Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных технологий

Кафедра проектирования информационных компьютерных систем

Дисциплина "Современные технологии проектирования информационных систем"

К защите допустить:
Руководитель курсовой работы
старший преподаватель кафедры
_____ А.В.Михалькевич
10.07.2025

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Разработка сайта для тату салона

БГУИР КР 1-40 05 01-10 № 169 ПЗ

Студент (подпись студента)

Курсовая работа
представлена на проверку
10.07.2025
_____
(подпись студента)

2025

Реферат

БГУИР КР 1-40 05 01-10 № 169 ПЗ, гр. 784371

, Разработка сайта для тату салона, Минск: БГУИР - 2025.

Пояснительная записка 76441 с., 20 рис., 3 табл.

Ключевые слова:

Предмет Современные технологии проектирования информационных систем, А.В.Михалькевич

Содержание

Введение

Курсовой проект посвящен изучению теории и практики разработки и проектирования распределённых баз данных. В данном курсовом проекте объектом исследования является среда WEB, как платформа приложений баз данных в информационных системах. Предметом исследования является система организации и ведения распределённых баз данных в Интернет. Целью курсового проекта является разработка сайта тату салона. Курсовой проект предполагает выполнение следующих задач: – спроектировать базу данных, описать предметную область, создать диаграмму классов и диаграмму состояний; – разработать с использованием WEB-интерфейса, алгоритма работы сайта тату салона. В качестве практической части в рамках курсового проекта создаётся сайт тату салона с использованием технологий языка программирования PHP и СУБД MySQL.

1 Описание предметной области

Архитектура сайта – систематизация информации и навигации по ней с целью помочь посетителям более успешно находить нужные им данные. Хорошо продуманная грамотная архитектура сайта гарантирует, что пользователи потратят меньше времени на поиск нужной информации.

Архитектура сайта ведётся с учётом наиболее важной информации с точки зрения продвижения товаров и услуг на интернет-рынке. Грамотное распределение приоритетов

между разделами и страницами сайта, делает их основными точками входа на сайт, что позволяет потенциальному потребителю быстро найти необходимую ему информацию об искомых товарах/услугах и повышает успешность бизнеса в интернете.

Архитектура сайта проста и интуитивно удобна, состоит из клиентской части, программной части и администрирования.

Программная часть архитектуры сайта рассматривается как взаимосвязь операционной части и серверной части.

В операционной части рассматривается среда разработки сайта.

Серверная часть содержит в себе размещение на сайте провайдера, поддерживающие технологии, используемые при создании сайта.

Сайт разрабатываются в среде *PHP*, либо – *Perl, ASP.NET, ColdFusion* и *Java*.

В клиентской части архитектуры разрабатывается максимально удобная и доступная работа потенциального клиента на страницах сайта. Разработка интерфейса, доступные и понятные диалоговые окна. Немаловажным фактором является обратная связь, позволяющая высказать клиенту свое мнение о тату салоне, о качестве обслуживания и сайта в целом.

Проанализировав работу уже имеющихся сайтов, делается вывод о том, что обязательно будет реализовано в курсовом проекте.

Структура сайта оформляется так, чтобы пользователь имел возможность получать о сайте исчерпывающую информацию (описание в виде текста плюс несколько фотографий).

Для создания сайта отдаётся предпочтение язык программирования *PHP*. Это мощная среда для разработки, совместимая со всеми операционными системами и браузерами, не требующая высоких аппаратных средств компьютера, довольно проста в освоении и продолжает развиваться и совершенствоваться. Также он поддерживается подавляющим большинством платных хостингов, что является несомненным плюсом.

Выбор платного хостинга заключается в том, что есть хоть какие-то гарантии, сайт получает имя на доменном уровне, поддерживаются все современные технологии, не будет назойливых рекламных баннеров, не относящихся к тематике сайта, скорость закачки будет заметно выше, обслуживание таких сайтов удобнее, есть возможности для развития, введения новых услуг для привлечения клиентов. Также можно заключить долгосрочный договор, что будет гарантировать бесперебойную работу сайта, его защиту от взлома и вирусов, позволит избежать неприятных сюрпризов вроде прекращения существования данного хостинга.

Проведём проектирование базы данных сайта тату салона.

Проектирование базы данных состоит главным образом в определении элементов данных, которые нужно включить в базу данных, отношения между ними и ограничений на значения данных. Внешнее представление содержит только те сущности, атрибуты и связи предметной области, которые интересны пользователю.

Помимо этого, различные представления могут по-разному отображать одни и те же данные. Разработаем базу данных для автоматизации работы сайта.

Для рационального управления сайта необходимо контролировать различную поступающую информацию, которую необходимо структурировать и хранить в различных базах данных.

Имеющиеся базы данных должны быть взаимосвязаны между собой. Для правильного

создания баз данных с такой информацией необходимо определить сущности сайта.

В соответствии с заданием в курсовом проекте необходимо спроектировать базу данных сайта по информации об услугах.

2 Описание основного процесса предметной области

Объектом автоматизации является тату салон, который работает с клиентами, предоставляя услуги создания, проектирования, наложения эскиза татуировки. Основная задача работы сайта – информирование клиента о своих услугах.

Основными задачами сайта являются:

информирование клиентов в соответствии с поступающими заявками на сайте и поступающими звонками;
улучшение работы сайта на основании внедрения современных технологий и компьютеризации информационных процессов.

Основные нотации концептуального моделирования: нотация *IDEF0/IDEF3*, нотация *ARIS*, и нотация *UML*.

Нотация *IDEF0* – это нотация, применяемая для моделирования широкого класса систем. Результатом данного моделирования является модель системы, которая состоит из иерархии диаграмм, текста документации, которые связанны друг с другом при помощи перекрестных ссылок.

Нотация *ARIS* – предполагает построение большого числа диаграмм, для описания динамики и статистики. Данные диаграммы классифицируются по видам, типам, уровням и ракурсам описания.

Нотация *UML* – семейство графических нотаций, в основе которого лежит единая метамодель. Нотация *UML* помогает в описании и проектировании программных систем, в особенности систем, которые построены с применением объектно-ориентированных технологий.

Осуществим сравнение данных нотаций, и представим результаты сравнения в таблицу 2.1

Таблица 2.1 – Сравнительный анализ нотация

Критерий	**Нотация *IDEF0***	**Нотация *ARIS***	**Нотация *UML***
Легкость в изучении и понимании	Легок в освоении	Очень сложный в освоении	Сложный в освоении
Подход к проектированию	Функциональный	Процессный	Объектно-ориентированный
Области применения	Бизнес-процессы, программное обеспечение	Бизнес-процессы	Бизнес-процессы, программное обеспечение

Как видно из таблицы 2.1, наиболее подходящей является нотация *IDEF0*, поскольку нотации *ARIS* и *UML* достаточно сложны в освоении.

В настоящее время существует множество *CASE* средств, поддерживающих функциональное моделирование в стандарте *IDEF0*. Однако наиболее популярной, и легкой в понимании является *AllFusion Process Modeler (BPwin).*

AllFusion Process Modeler – это мощный инструмент моделирования, который создала фирма *Computer Associates Technologies*, и который применяется для анализа, документирования и реорганизации сложных бизнес-процессов.

Для построения контекстной модели, необходимо определить входные, выходные данные, управляющая информация и механизм.

В данном случае, входной информацией будет: желание.

Выходной информацией будет сделанная татуировка.

Управляющей информацией будет: санитарные нормы, сертификат мастера.

Механизмом управления будет клиент, мастер.

Построенная контекстная диаграмма показана на рисунке 2.1.

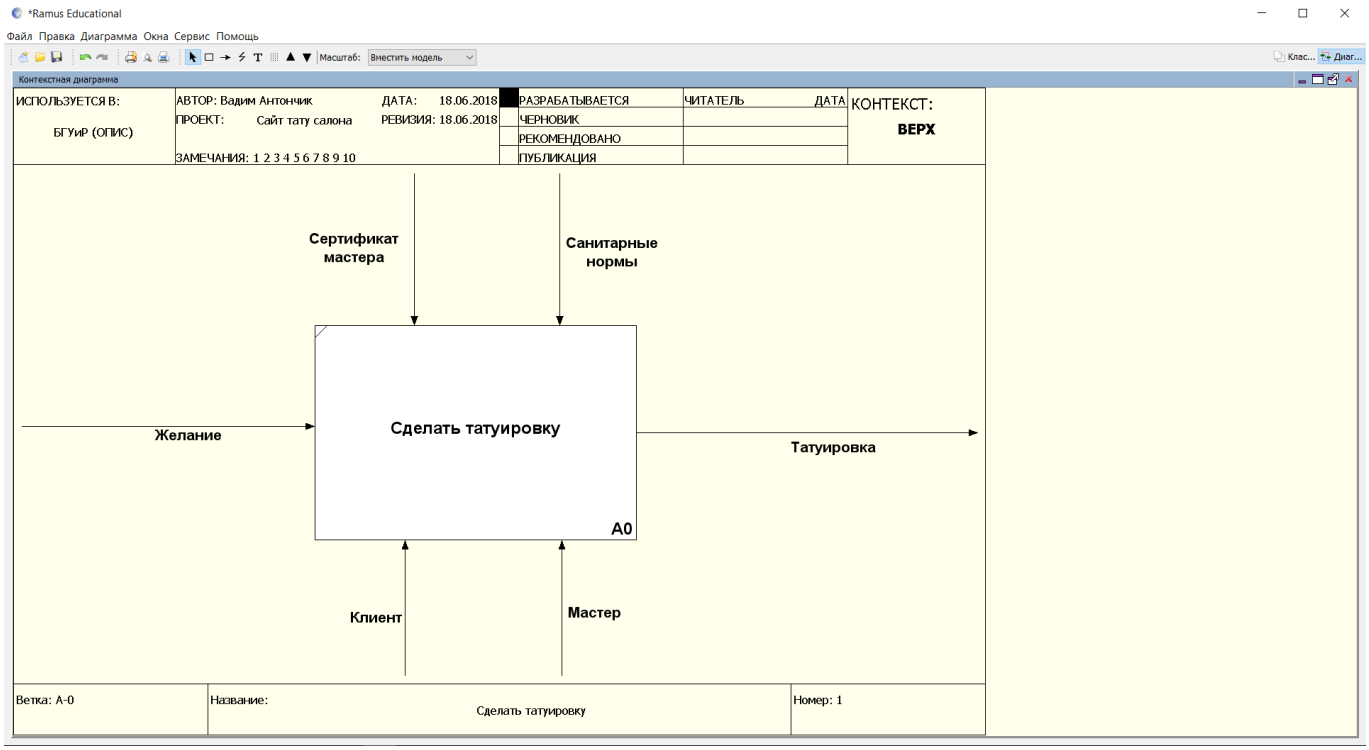

Рисунок 2.1 – Контекстная диаграмма

Декомпозируем контекстную диаграмму – рисунок 2.2.

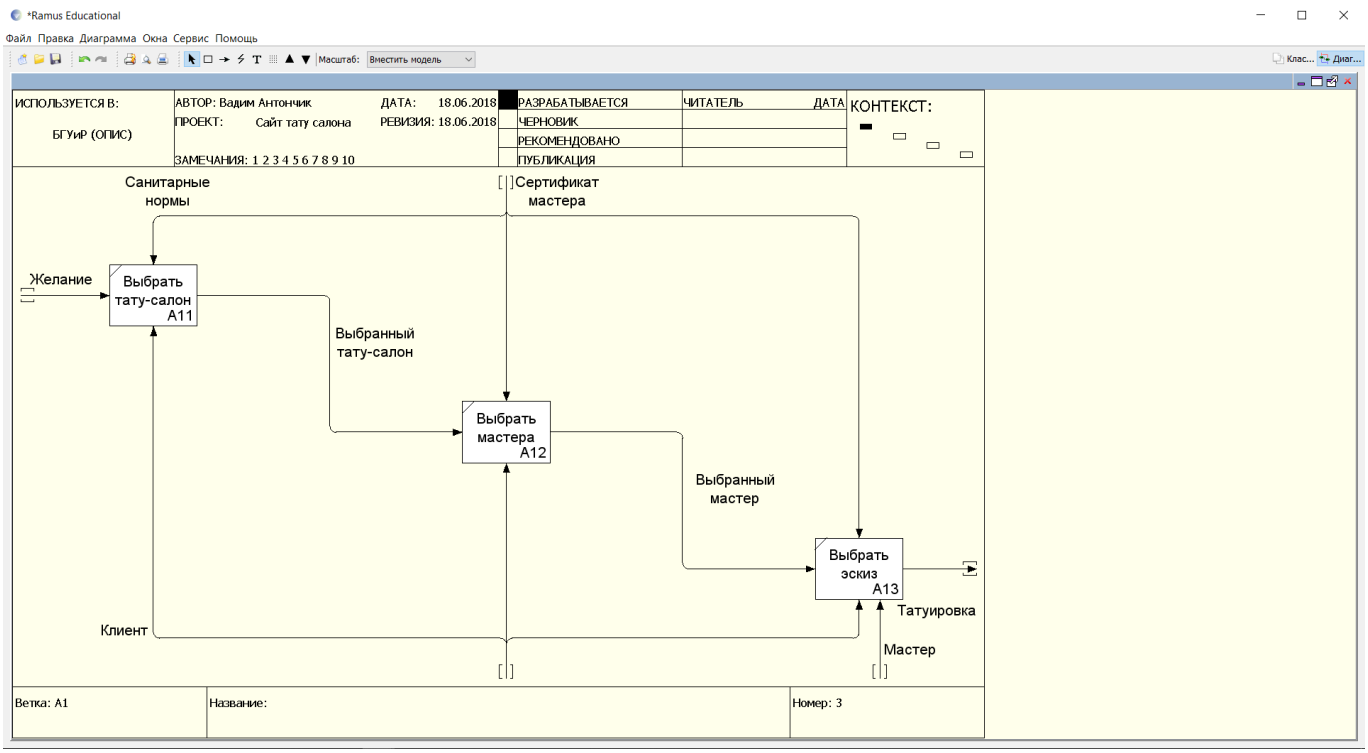

Рисунок 2.2 – Диаграмма декомпозиции контекстной диаграммы

Как видно из рисунка 2.2, диаграмма декомпозиции состоит из трех процессов: выбрать тату-салон; выбрать мастера; выбрать эскиз.

Процесс деятельности тату салона обладает следующими недостатками:

оформление всей документации вручную, что занимает достаточное большое время;
возможность допущения ошибок;
поиск эскиза, также занимает большое количество времени, поскольку необходимо пересмотреть все имеющиеся эскизы (по критериям);

После построения модели, можно сформулировать требования к новой ИСУ.

Функциональные требования:

оформление и редактирование данных о клиенте;
редактирование эскизов;
получение информации о клиенте;

Нефункциональные требования:

интерфейс программы должен быть простым и легким в освоении;
восстановление ИСУ после сбоя – не более 12 часов;
ИСУ должна иметь многопользовательский режим работы;
отклик ИСУ должен составлять не более 10 секунд.

3 Спецификация вариантов использования системы

Спецификация – документ, который точно, полностью и в поддающейся проверке форме определяет требования, устройство, поведение или другие особенности системы, компонента, продукта, результата или услуги, а также процедуры, способные определить, были ли выполнены эти условия.

Спецификация может содержать:

описательное название, номер или другой идентификатор спецификации;
время последнего пересмотра и отметку, кем был выполнен пересмотр;
логотип или торговую марку, указывающую, кому принадлежит право на копирование, владельца и происхождение документа;
содержание документа, если документ длинный;
ответственное лицо или организацию по вопросам по спецификации, по обновлениям и отклонениям;
важность, область применения спецификации и её назначение;
термины, определения и аббревиатуры для пояснения сути спецификации;
способы проверки для всех установленных требований и характеристик;
материальные требования: физические, механические, электрические, химические и другие. Целевые и допустимые;
требования по эксплуатационному тестированию. Целевые и допустимые;
изображения, фотографии или технические иллюстрации;
требования по мастерству;
требования к сертифицированности;
требования по технике безопасности;
экологические требования;
контроль по обеспечению качества, образец для проверки, проверка, критерий приёма работы;
лицо или организация ответственное за выполнение спецификации;
выполнение и доставка;
условия по отклонениям, перепроверке, пересмотре, корректировке измерений и характеристик;
ссылки и цитаты в тексте спецификации, которые могут потребоваться для установки ясности документа;
подписи и разрешения, если они необходимы;
контроль изменений (с помощью специальных компьютерных программ) для последовательной разработки, проверки и выполнения, если документ предназначен для внутреннего использования;
приложения, которые раскрывают детали, добавляют ясности, или пояснения по оплате.

Сценарий использования, вариант использования, прецедент использования (англ. *use case*) – в разработке программного обеспечения и системном проектировании это описание поведения системы, когда она взаимодействует с кем-то (или чем-то) из внешней среды. Система может отвечать на внешние запросы Актёра (англ. *actor*) (может применяться термин Актант), может сама выступать инициатором взаимодействия. Другими словами, сценарий использования описывает, «кто» и «что» может сделать с рассматриваемой системой, или что система может сделать с «кем» или «чем». Методика сценариев использования применяется для выявления требований к поведению системы, известных также как пользовательские и функциональные требования.

В системном проектировании сценарии использования применяются на более высоком уровне, чем при разработке программного обеспечения, часто представляя цели заинтересованных лиц или миссии. На стадии анализа требований сценарии использования могут быть преобразованы в ряд детальных требований и задокументированы с помощью диаграмм требований *SysML* или других подобных механизмов.

Диаграмма прецедентов (диаграмма вариантов использования) в *UML* – диаграмма, отражающая отношения между актёрами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Прецедент – возможность моделируемой системы (часть её функциональности), благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе.

Диаграмма вариантов использования представлена на рисунке 3.1.

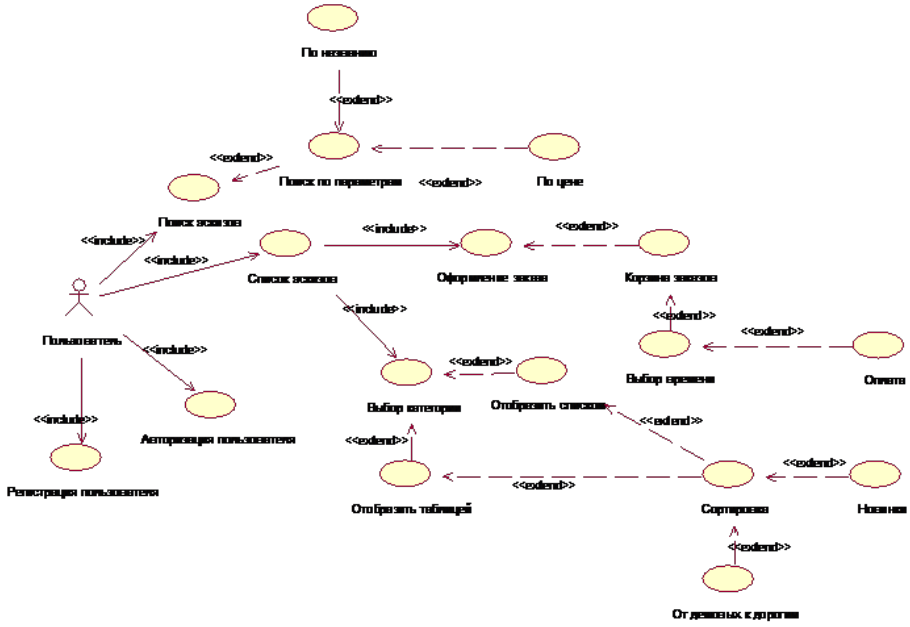

Рисунок 3.1 – Диаграмма вариантов использования

Разработка веб-приложения «Сайт тату салона» включает в себя следующий функционал:

применение фильтров к эскизам по категориям;
сортировка эскиза;
выбор категории эскиза;
просмотр фото-галереи эскизов;
добавление эскиза в корзину;
бронь времени;
оплата.

4 Информационная модель системы и её описание

Созданная база данных состоит из одной таблицы. Структура базы данных отображена на рисунке 4.1. Описание таблиц базы данных приведено в таблице 4.1.

Таблица 4.1 – Состав таблиц базы данных

Имя таблицы	Описание
1 *posts*	Справочник эскизов

В таблице 4.2 приведено описание состава таблицы спроектированной базы данных.

Таблица 4.2 – Справочник эскизов

Ключ (Да/Нет)	Поле	Тип	Значение по умолчанию	Пустые значения (Да/Нет)
PK	id	число	Нет	Нет
Нет	name	строка	Нет	Нет
Нет	text	строка	Нет	Нет

В результате была спроектирована база данных эскизов для ведения сайта тату-салона. База данных включает в себя информацию о эскизах.

5 Обоснование выбора компонентов и технологий для реализации курсового проекта

Наиболее распространенным языком разработки сайта является Язык разметки гипертекстовых страниц (*HTML – Hypertext Markup Language*) представляет собой язык, разработанный специально для создания *Web*-документов. Он определяет синтаксис и размещение специальных инструкций (тегов), которые не выводятся на экран, но указывают браузеру, как отображать содержимое документа. Он также используется для создания ссылок на другие документы, локальные или сетевые, например, находящиеся в сети Интернет.

Стандарт *HTML* и другие стандарты для *Web* разработаны под руководством консорциума *W3C* (*World Wide Web Consortium).* Стандарты, спецификации и проекты новых предложений можно найти на сайте *http://www.3w.org/.* В настоящее время действует спецификация *HTML5.0,* поддержка которой со стороны основных браузеров постоянно растет.

На практике на стандарт *HTML* большое влияние оказывает наличие тегов, предложенных и поддерживаемых наиболее известными браузерами, такими как *Microsoft Internet Explorer* и *Netscape Navigator.* Эти теги в данный момент могут, как входить, так и не входить в состав действующей спецификации *HTML*.

PHP – язык программирования, используемый на стороне *Web*-сервера для динамической генерации *HTML*-страниц. Об этом говорит и расшифровка его названия: *PHP – Personal HyperText Processor*.

PHP – один из немногих языков программирования, созданных специально для разработки *Web*-приложений. Поэтому он включает в себя все функции, необходимые именно для работы на *Web*-сервере, и при этом лишен избыточности, свойственной многим его конкурентам.

Очень приятная особенность *PHP* – то, что его команды включаются в обычные *HTML*-страницы с помощью специальных тегов, которые и заставляют *PHP*-машину выполнять на сервере нужные действия. Программам на *PHP* не нужны специальные *CGI*-директории с особыми правами доступа. Более того, на одной страничке можно произвольно чередовать «простой» *HTML* и *PHP*-код.

PHP не зависит от платформы. *PHP* прекрасно интегрируется во все популярные *Web*-серверы: *Apacce* и *IIS*, *Zens* и *Netscape Enterprise Server*, работает под *Windows* и *OS/2*, *MacOS*

и практически всеми *UNIX*-подобными системами. Как следствие – *PHP* работает практически у всех хостеров, разрешающих собственные выполняемые скрипты.

Замечательная особенность *PHP* – его интегрированность практически со всеми современными интернет-технологиями. *PHP* поддерживает большинство современных *Web*-протоколов: *IMAP, FTP, POP, XML, SNMP* и другие. *PHP* прекрасно работает с базами данных. Трудно найти СУБД, поддержка которой не была бы реализована в *PHP*. *MySQL* и *MS SQL Server*, *PostgreSQL* и *Oracle*, *Sybase* и *Interbase*. Один только перечень баз данных, поддерживаемых *PHP*, займет, наверное, целый экран.

PHP включает в себя огромное количество встроенных функций: обработки строк и массивов, работы с файловой системой и с *HTTP*, электронной почтой, датой и временем, кириллицей и другими национальными алфавитами, и большим количеством встроенных функций. Благодаря им многие алгоритмы, требующие в большинстве языков написания программного кода размером в несколько экранов, реализуются на *PHP* одной командой (точнее, вызовом одной функции).

Современные тенденции развития языков программирования не обошли стороной и *PHP*. Средства объектно-ориентированного программирования появились еще в *PHP3*. А в объектной модели *PHP4* в полном объеме реализованы классические понятия объектно-ориентированного программирования: наследование, инкапсуляция и полиморфизм.

Основное отличие от *CGI*-скриптов, написанных на других языках, типа *Perl* или *C* – это то, что в *CGI*-программах вы сами пишете выводимый *HTML*-код, а, используя *PHP* – вы встраиваете свою программу в готовую *HTML*-страницу, используя открывающий и закрывающий теги.

Отличие *PHP* от *JavaScript*, состоит в том, что *PHP*-скрипт выполняется на сервере, а клиенту передается результат работы, тогда как в *JavaScript*-код полностью передается на клиентскую машину и только там выполняется.

Диаграмма последовательности (англ. *sequence diagram*) – диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определённого объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие актёров (действующих лиц) ИС в рамках какого-либо определённого прецедента (отправка запросов и получение ответов). Используется в языке *UML*.

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни» (англ. *lifeline*), отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

Диаграмма последовательности представлена на рисунке 5.1.

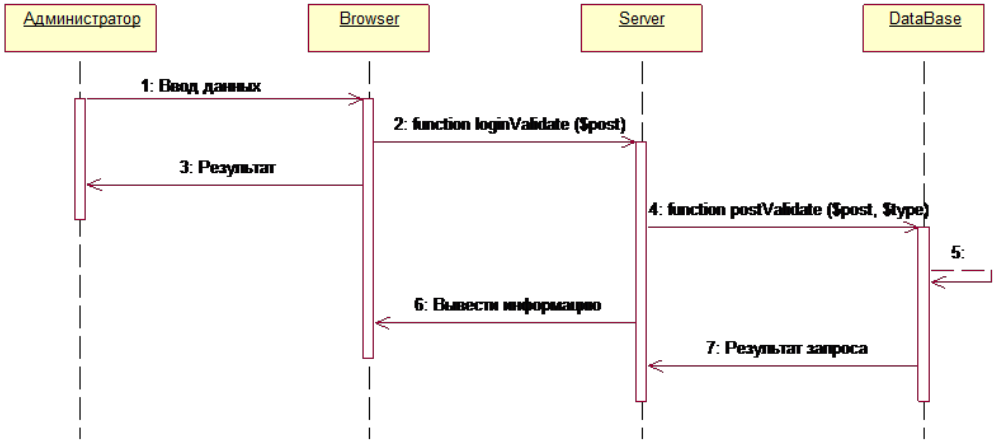

Рисунок 5.1 – Диаграмма последовательности

Диаграмма компонентов (англ. *Component diagram*) – элемент языка моделирования *UML*, статическая структурная диаграмма, которая показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т. п.

Диаграмма компонентов представлена на рисунке 5.2.

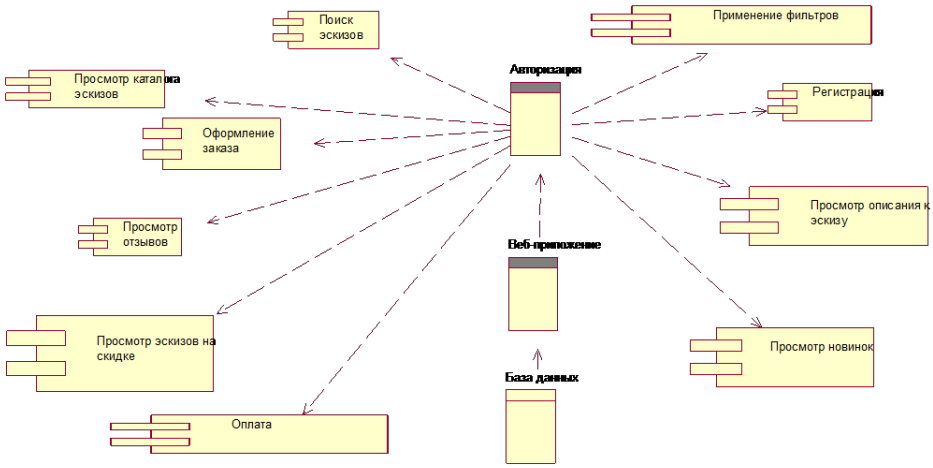

Рисунок 5.2 – Диаграмма компонентов

Диаграмма развёртывания, *Deployment diagram* в *UML* моделирует физическое развертывание артефактов на узлах. Например, чтобы описать веб-сайт диаграмма развертывания должна показывать, какие аппаратные компоненты («узлы») существуют (например, веб-сервер, сервер базы данных, сервер приложения), какие программные компоненты («артефакты») работают на каждом узле (например, веб-приложение, база данных), и как различные части этого комплекса соединяются друг с другом (например, *JDBC*, *REST*, *RMI*).

Диаграмма развертывания представлена на рисунке 5.3.

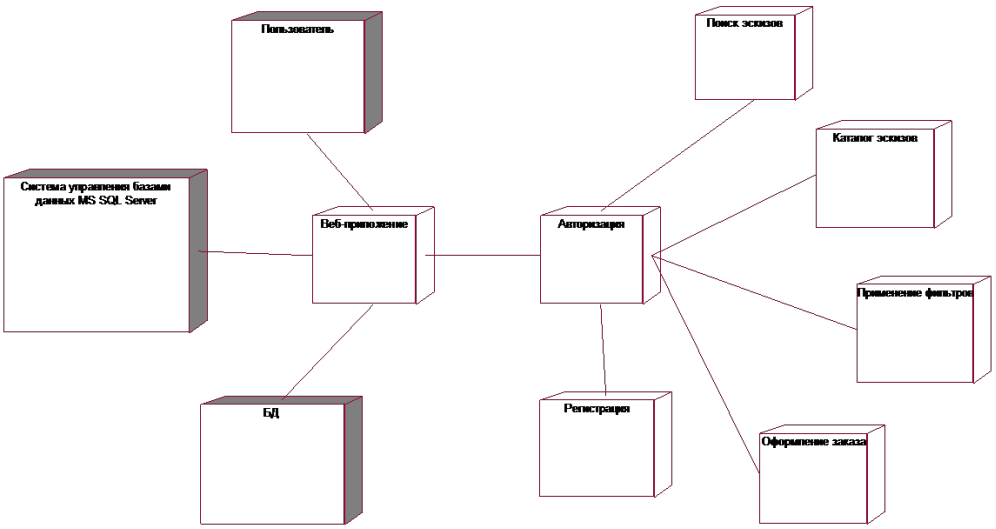

Рисунок 5.3 – Диаграмма развертывания

6 Модели представления системы и их описание

В данном разделе будет продемонстрировано моделирование системы с помощью стандарта UML, который использует графические обозначения для создания абстрактной модели системы и предназначен для определения, визуализации, проектирования и документирования в основном программных систем. UML позволяет описать систему практически со всех возможных точек зрения и разные аспекты поведения системы.

Для данной курсового проекта были построены такие диаграммы, как диаграммы вариантов использования, последовательности, состояний, классов, развертывания и компонентов.

Диаграмма вариантов использования состоит из актеров, для которых система производит действие и собственно действия *Use Case*, которое описывает то, что актер хочет получить от системы.

Диаграмма состояний предназначена для отображения состояний объектов системы, имеющих сложную модель поведения. Она показывает пространство состояний системы или ее элементов, события, которые влекут переход из одного состояния в другое, действия, которые происходят при изменении состояния. Объекты меняют своё состояние в ответ на происходящие события и стечением времени. Диаграмма состояний представляет состояния объекта и переходы между ними, а также начальное и конечное состояние объекта. Диаграмма состояний представлена на рисунке 6.1.

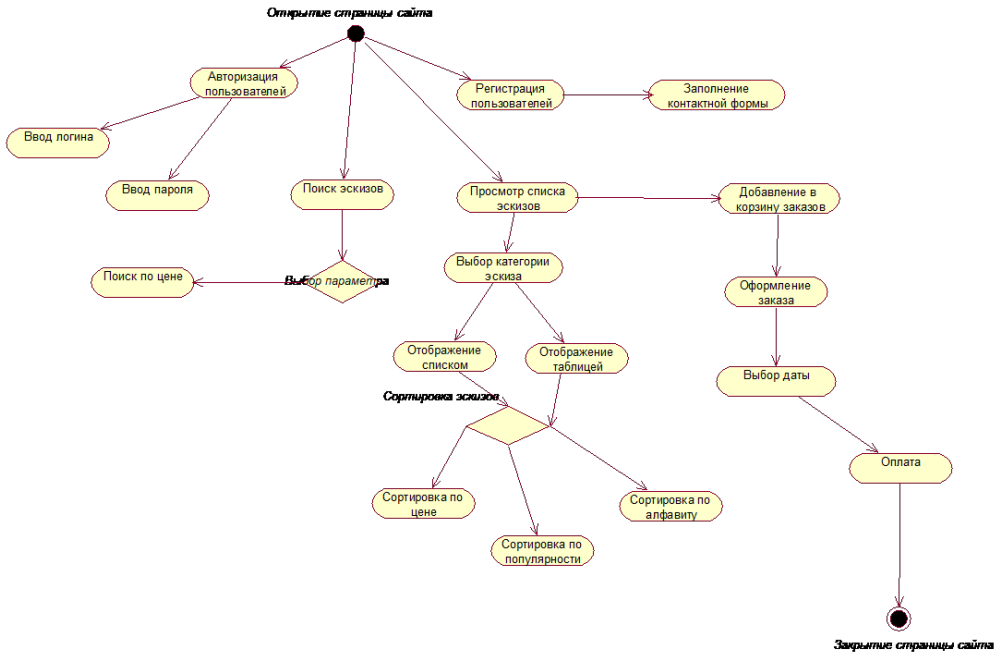

Рисунок 6.1 – Диаграмма состояний

Для моделирования взаимодействия объектов во времени в языке *UML* используются диаграммы последовательностей. Диаграмма последовательно- сти представлена в приложении Б.

Диаграмма классов описывает структуру системы, показывая её классы, их атрибуты и операторы, а также взаимосвязи этих классов. Диаграмма классов представлена в приложении Б.

Диаграмма компонентов показывает разбиение программной системы на структурные компоненты и связи между компонентами. Диаграмма компонентов представлена в приложении Б.

Диаграмма развёртывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения. Диаграмма развёртывания представлена в приложении Б.

Перед программной реализацией необходимо определиться с содержимым сайта. Информация, которая должна представляться на странице, должна удовлетворять следующим критериям:

соответствовать целям разработки сайта;
учитывать особенности целевого сегмента потребителей;
быть уникальной, чтобы привлечь внимание посетителей;
быть оперативной и достоверной, то есть информация на сайте постоянно должна обновляться.

Разработка графического дизайна интернет-каталога сайта:

Первым этапом разработки сайта является разработка дизайна. При разработке дизайна необходимо решить некоторые задачи, а именно соответствие сайта стилю предприятия, использование логотипа и цветов предприятия, а также удобство сайта для посетителей.

Графическое оформление сайта подразумевает выбор цветового оформления, создание статических и динамических элементов, подбор шрифтов и подбор фона.

Оформление для интернет-каталога сайта было выбрано стандартное, то есть в данное оформление не включается разработка оригинальных графических элементов, а используются оригиналы графических элементов, представленные предприятием и подбор подходящего шаблона.

Разрабатываемый дизайн будет соответствовать следующим требованиям:

поскольку интернет-каталог сайт несет информационную нагрузку, то графическое оформление должно быть легким, и не раздражающим глаза;
цвета, шрифты и графика должны быть выдержаны в едином стиле для всех станиц сайта. В дизайне использоваться цвета: голубой – серый – белый;
графика должна быть качественной и сочетаться с остальными составляющими страницы;
текст должен легко читаться и не сливаться с фоном.

Моделирование и разработка интернет-каталога сайта:

Этап разработки структуры имеет особое значение, поскольку от него зависит удобство пользования сайтом. На данном этапе разрабатывается документ, который служит исходным материалом для создания сайта: разработки сценария, графической концепции и структуры, программных инструментов, обеспечивающих необходимые функциональные ресурсы, и так далее.

Структура разрабатываемого сайта будет довольно простой и будет иметь все элементы навигации для удобного перемещения пользователя по сайту, структура разрабатываемого сайта приведена на рисунке 6.2.

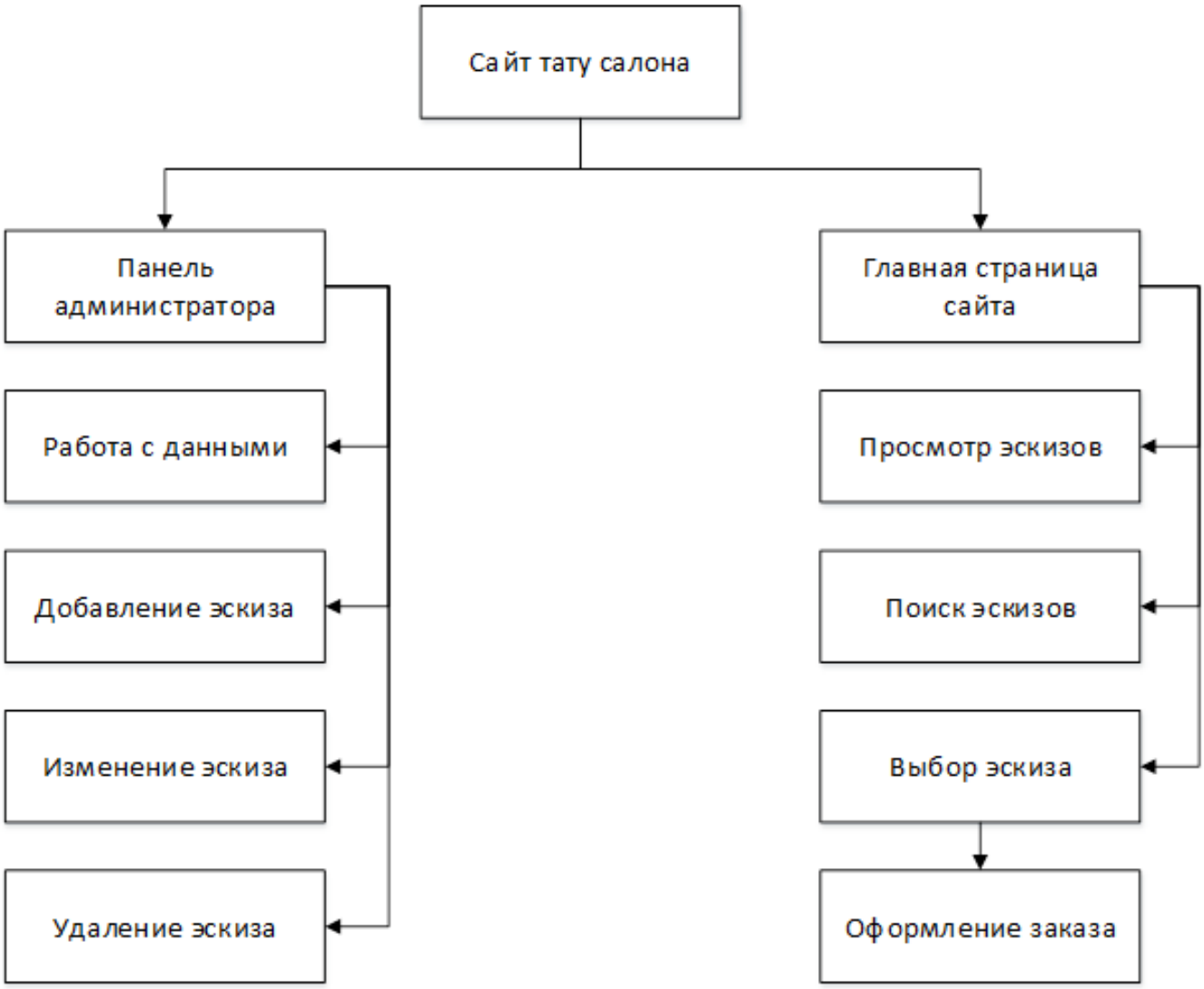

Рисунок 6.2 – Структура сайта

Блок-схема алгоритма – графическое изображение алгоритма в виде связанных между собой с помощью стрелок (линий перехода) и блоков – графических символов, каждый из которых соответствует одному шагу алгоритма. Внутри блока дается описание соответствующего действия.

Блок «процесс» применяется для обозначения действия или последовательности действий, изменяющих значение, форму представления или размещения данных. Для улучшения наглядности схемы несколько отдельных блоков обработки можно объединять в один блок. Представление отдельных операций достаточно свободно.

Блок «решение» используется для обозначения переходов управления по условию. В каждом блоке «решение» должны быть указаны вопрос, условие или сравнение, которые он определяет.

Блок «модификация» используется для организации циклических конструкций. (Слово «модификация» означает «видоизменение, преобразование»). Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.

Блок «предопределенный процесс» используется для указания обращений к вспомогательным алгоритмам, существующим автономно в виде некоторых самостоятельных модулей, и для обращений к библиотечным подпрограммам.

Блок-схема алгоритма веб-приложения представлена на рисунке 6.3.

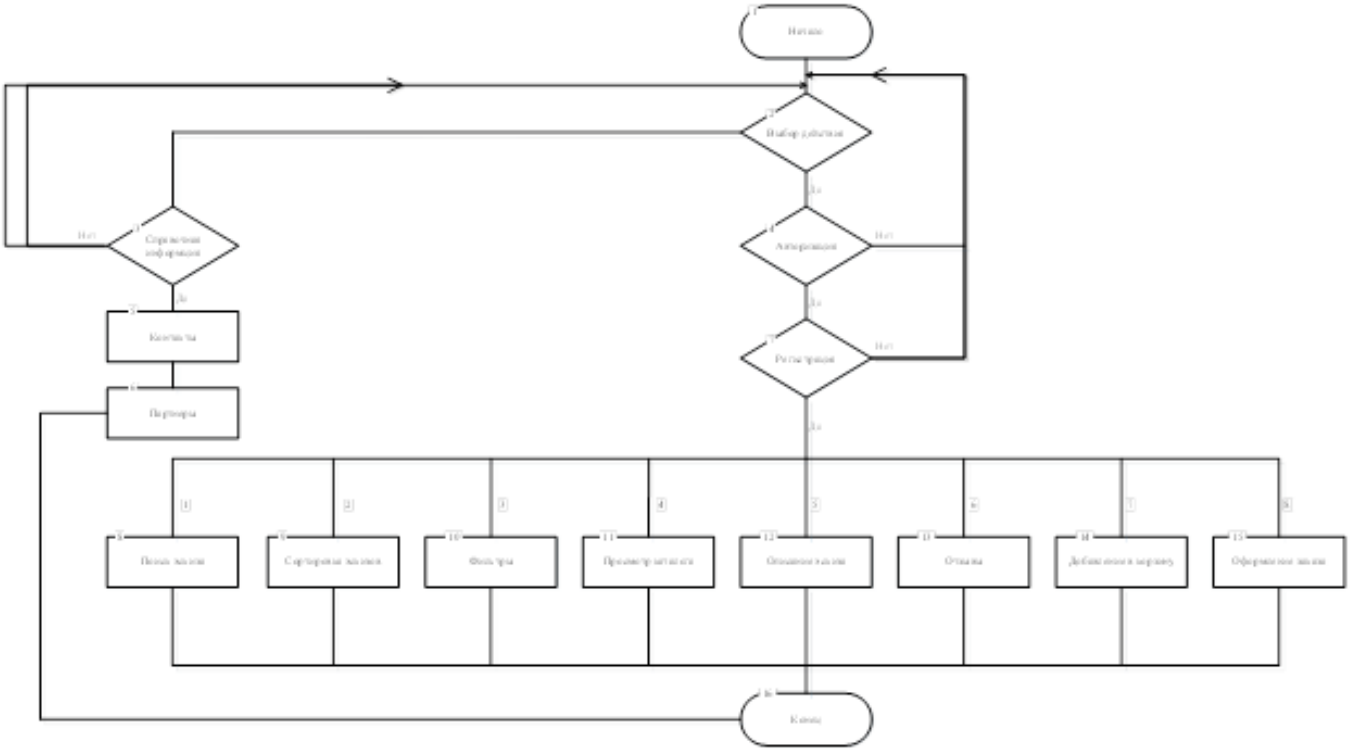

Рисунок 6.3 – Блок-схема

В фазе анализа и проектирования системы бизнес-логика воплощается в классах и методах классов, в случае использования объектно-ориентированных языков программирования, или процедур и функций, в случае применения процедурных языков.

В многоуровневых информационных системах этот уровень взаимодействует с нижележащим уровнем инфраструктурных сервисов (*Infrastructure Layer*), например, интерфейсом к базе данных или файловой системе (*Data-Access Layer, DAL*) и вышележащим уровнем сервисов приложения (*Application Services Layer*), который уже, в свою очередь, взаимодействует с уровнем пользовательского интерфейса (*User Interface Layer*) или внешними системами.

7 Описание применения паттернов проектирования

Паттерн проектирования – это часто встречающееся решение определённой проблемы при проектировании архитектуры программ.

В отличие от готовых функций или библиотек, паттерн нельзя просто взять и скопировать в программу. Паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы, которую нужно будет ещё подстроить под нужды вашей программы.

Паттерны часто путают с алгоритмами, ведь оба понятия описывают типовые решения каких-то известных проблем. Но если алгоритм – это чёткий набор действий, то паттерн – это

высокоуровневое описание решения, реализация которого может отличаться в двух разных программах.

Если привести аналогии, то алгоритм – это кулинарный рецепт с чёткими шагами, а паттерн – инженерный чертёж, на котором нарисовано решение, но не конкретные шаги его реализации.

Описания паттернов обычно очень формальны и чаще всего состоят из таких пунктов:

проблема, которую решает паттерн;
мотивации к решению проблемы способом, который предлагает паттерн;
структуры классов, составляющих решение;
примера на одном из языков программирования;
особенностей реализации в различных контекстах;
связей с другими паттернами.

Такой формализм в описании позволил создать обширный каталог паттернов, проверив каждый из них на состоятельность.

Диаграмма классов (англ. *Static Structure diagram*) – диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними. Входит в *UML*.

Существует два вида:

статический вид диаграммы рассматривает логические взаимосвязи классов между собой;
аналитический вид диаграммы рассматривает общий вид и взаимосвязи классов, входящих в систему.

Существуют разные точки зрения на построение диаграмм классов в зависимости от целей их применения:

концептуальная точка зрения – диаграмма классов описывает модель предметной области, в ней присутствуют только классы прикладных объектов;
точка зрения спецификации – диаграмма классов применяется при проектировании информационных систем;
точка зрения реализации – диаграмма классов содержит классы, используемые непосредственно в программном коде (при использовании объектно-ориентированных языков программирования).

Диаграмма классов является ключевым элементом в объектно-ориентированном моделировании. На диаграмме классы представлены в рамках, содержащих три компонента:

в верхней части написано имя класса. Имя класса выравнивается по центру и пишется полужирным шрифтом. Имена классов начинаются с заглавной буквы. Если класс абстрактный – то его имя пишется полужирным курсивом.
посередине располагаются поля (атрибуты) класса. Они выровнены по левому краю и начинаются с маленькой буквы.
нижняя часть содержит методы класса. Они также выровнены по левому краю и пишутся с маленькой буквы.

Диаграмма классов представлена на рисунке 7.1.

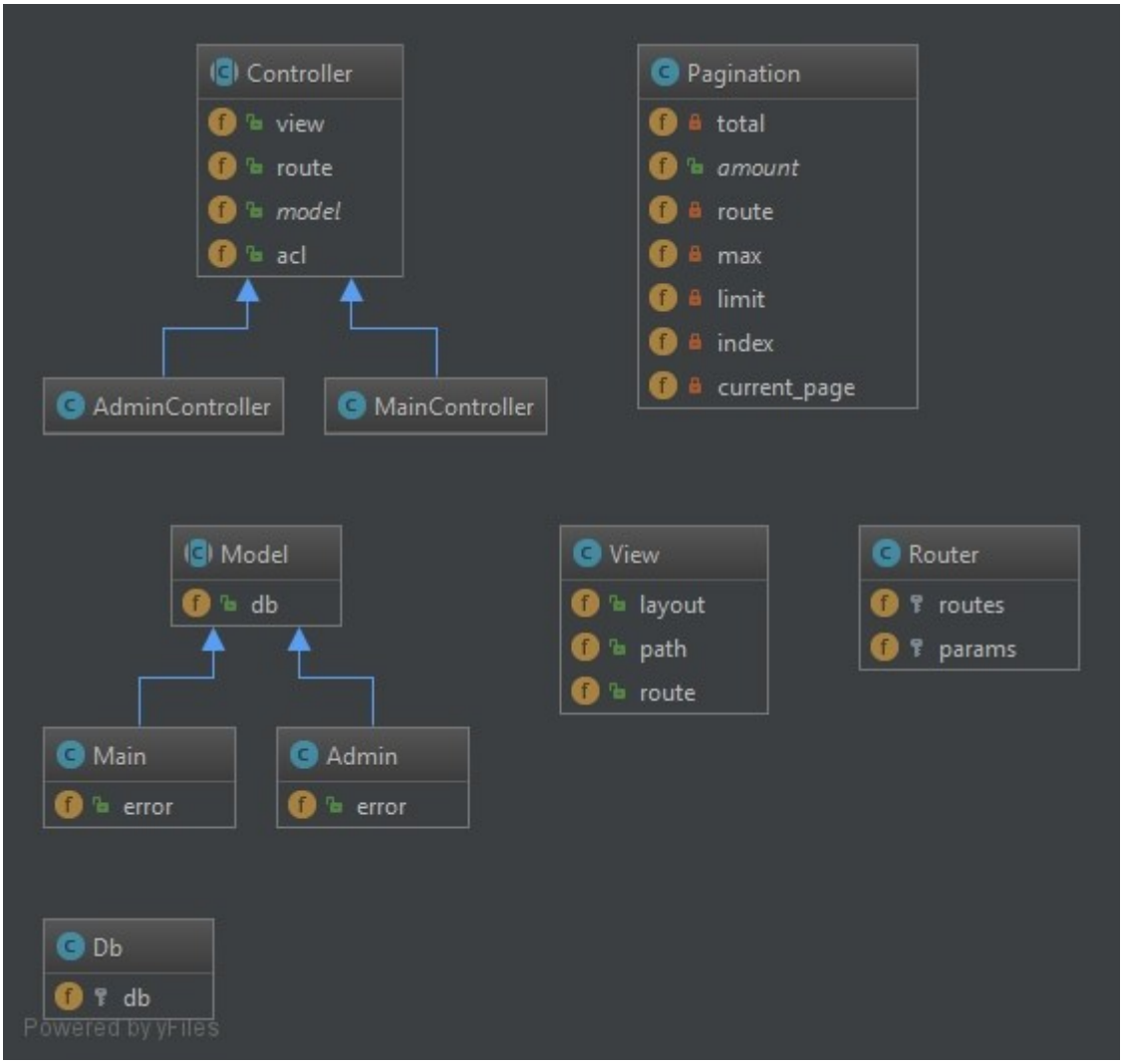

Рисунок 7.1 – Диаграмма классов

На рисунке 7.1 представлена диаграмма классов сайта тату салона.

Основной паттерн для создания фреймворков и других web-приложений. Фреймворки в *PHP* зачастую используют для больших проектов. Основное преимущество - это, конечно же, предоставление возможности строить проект при помощи паттерна *MVC* (*Model-View-Controller*).

Плюсы:

вложенность шаблонов
независимость представления от контроллера
целостность шаблона
возможность кэширования
видимость переменных
лаконичность кода

Расшифруем само понятие *MVC*:

Model - модели данных, которые многие и без того используют без фреймфорков. Фактически обычные классы для работы с разными данными.

View - представления. Это шаблонизатор, например, *SMARTY* либо собственный. Представления - это вид, в котором отображаются данные.

Controller – основной вызываемый класс, содержащий базовую логику приложения.

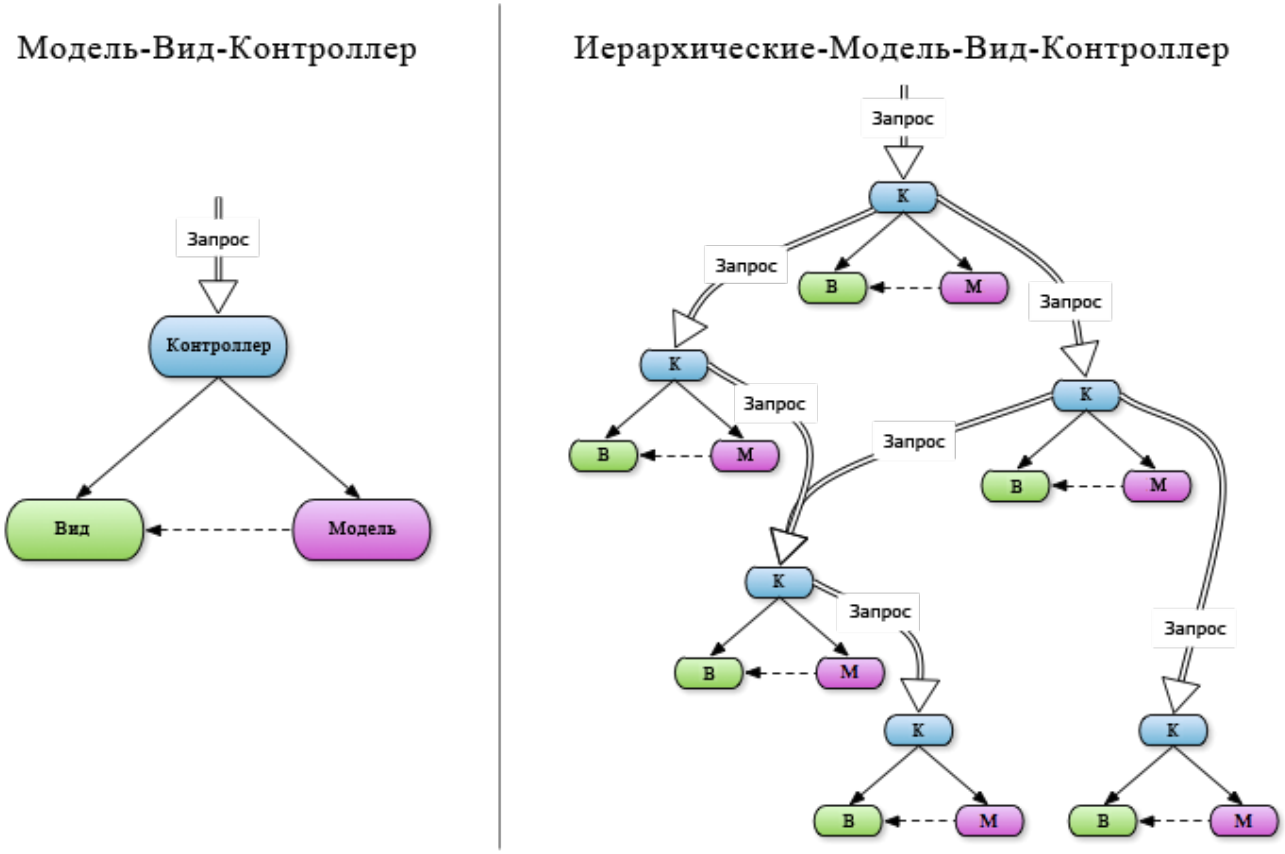

Рисунок 7.2 – Модели

Для понимания модели *HMVC* необходимо также иметь представление о роутинге (или маршрутизации) запросов. Главное отличие от *MVC*-паттерна: возможность передачи запроса по контроллерам.

По такому принципу построены почти все современные web-фреймворки (за исключением клиентских, таких как *Angular*, *Backbone* и др.)

В *HMVC* фактически процесс не меняется, т.к. последовательность действий в случае использования фреймворка остается той же, что и без него (принимаем данные - обрабатываем их в модели - выводим результат через представление).

HMVC позволяет легко соби рать воедино и легко управлять большими частями кода. А фреймворк вносит существенную долю автоматизации и простоты управления. Фреймворк - это склад различных классов и библиотек, которые позволяют отказаться от изобретения велосипедов и начать использовать готовые решения, тем самым увеличив скорость разработки. Любой разработчик, если он занимается профессиональной разработкой, со временем приходит к созданию собственной библиотеке классов, основанной, как правило, на уже готовых классах.

Современные фрэймворки не только предлагают для использования готовые классы, но и свою структуру папок.

У каждого из фрэймворков есть свои преимущества и свои недостатки. По концепции *MVC*, когда мы делаем запрос, мы, сперва, попадаем в контроллер (*Controller*). Затем в контроллере может происходить вызов модели (*Model*) (т.е. получение данных из модели), а затем передача этих данных в шаблон представления (*View*). Все очень просто, но это не всегда бывает удобно,

хотя бы потому, что часто приходится вносить изменения в контроллеры либо дублировать контроллеры из-за того, что в них вносятся незначительные изменения. В связи с этим придумали концепцию *HMVC*, т.е. иерархическая *MVC*. По данной концепции мы также сперва делаем запрос к контроллеру, который в свою очередь может передать запрос к другому контроллеру. Взаимосвязь самого контроллера с моделью и шаблоном представления осталась той же. Концепцию *HMVC* помогают понять следующие технологии:

наследование классов;
использование переменных-шаблонов.

Запрос из адресной строки попадает в так называемый обработчик маршрутов, или маршрутизатор, или роутер (*routes*). Маршрутизатор определяет, какой контроллер необходимо вызывать. Маршруты находятся в папке *routes*.

8 Система контроля версий

Git- мощная и сложная распределенная система контроля версий.

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

СКВ даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Вообще, если, пользуясь СКВ, вы всё испортите или потеряете файлы, всё можно будет легко восстановить.

TortoiseGit — визуальный клиент системы управления исходными кодами программ *git* для ОС *Microsoft Windows*. Распространяется по лицензии *GNU GPL*.

Реализован как расширение проводника *Windows* (*shell extension*). Подрисовывает иконки к файлам, находящимся под управлением *Git*, для отображения их статуса в *Git*.

Взаимодействие с системой контроля версий основано на *mSysGit*, *TortoiseGit* использует его внутри себя, и требует установки последнего на машину.

Предназначен для удобства работы: по существу, все операции с репозиторием *Git* можно выполнять из графического интерфейса *TortoiseGit,* без использования консольных команд.

Зарегистрировались на *GitHub.com*.

Прописали следующие команды в консоли:

Git init – создание репозитория;
Git add * - добавляет содержание рабочей директории в индекс (*stagingarea*) для последующего изменения;

Git commit –m “сообщение”- изменение последнего изменения;

Git remote add project https://github.com/bluntantoff/tattoo

добавление удаленного репозитория;
Git push project master - вносим изменения в удаленный репозиторий.

Ссылка на удаленный репозиторий:

https://github.com/bluntantoff/tattoo.git

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

Также приведена ссылка на удаленный репозиторий, по которой можно найти данный проект.

9 Руководство по развёртыванию системы

Для начала необходимо запустить локальный сервер, после перейти на вкладку «*MySOL*» – «*Admin*». Перед программистом в окне браузера откроется страница с авторизацией (рисунок 9.1).

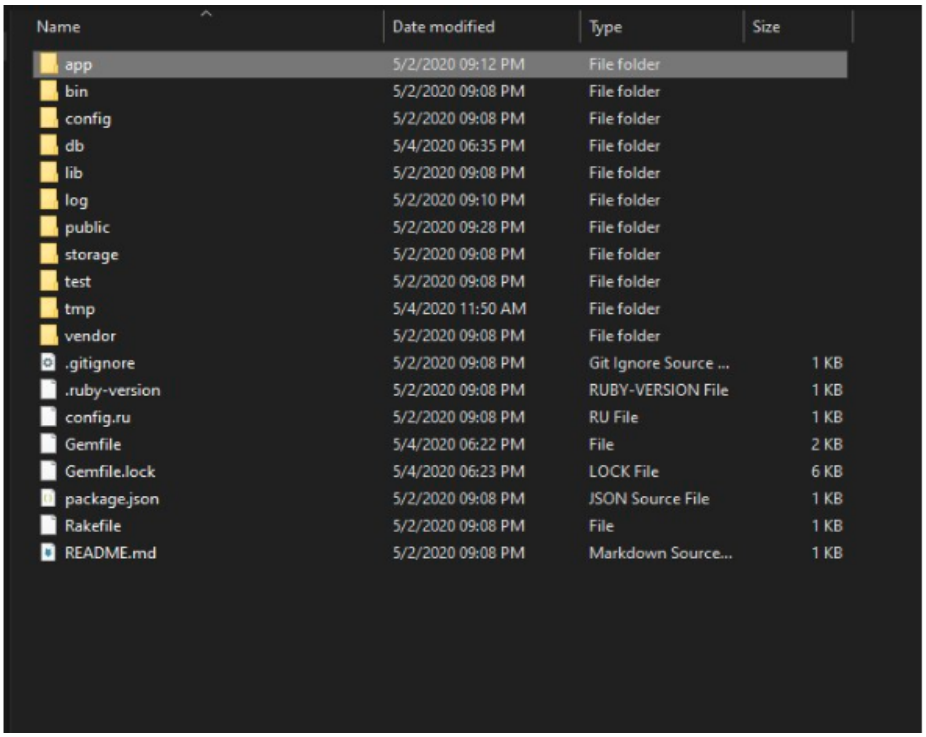

Рисунок 9.1 – Авторизация PHPMyAdmin

Далее система перенаправит программиста на главную страницу СУБД для дальнейшего импорта спроектированной базы данных. Главная страница представлена на рисунке 9.2.

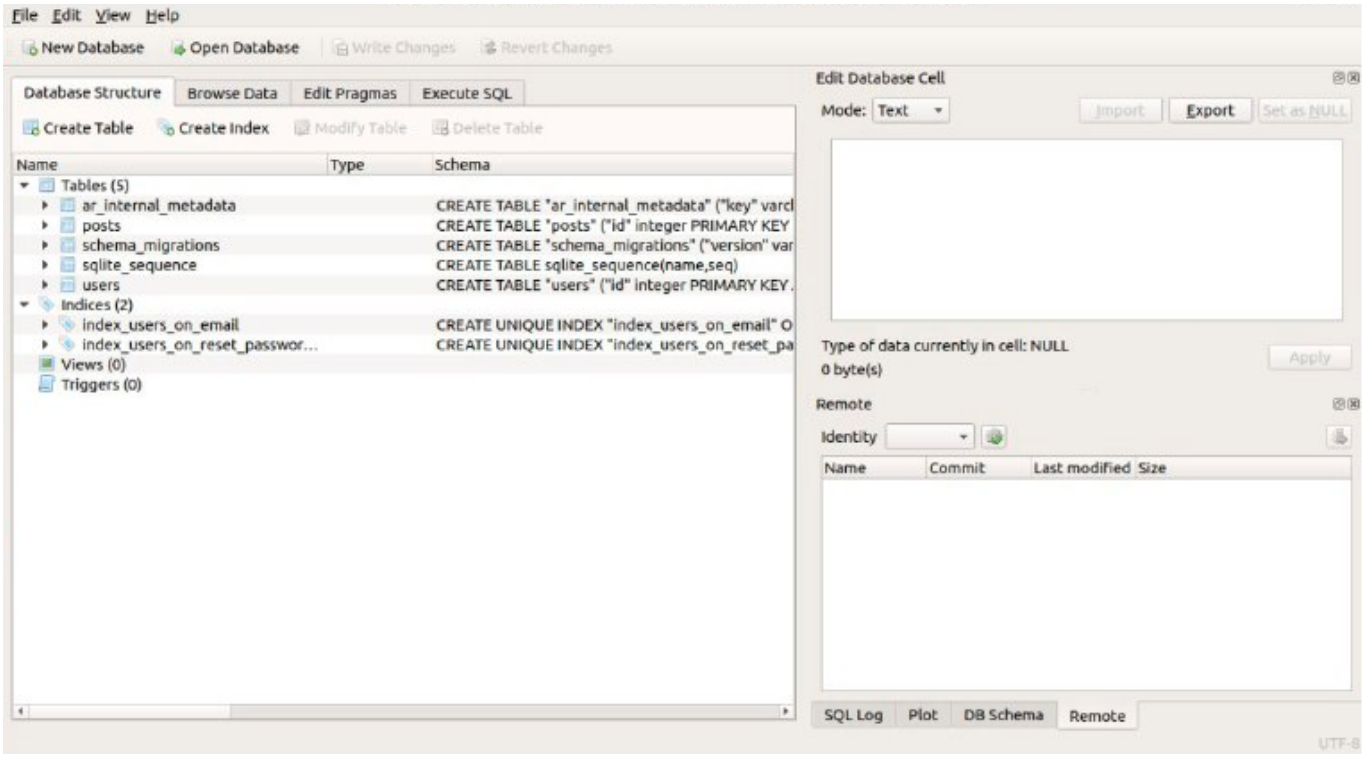

Рисунок 9.2 – Главная страница СУБД

С главной страницы программист создает новую базу данных с названием «*Myshop*» и кодировкой *UTF8*. Далее внутри базы нажимает импорт, выбирает файл **.sql* спроектированной базы данных и нажимает кнопку «Вперед». Результат импорта данных представлен на рисунке 9.4.

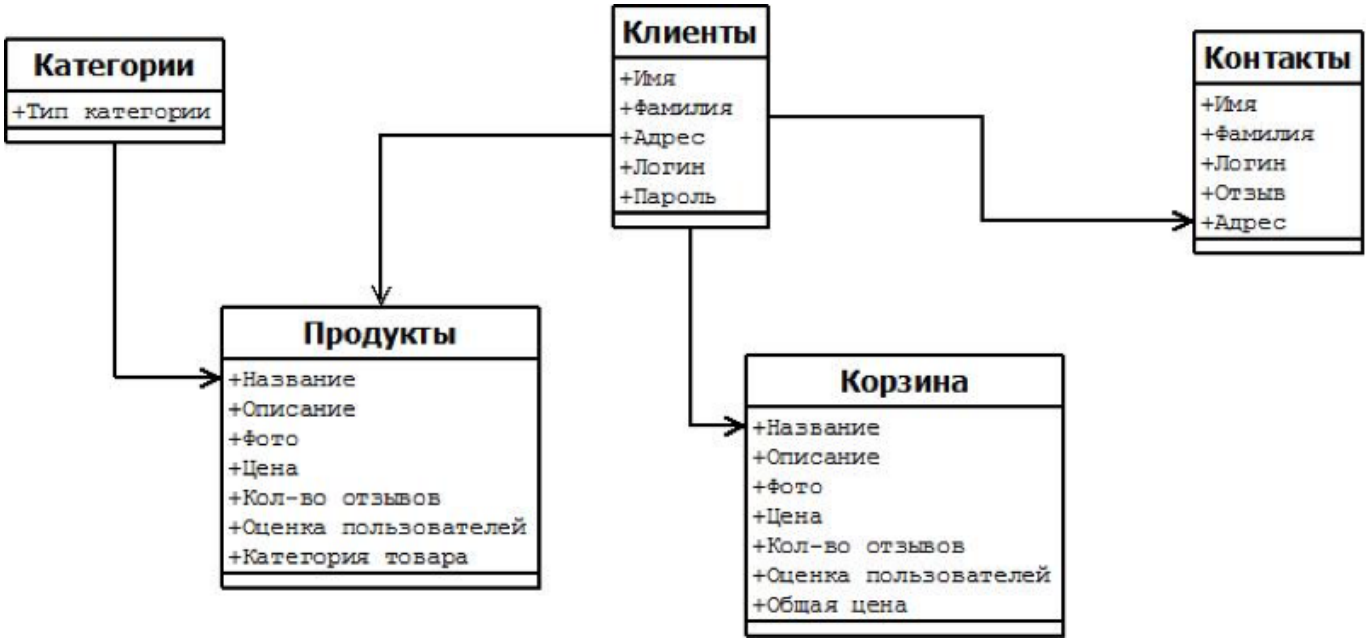

Рисунок 9.3 – Импорт базы данных

10 Результаты тестирования разработанной системы, и оценка выполнения задач

После запуска исполняемого файла перед пользователем открывается главная страница сайта, главная страница сайта при входе пользователя приведена на рисунке 10.1.

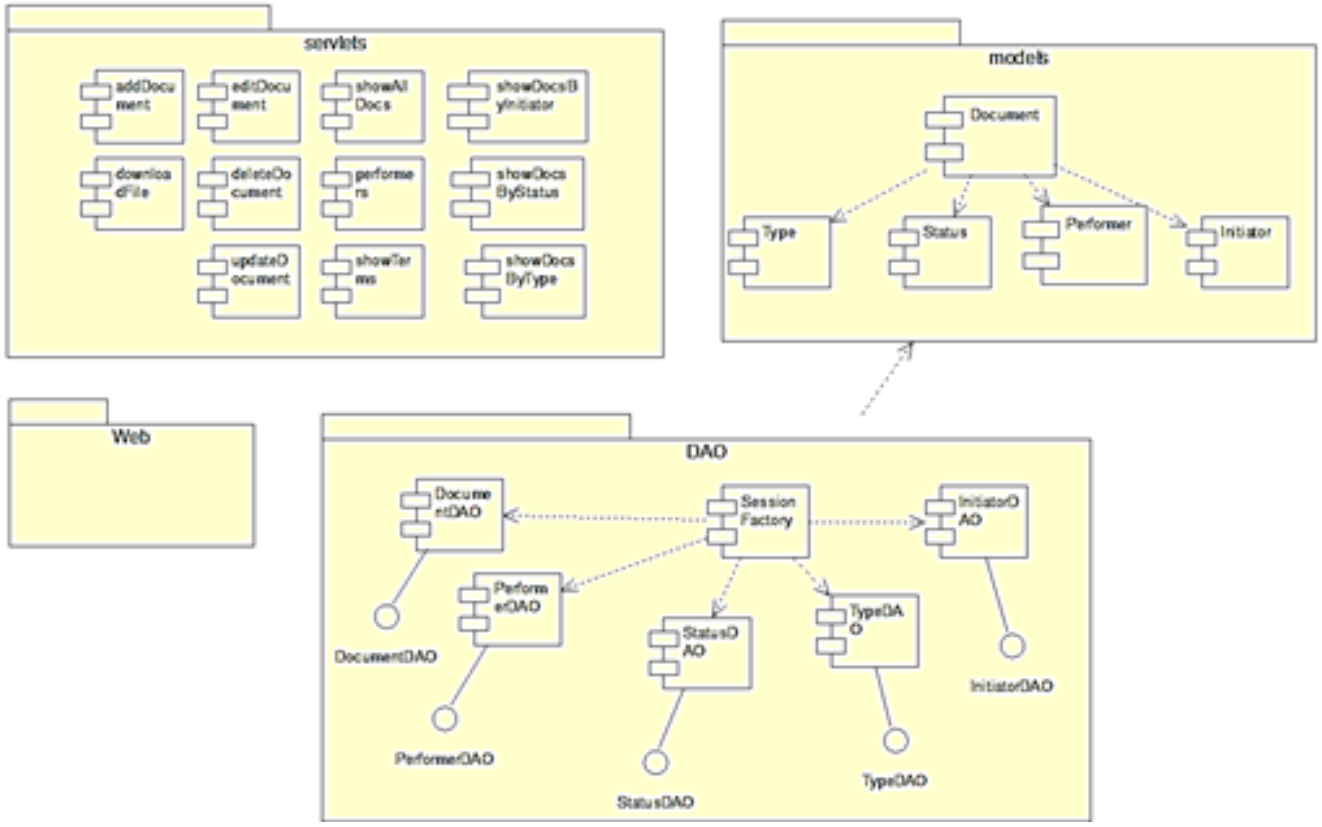

Рисунок 10.1 – Главная страница сайта тату салона

С главной страницы можно просмотреть информацию о стилях (эскизах) татуировках, информацию о сайте, контакты.

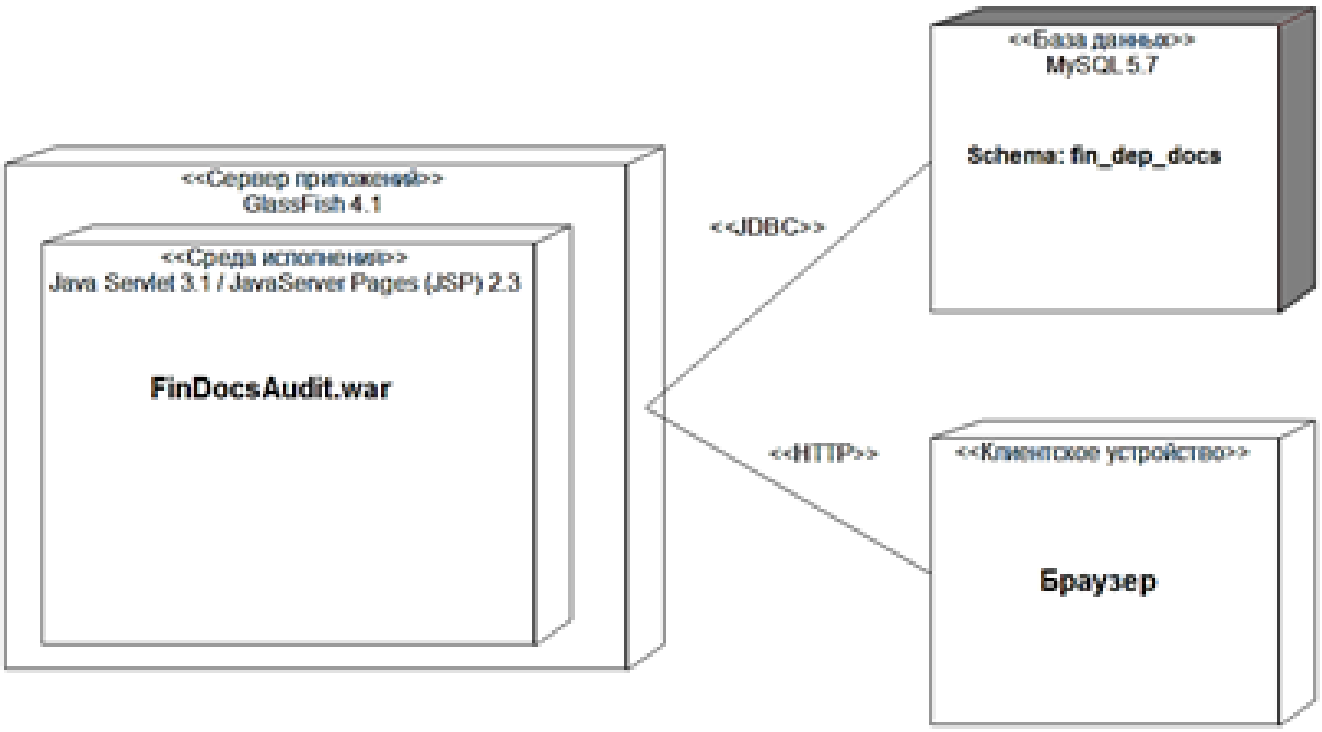

Рисунок 10.2 – Стили татуировок

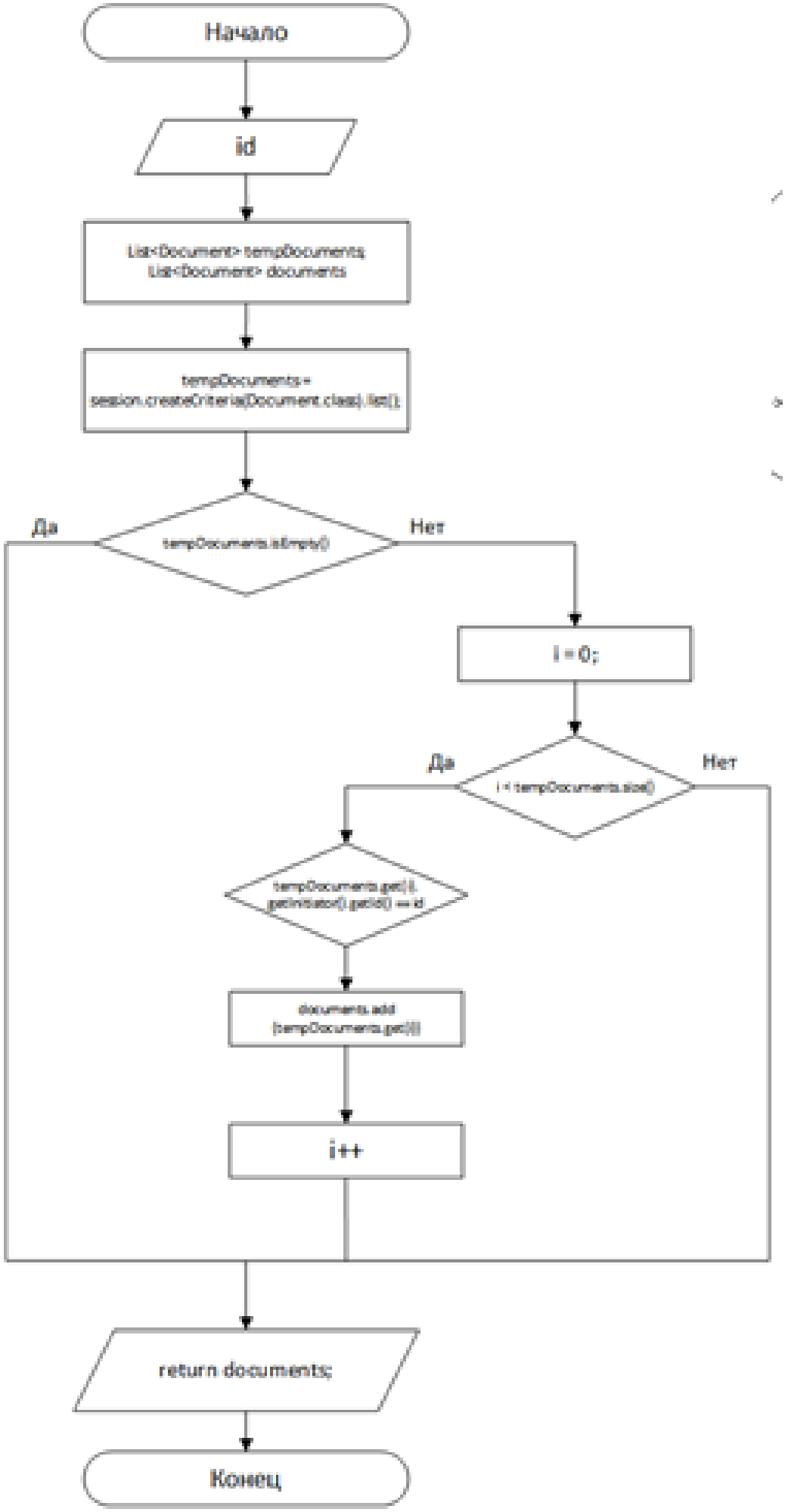
Начало
id
List<Document> tempDocuments;
List<Document> documents
tempDocuments =
session.createCriteria(Document.class).list();
Да
tempDocuments.isEmpty()
Нет
i = 0;
Да
i < tempDocuments.size()
Нет
tempDocuments.get(i).
getInstance().getId() == id
documents.add
(tempDocuments.get(i))
i++
return documents;
Конец

Рисунок 10.3 – О нас

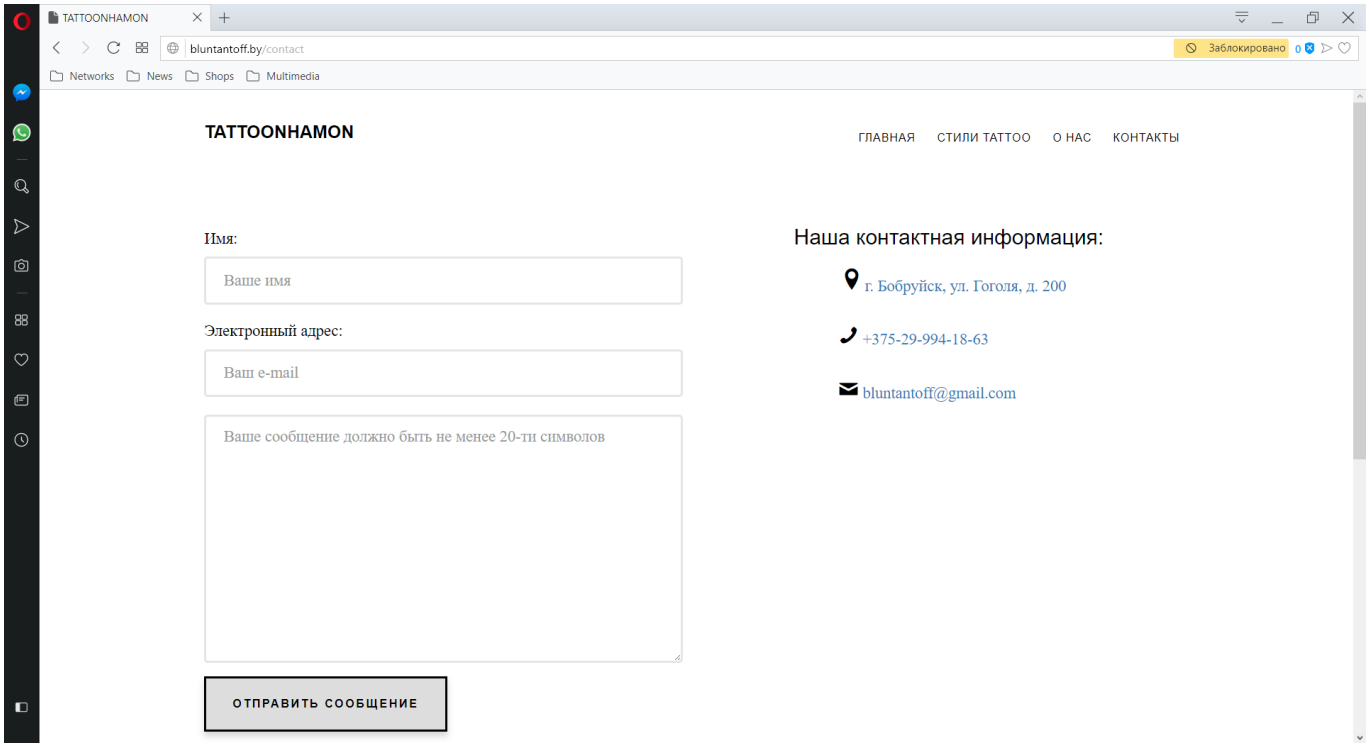

Рисунок 10.4 – Контакты

Также на сайте доступна возможность входа от администратора, с редактированием базы данных.

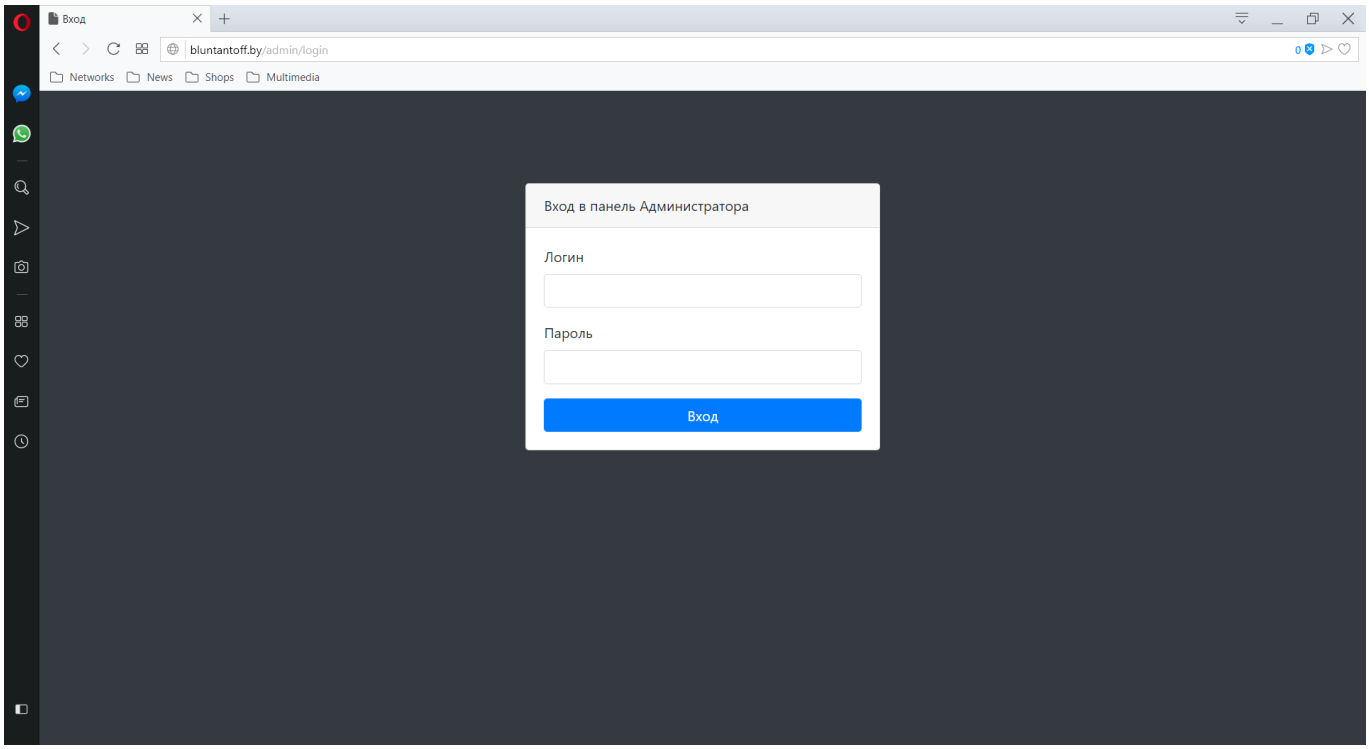

Рисунок 10.5 – Вход в панель администратора

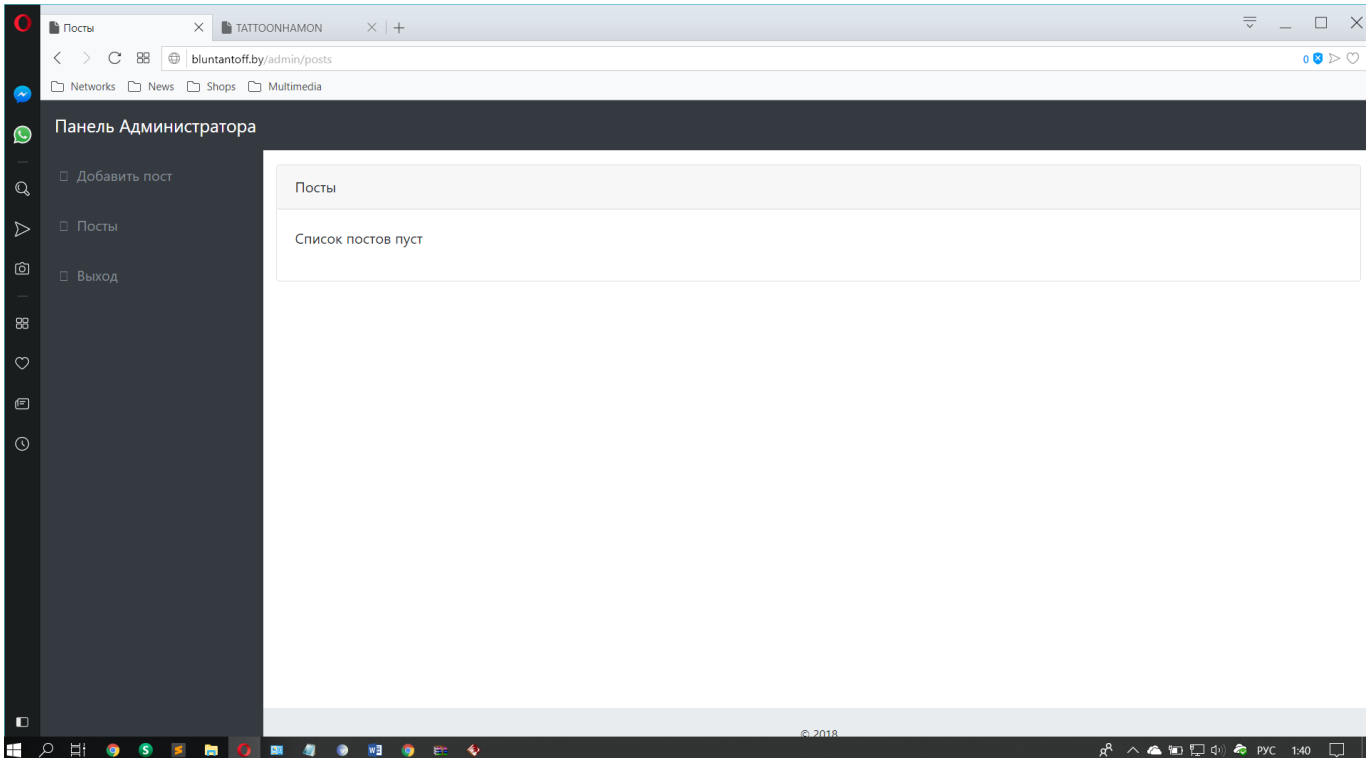

Рисунок 10.6 – Панель администратора

Таким образом, был разработан сайт, позволяющий просматривать всю необходимую информацию. Разработанным веб-приложением предусмотрен подробное описание к эскизу.

Заключение

Результатом выполнения курсового проекта является разработанный сайт тату салона. В ходе проектирования веб-приложения были рассмотрены актуальные вопросы разработки и создания современного дизайна главной страницы сайта. При этом мною были решены следующие частные задачи: – создана главная страница сайта. – отформатировано его содержимое средствами CSS. – разработаны клиентские сценарии средствами JavaScript. – произведено тестирование разработанного Веб-продукта. В результате проведенных работ на базе выбранных технологий был создан сайт тату салона.

Список использованных источников

1. [url] **Wikipedia [Электронный ресурс].** http://ru.wikipedia.org/wiki/Веб-приложение/
2. [url] **Blojek [Электронный ресурс]**
http://blojek.info/preimushhestva-i-nedostatki-veb-prilozhenij/
3. [url] **Web-приложения [Электронный ресурс]**
http://mydiv.net/arts/view-web-prilozhenija_preimuxhestva_i_nedostatki.html
4. [url] **GitHub** https://github.com/bluntantoff/tattoo.git

Приложения

1. [электронный документ] 5ed0004a9424a_Лист задания.docx