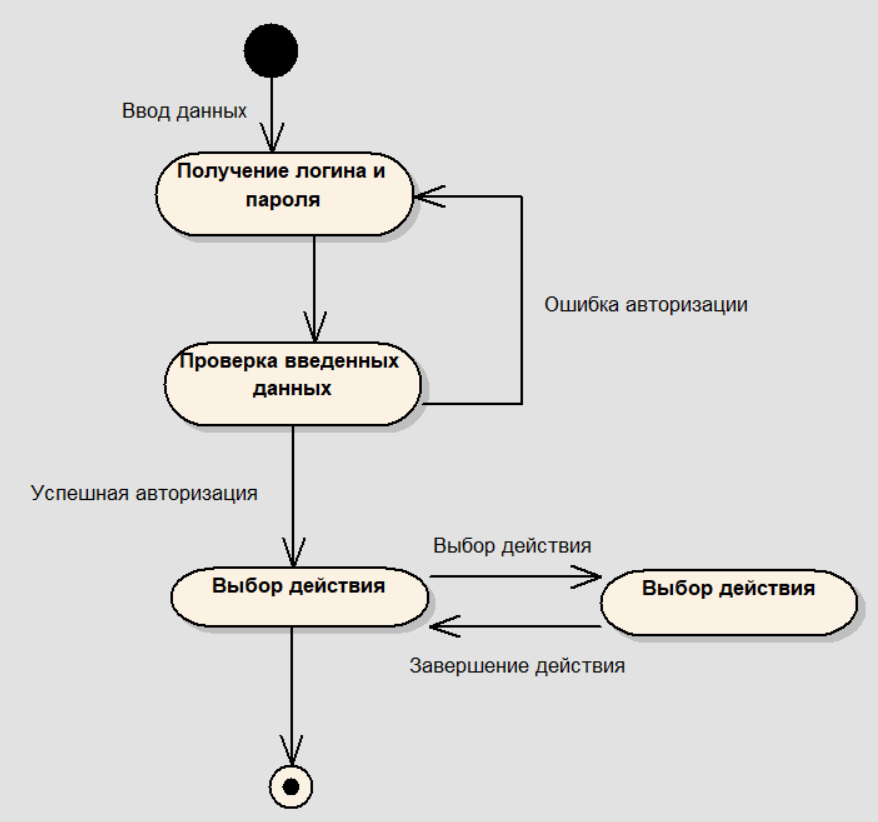


Рисунок 1.10 – Диаграмма состояний

# **ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## **Постановка задачи**

Задачей курсовой работы является разработка веб-приложения для автоматизации рабочего процесса салона красоты для животных, которое позволит клиентам записываться на услуги, а администратору отслеживать аналитику записей.

Первостепенной целью разработки данного программного продукта является упрощение и ускорение рабочего процесса салона, в частности, регистрации клиентов, их записи на услуги, составление графиков работы мастеров.

Программное средства должно обладать следующими пользовательскими функциями:

- просмотр информации об услугах и мастерах на сайте;

- возможностью регистрации и записи на процедуры.

Программное средства должно обладать следующими функциями для мастера:

- просмотр информации об услугах и мастерах на сайте;

- возможностью регистрации и изменения информации о себе;

- возможность добавлять и изменять услуги.

Программное средства должно обладать следующими функциями для администратора:

- просмотр информации об услугах и мастерах на сайте;

- возможностью добавлять категории услуг и изменять их;

- возможностью изменять роль и удалять пользователей;

Возможностью просматривать аналитику записей.

В ходе реализации необходимо разделить роли администратора, клиента и мастера.

Данное приложение должно облегчить работу администратора, а также позволить клиентам записывать своих питомцев на процедуры в любое удобное для них время.

## **Обоснование выбора компонентов и технологий для реализации программного средства**

PostgreSQL (или просто Postgres) – мощная объектно-реляционная система управления базами данных (СУБД), которая позволяет эффективно хранить, организовывать и манипулировать большими объемами структурированных данных. PostgreSQL является свободной и открытой программой с открытым исходным кодом и поддерживает широкий спектр операционных систем, включая Linux, Unix и Windows.

PostgreSQL предоставляет множество функций и возможностей, таких как поддержка транзакций, проверка целостности данных, гибкий язык запросов SQL, процедурное расширение на языке PL/pgSQL и многие другие. Благодаря своей надежности и масштабируемости PostgreSQL широко используется в качестве базы данных для крупных приложений, включая социальные сети, электронную коммерцию и финансовые системы.

Некоторые из основных возможностей Postgres:

1. Масштабируемость: PostgreSQL может управлять большим объемом данных и поддерживать множество соединений с базой данных одновременно.
2. Гибкость: PostgreSQL позволяет определять пользовательские типы данных, функции и операторы.
3. Транзакционность: PostgreSQL поддерживает транзакционность, что позволяет гарантировать целостность данных при выполнении нескольких операций в одной транзакции.
4. Безопасность: PostgreSQL предоставляет механизмы авторизации и аутентификации, а также возможность управления доступом к базе данных.
5. Индексация: PostgreSQL поддерживает различные типы индексов, которые могут использоваться для ускорения выполнения запросов к базе данных.
6. Репликация: PostgreSQL поддерживает репликацию данных, что позволяет создавать резервные копии базы данных и обеспечивать отказоустойчивость системы.
7. Расширяемость: PostgreSQL поддерживает плагины и расширения, которые позволяют добавлять дополнительную функциональность к базе данных.

Java Spring – фреймворк для разработки приложений на языке Java. Он предоставляет набор инструментов и библиотек, которые помогают упростить процесс разработки и улучшить производительность. Spring является одним из самых популярных и широко используемых фреймворков для разработки приложений на Java.

Spring был создан для упрощения разработки приложений на Java и предоставления более эффективного способа интеграции компонентов и библиотек. Он состоит из различных модулей, которые могут быть использованы в сочетании или отдельно для создания приложения.

Одним из наиболее известных модулей Spring является Spring MVC, который предоставляет архитектуру Model-View-Controller для построения веб-приложений. Вместе с Spring Boot, который предоставляет упрощенный способ создания приложений на основе Spring, Spring является одним из наиболее распространенных фреймворков для разработки приложений на Java.

MVC (Model-View-Controller) – это архитектурный паттерн, который используется для разработки веб-приложений. Он разделяет приложение на три основных компонента: Model, View и Controller. Каждый из этих компонентов выполняет свою специфическую функцию, которая позволяет разработчикам разделять логику и данные, обрабатывать пользовательский ввод и представлять информацию пользователю.

Model – это компонент, который отвечает за хранение данных и выполнение бизнес-логики. Этот компонент может включать базу данных, модели данных и другие компоненты, которые работают с данными приложения.

View – это компонент, который отвечает за представление данных пользователю. Этот компонент может включать HTML-шаблоны, CSS-стили и JavaScript-скрипты, которые отображают данные пользователю.

Controller – это компонент, который отвечает за управление потоком данных и управление пользовательским вводом. Этот компонент может включать маршрутизаторы, контроллеры и другие компоненты, которые управляют потоком данных между моделью и представлением.

Преимущества использования паттерна MVC включают:

Разделение логики приложения на компоненты, которые могут быть легко изменены и заменены.

Улучшение тестируемости приложения, поскольку каждый компонент может быть протестирован независимо от других компонентов.

Улучшение масштабируемости приложения, поскольку каждый компонент может быть масштабирован независимо от других компонентов.

Однако использование паттерна MVC также может привести к более сложному коду и более длительному процессу разработки, поскольку разработчики должны уделять внимание разделению логики на компоненты и управлению потоком данных между ними.

Spring Security – это мощный и гибкий фреймворк для аутентификации и авторизации в приложениях, построенных на Spring Framework. Он предоставляет ряд функций, которые позволяют разработчикам создавать безопасные приложения, защищая их от несанкционированного доступа.

Spring Security включает в себя ряд стандартных функций, таких как аутентификация пользователя, авторизация доступа, управление сессией и управление ролями пользователей. Он также обеспечивает механизмы защиты от атак CSRF (межсайтовая подделка запросов) и XSS (межсайтовые скрипты).

Spring Security обеспечивает разделение обязанностей, чтобы упростить процесс создания безопасных приложений. Он позволяет разработчикам определить правила безопасности в конфигурационном файле, а не в коде приложения. Это упрощает поддержку приложения и делает его более легким для изменения.

Spring Security также поддерживает множество способов аутентификации, включая базовую аутентификацию, аутентификацию на основе формы, аутентификацию на основе SSO (одно учетная запись для всех приложений), аутентификацию на основе токенов и многое другое.

В целом, Spring Security предоставляет мощный и гибкий набор инструментов для обеспечения безопасности приложений на Spring Framework. Он упрощает создание безопасных приложений, уменьшает вероятность ошибок и увеличивает общую безопасность системы.

Hibernate Framework – это фреймворк для языка [Java](https://blog.skillfactory.ru/java-komu-i-dlya-chego-nuzhen/), предназначенный для работы с [базами данных](https://blog.skillfactory.ru/glossary/baza-dannyh/). Он реализует [объектно-реляционную модель](https://blog.skillfactory.ru/glossary/oop-obektno-orientirovannoe-programmirovanie/) – технологию, которая «соединяет» программные сущности и соответствующие записи в базе.

Hibernate создаёт связь между таблицами в базе данных и Java-классами и наоборот. Это избавляет разработчиков от огромного количества лишней, рутинной работы, в которой крайне легко допустить ошибку и крайне трудно потом её найти.

Какие же преимущества даёт использование Hibernate:

* обеспечивает простой API для записи и получения Java-объектов в/из БД.
* минимизирует доступ к БД, используя стратегии fetching;
* не требует сервера приложения;
* позволяет нам не работать с типами данных языка SQL, а иметь дело с привычными нам типами данных Java;
* заботится о создании связей между Java-классами и таблицами БД с помощью XML-файлов не внося изменения в программный код;
* если нам необходимо изменить БД, то достаточно лишь внести изменения в XML-файлы.

Hibernate поддерживает все основные СУБД: MySQL, Oracle, PostgreSQL, Microsoft SQL Server Database, HSQL, DB2.  
Hibernate также может работать в связке с такими технологиями, как Maven и J2EE.

Hibernate поддерживает такие API, как JDBC, JNDI, JTA.  
JDBC обеспечивает простейший уровень абстракции функциональности для реляционных БД. JTA и JNDI, в свою очередь, позволяют Hibernate использовать серверы приложений J2EE.

Рассмотрим отдельно каждый из элементов Hibernate:

- transaction: этот объект представляет собой рабочую единицу работы с БД. В Hibernate транзакции обрабатываются менеджером транзакций;

- sessionFactory: самый важный и самый тяжёлый объект (обычно создаётся в единственном экземпляре, при запуске приложения). Нам необходима как минимум одна SessionFactory для каждой БД, каждый из которых конфигурируется отдельным конфигурационным файлом;

- session: сессия используется для получения физического соединения с БД. Обычно, сессия создаётся при необходимости, а после этого закрывается. Это связано с тем, что эти объекты крайне легковесны. Чтобы понять, что это такое, модно сказать, что создание, чтение, изменение и удаление объектов происходит через объект Session;

- query: этот объект использует HQL или SQL для чтения/записи данных из/в БД. Экземпляр запроса используется для связывания параметров запроса, ограничения количества результатов, которые будут возвращены и для выполнения запроса;

- configuration: этот объект используется для создания объекта SessionFactory и конфигурирует сам Hibernate с помощью конфигурационного XML-файла, который объясняет, как обрабатывать объект Session;

- criteria: используется для создания и выполнения объектно-ориентированных запроса для получения объектов.

## **Архитектурные решения**

Диаграмма компонентов позволяет определить состав программных компонентов, а также установить зависимости между ними.

Данная диаграмма обеспечивает согласованный переход от логического к физическому представлению системы в виде программных компонентов. Одни компоненты могут существовать только на этапе компиляции программного кода, другие – на этапе его исполнения. Основными элементами диаграммы являются компоненты, интерфейсы и зависимости между ними. Кроме этого, на ней могут отображаться ключевые классы, входящие в компоненты.

Компонент – это физическая часть системы. Компоненты, представляющие собой файлы с исходным кодом классов, библиотеки, исполняемые модули и т.п., которые должны обладать согласованным набором интерфейсов.

Внутри компонента, как и класса, могут быть выделены дополнительные секции, в которых указываются предоставляемые или необходимые для работы интерфейсы и классы, методы, наименование файла-компонента и т.п.

Отношение ассоциации отображается между компонентами и их интерфейсами. Отношение зависимости означает зависимость реализации одних компонентов от реализации других.

В курсовой работе между собой связаны 3 компонента: страница сайта в браузере для клиента, сервер и БД. Связь компонентов представлена на рисунке 2.1.

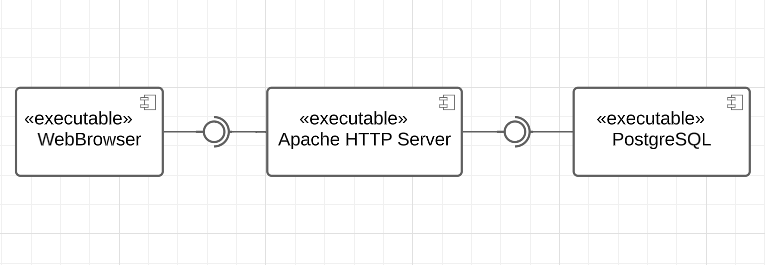


Рисунок 2.1 – Диаграмма компонентов

Второй формой физического представления программной системы является диаграмма развертывания. Она применяется для представления общей конфигурации и топологии распределенной информационной системы, содержит сведения о распределении компонентов по отдельным узлам системы и каналом связи между аппаратными средствами.

Таким образом, диаграмма развертывания предназначена для визуализации элементов и компонентов системы, существующих лишь на этапе ее исполнения, к которым относятся исполнимые файлы, динамические библиотеки, таблицы БД и т. д. Те компоненты, которые не используются на этапе исполнения, на диаграмме не показываются.

Основные цели, преследуемые при разработке диаграммы развертывания:

- распределение компонентов системы по ее физическим узлам;

- отображение физических связей между узлами системы на этапе исполнения;

- выявление узких мест системы и реконфигурация ее топологии для достижения требуемой производительности.

Элементами диаграммы развертывания являются узлы, компоненты и связи между ними.

Диаграмма развертывания программного средства представлена на рисунке 2.2.

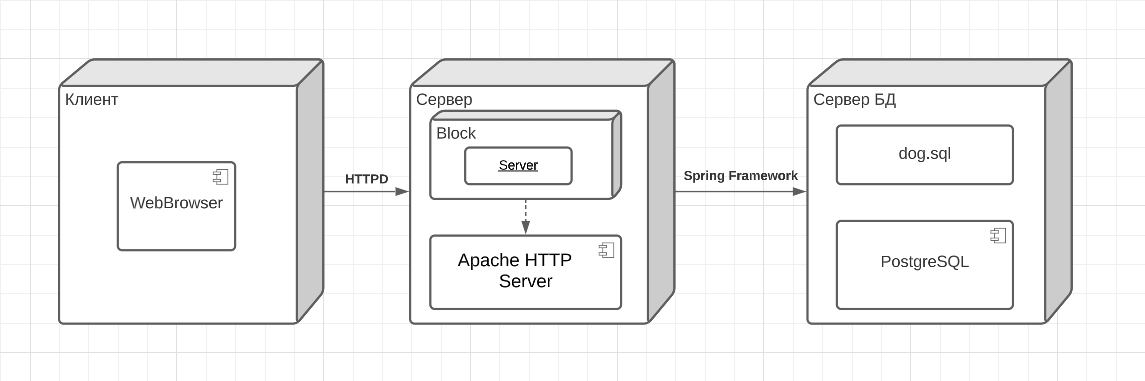


Рисунок 2.2 – Диаграмма развертывания

При составлении диаграммы классов необходимо учитывать особенности задания атрибутов, методов и отношений между классами с учетом специфики используемого языка программирования, в данном случае Java.

Графически класс отображается в виде прямоугольника, который может быть разделен горизонтальными линиями на секции. В этих секциях указывается имя, атрибуты и методы.

С точки зрения структурного подхода, атрибуты – это переменные, а методы – это функции, описанные в теле класса. Они могут быть доступны или не доступны для изменения или выполнения внешними объектами.

Обязательным элементом обозначения класса на диаграмме является его имя. Оно должно быть уникальным в пределах пакета. Если класс является абстрактным, то его имя пишется курсивом.

Отношения, которые можно устанавливать между классами:

- отношение ассоциации: означает наличие атрибута, в котором будет храниться ссылка на объект класса, в сторону которого направлена стрелка ассоциации;

- отношения агрегации и композиции являются частными случаями ассоциации;

- отношение обобщения в тексте программы на языке Java показывается ключевым словом «extends» в дочернем классе;

- отношение зависимости не приводит к автоматической генерации кода программы, но свидетельствует об обращении из объекта зависимого класса к атрибутам, методам или непосредственно к объектам независимого класса;

- отношение реализации - отображается только между классами и интерфейсами. В тексте на языке Java данное отношение обозначается ключевым словом «implements».

## **Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства**

Бизнес-логика программного средства (ПС) представляет собой набор правил и процессов, которые определяют, как ПС должно функционировать, чтобы соответствовать потребностям бизнеса и достигать поставленных целей. Бизнес-логика является основным элементом любого ПС и включает в себя описание бизнес-процессов, правил и требований, которые необходимы для решения задач бизнеса.

Бизнес-логика может быть представлена в виде диаграмм, документации, кода или других форматов. В современном ПС, бизнес-логика часто разделяется на модули, чтобы обеспечить более эффективное управление и обслуживание.

Основные функции бизнес-логики ПС включают:

- управление бизнес-процессами – определяет последовательность действий и операций, необходимых для выполнения определенной задачи.

- управление данными – определяет, как данные будут храниться, обрабатываться и анализироваться в ПС.

- управление правами доступа – определяет, кто имеет доступ к каким данным и функциям ПС.

- управление логикой приложения – определяет, как ПС будет взаимодействовать с пользователем и какие действия будут выполняться в ответ на конкретные действия пользователя.

Главная задача приложения салона красоты для животных позволить клиенту быстро и самостоятельно записываться на услуги. Вначале клиент регистрируется на сайте для записи на услугу, затем выбирает услугу, на которую ему необходимо записаться, далее выбирает мастера, дату и время, которые ему необходимы. Мастер же в свою очередь после создания записи клиентом видит в личном кабинете новую запись и принимает решение одобрить либо же отклонить запись. После одобрения записи и выполнения процедуры мастер отмечает услугу как выполненную, тогда она появляется в аналитике, которая доступна к просмотру администратору. Также администратор, контролирует учетные записи клиента и мастера, может назначить им роли и удалять.

## **2.5 Проектирование пользовательского интерфейса**

Проектирование пользовательского интерфейса является важной частью разработки программного обеспечения, сайтов и мобильных приложений. Цель проектирования пользовательского интерфейса заключается в разработке ясного и понятного представления о том, каким должен быть интерфейс системы. Этот инструмент помогает достижению бизнес-целей и созданию конкурентных преимуществ, а также предоставляет разработчикам ясные указания по созданию системы. В процессе проектирования пользовательского интерфейса необходимо учитывать следующие факторы:

1. Целевая аудитория: дизайн должен быть нацелен на определенную аудиторию пользователей. Например, дизайн интерфейса для старших людей будет отличаться от дизайна интерфейса для молодых.
2. Цели и задачи пользователя: дизайн должен помочь пользователям быстро и легко достигать своих целей. Например, если сайт продает товары, то интерфейс должен быть удобным для поиска и покупки товаров.
3. Информационная архитектура: важно определить структуру и организацию информации на странице или экране, чтобы пользователи могли быстро и легко найти нужную им информацию.
4. Навигация: дизайн должен обеспечивать удобную и понятную навигацию для пользователя, чтобы он мог легко переходить между различными разделами и функциями.
5. Визуальный дизайн: дизайн должен быть привлекательным и легко воспринимаемым пользователем. Это включает выбор цветовой палитры, типографики, иконок и других элементов дизайна.
6. Размер и расположение элементов: дизайн должен учитывать размер и расположение элементов интерфейса, чтобы пользователь мог легко их обнаружить и использовать.
7. Контекст использования: дизайн должен учитывать контекст использования приложения или сайта, чтобы пользователи могли использовать его в различных ситуациях и на различных устройствах.

В целом, проектирование пользовательского интерфейса – это искусство сбалансировать между несколькими факторами и создать удобный, привлекательный и эффективный интерфейс для пользователей.

# **ЗАКЛЮЧЕНИЕ**

# **Список использованных источников**

# **ПРИЛОЖЕНИЕ А**

# **ПРИЛОЖЕНИЕ Б**

# **ПРИЛОЖЕНИе В**