Частное учреждение образования

«Колледж бизнеса и права»

WEB-ПРИЛОЖЕНИЕ ИГРОВОГО ЦЕНТРА

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту по дисциплине

«БДиСУБД»

КП Т. 792013.401

Руководитель проекта (Ржеутская Н.В.)

Учащийся (Рабешко А.В.)

2020

Содержание

Изм.

Лист

№ докум.

Подпись

Дата

Лист

3

КП Т.792013.401

Разраб.

Провер.

.

Утверд.

*Web-приложение игрового центра*

Лит.

Листов

КБП

26

у

Введение

1. Объектно-ориентированный анализ и проектирование приложения
   1. Назначение и цели создания Web-приложения
   2. Проектирование модели
2. Проектирование веб-приложения
   1. Требования к Web-приложению
   2. Структура Web-приложения
   3. Проектирование макета Web-приложения
   4. Программно-технические средства, необходимые для разработки приложения
   5. Защита и сохранность данных
   6. Организация и ведение информационной базы
3. Реализация Web-приложения
   1. Разработка административной части приложения
   2. Разработка клиентской части приложения
   3. Описание используемых функций и процедур
4. Описание Web-приложения
   1. Общие сведения
   2. Функциональное назначение
   3. Описание разделов сайта
5. Методика испытаний
   1. Технические требования
   2. Функциональное тестирование
6. Применение
   1. Назначение программы
   2. Программно-аппаратное обеспечение сервера и клиента

Заключение

Список информационных источников

Приложение А Текст программы

*Ржеутская Н.В.*

*Рабешко А.В.*

Введение

Игровой центр представляет собой сервис который предоставляет клиентам возможность взять в аренду игровые устройства и комфортабельное место на определенный промежуток времени.

Курсовой проект посвящен созданию программного средства для автоматизации рабочего места администратора игрового центра. Рассматривается задача регистрации пользователя, регистрация заказов и клиентов, учет устройств. Основная функция создаваемого сервиса – это хранение данных о заказах.

Пользователь системы должен иметь возможность:

регистрироваться;

входить в аккаунт;

выходить из аккаунта;

добавлять устройства;

редактировать редактировать;

удалять устройства;

добавалять клиентов;

добавлять заказы;

редактировать заказы;

добавлять игровые зоны.

Существуют программные средства аналогичные данному, написанные на других языках, чаще всего встречается программное обеспечение такого типа, написанное на языке 1С.

В состав пояснительной записки будут входить шесть разделов, содержащие необходимую информацию по организации и эксплуатации Web-приложения. В первом разделе «Объектно-ориентированный анализ и проектирование приложения» описываются назначения и цели Web-приложения, а также проектирование модели.

Второй раздел «Проектирование Web-приложения» включает в себя описание требований к Web-приложению, проектирование макета Web-приложения, программно-технические средства, необходимые для разработки приложения, описание защиты и сохранности данных, организацию и ведение информационной базы.

Третий раздел «Реализация Web-приложения» содержит разработку клиентской части приложения, описание используемых функций и процедур.

Четвертый раздел «Описание Web-приложения» содержит общие сведения, функциональное назначение и описание разделов сайта.

Пятый раздел «Методика испытаний» содержит технические требования и порядок проведения испытаний: функциональное и полное тестирования.

В шестом разделе «Применение» описывается назначение программы и программно-аппаратное обеспечение сервера и клиента.

В приложении представлен текст программы.

Графическая часть представлена четырьмя диаграммами: вариантов использования, структура Web-приложения, развёртывания, деятельности.

Объектно-ориентированный анализ и проектирование приложения

Назначение и цели создания web-приложения

Актуальность данного программного продукта обусловлена возможностью увидеть товары и информацию о них с любого устройства, так же возможность загружать фото продукта.

Перечислим основные функции программы:

регистрация аккаунта;

создание заказов;

добавление устройств и мест;

создание отчетов.

Цель данного курсового проекта – создать программное средство для хранения данных о игровом центре.

Пользователь осуществляет регистрацию своей учетной записи, входит в систему. Переходя по кнопке «Главная», пользователь попадает на страницу с кнопкой «Create New» при нажатии на которую, он переходит на новую страницу, где регистрирует консоль и нажимает кнопку «Сохранить» после чего на странице появляется новая запись с названием консоли.

Для достижения поставленной цели необходимо было последовательно решить следующие задачи:

изучить возможности среды программирования, необходимые для реализации программного средства;

изучить возможности языка программирования и фреймворка;

изучить госты оформления текстовой информации;

̶ реализовать программное средство, отвечающее перечисленным выше требованиям.

Проектирование модели

На основании проведенного анализа предметной области и выявленных функциональных возможностей, необходимо будет построить модель, которая бы отображала функциональную структуру объектов программного средства, действия, производимые ими связь между этими действиями.

В качестве инструмента для построения такой модели программного средства будет выбран унифицированный язык моделирования. Данный язык моделирования предназначен для специфицирования, визуализации, конструирования и документирования программных средств.

В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций – диаграмм. В терминах языка UML определены следующие виды диаграмм: диаграмма вариантов использования, диаграмма классов, диаграмма поведения (диаграмма деятельности), диаграмма взаимодействия (диаграмма последовательности), диаграмма реализации (диаграмма компонентов), диаграмма «Сущность-связь».

Перечень этих диаграмм представляет собой неотъемлемую часть графической нотации языка UML, сам процесс объектно-ориентированного программирования (ООП) неразрывно связан с процессом построения этих диаграмм.

Главной целью проектирования моделей является отображение функциональной структуры объекта, то есть производимые ими действия и связи между этими действиями. Наиболее распространенным средством моделирования данных являются диаграммы «Сущность-связь», которые предназначены для графического представления моделей данных

разрабатываемой программной системы и предлагают некоторый набор стандартных обозначений для определения данных и отношения между ними. С помощью этого вида диаграмм можно описать отдельные компоненты концептуальной модели данных и совокупность взаимосвязей между ними, имеющих важное значение для разрабатываемой системы.

Основными понятиями данной нотации являются понятия сущности и связи. При этом под сущностью понимается произвольное множество реальных и абстрактных объектов, каждый из которых обладает одинаковыми свойствами и характеристиками. В этом случае каждый рассматриваемый объект может иметь уникальное имя или идентификатор, а также отличаться от других экземпляров данной сущности.

Связь определяется как отношение или некоторая ассоциация между отдельными сущностями. Примерами связей могут являться родственные отношения типа «отец-сын» или производственные отношения типа «начальник-подчиненный». Другой тип связей задается отношениями «иметь в собственности» или «обладать свойством».

Графическая модель строится таким образом, чтобы связи между отдельными сущностями отражали не только семантический характер соответствующего отношения, но и дополнительные аспекты обязательности связей, а также кратность участвующих в данных отношениях экземпляров сущностей.

Определим сущности для данного программного средства и построим диаграмму «Сущность-связь». Исследовав предметную область, можно выделить следующую сущность, относящиеся к данному курсовому проекту:

Для сущности «Пользователь» атрибутами являются:

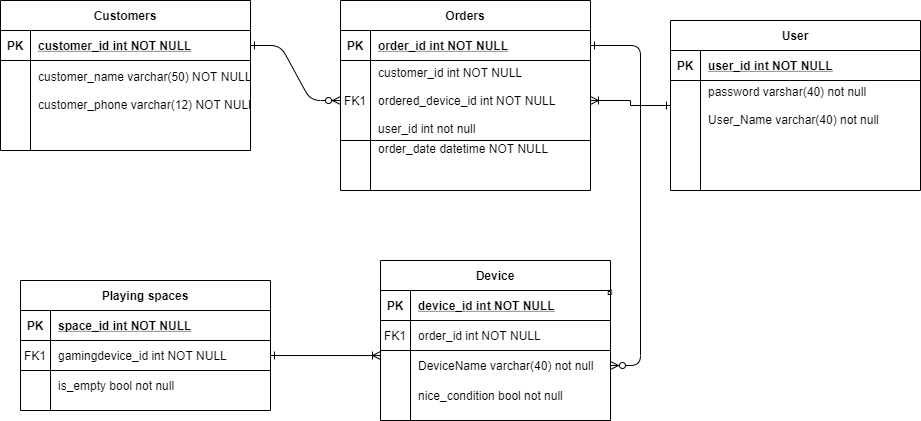
- уникальный номер;

- имя пользователя;

- пароль пользователя.

Диаграмма «Схема базы данных» представлена на рисунке 1.1

Рисунок 1.1 – Модель проекта



На диаграммах вариантов использования изображаются варианты использования, между которыми существуют отношения. Сущность – любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. Вариант использования описывает, с точки зрения действующего лица, группу действий в системе, которые приводят к конкретному результату. Вариант использования обозначается на диаграмме эллипсом, внутри которого содержится его кратное название или имя в форме глагола с пояснительными словами. Отношение – семантическая связь между отдельными элементами модели.

Включение (include) в языке UML – это разновидность отношения зависимости между базовым вариантом использования и его специальным случаем. Отношение расширения (extend) определяет взаимосвязь базового варианта использования с другим вариантом использования, функциональное поведение которого задействуется базовым не всегда, а только при выполнении дополнительных условий.

В диаграмме классов схематический класс изображается в виде прямоугольника, который дополнительно может быть разделено горизонтальными линиями на разделы. Верхний раздел содержит название класса, в среднем указывается атрибуты класса (видимость, тип и название), нижний содержит методы (видимость, тип и название). «+» – общедоступный атрибут – доступен для чтения и модификации из объектов любого класса; «–» – закрытый атрибут — доступен только объектам описываемого класса. Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности. Композиция — это такая агрегация, где объекты не могут существовать друг без друга. Диаграмма классов находится в графической части на листе 2.

В данном курсовом проекте будут реализованы классы и их методы, представленные в таблице 1.2.

Таблица 1.2 – Классы

|  |  |  |
| --- | --- | --- |
| Класс | Поля | Методы |
| Customers | Id, Name,Phone | Get, GetAll, Create |
| User | Id,Name,Password | Get,GetAll,Create,Update |
| Orders | Id,CustomerId,DeviceId,UserId,Order\_Date | Get, GetAll, Create |
| Devices | Id,OrderId,Name,Condition | Get, GetAll, Create |
| PlayingSpaces | Id,DeviceId,Empty | Get, GetAll, Create |

Диаграмма последовательности является одной из разновидности диаграмм взаимодействия и предназначена для моделирования взаимодействия объектов системы во времени, а также обмена сообщениями между ними

В данной диаграмме у нас есть актёр (пользователь), который выбирает желаемые ему песни и добавляет их в плейлист. Далее пользователь взаимодействует с плейлистом.

Диаграмма последовательности находится в графической части на листе 3.

Диаграмма деятельности – это технология, позволяющая описывать логику процедур, бизнес-процессы и потоки работ. Ветвления означают то, что есть один входной поток и несколько параллельных потоков. Поток представляет собой самый общий вид перехода и задает порядок выполнения операций. Диаграмма деятельности находится на листе 4.

Диаграмма компонентов будет представлена на листе 5.

Проектирование Web-приложения

Требование к Web-приложению

Разрабатываемое программное средство будет представлять собой Web-приложение, которое подразумевается использовать по сети Интернет.

Программное приложение будет находиться на сервере, и доступ к нему будут иметь все пользователи. Для каждой группы пользователей предусмотрено разграничение прав доступа к приложению. Работники агентства могут просматривать возможные заказы, откликаться на них, а также просматривать информацию. Операторы могут осуществлять заказы, редактировать информацию о них и клиентах, просматривать отчёты. Администратор приложения может осуществлять все действия доступные в приложении, в том числе регистрацию, изменение и удаление любой информации в приложении.

Приложение должно быть оформлено в тонах, не раздражающих и не режущих глаза. Ключевым требованием должно быть наличие удобной системы навигации и продуманный дизайн.

Визуальное оформление Web-приложения должно быть спокойным и не отвлекать от работы. Дизайн Web-сайта должен быть современным, но деловым.

Для представления текстовых материалов должен использоваться один из стандартных шрифтов. Размер (кегль) шрифта должен обеспечивать удобство восприятия текста.

Контент сайта будет представлять собой текст, таблицы и изображения.

Web-приложение должно поддерживать кроссбраузерность и производить впечатление практичной и удобной информационной системы. Пользовательский интерфейс должен быть доступным и понятным для работы. Пользователь должен легко получать доступ к необходимой информации и не использовать затруднений в процессе навигации по разделам ресурса. Для реализации этих задач необходимо найти универсальное решение, позволяющее разместить элементы Web-страниц в наиболее понятном и удобном для конечного пользователя порядке.

Верстка страниц Web-приложения должна производиться в рамках идеологии разделения структуры и представления. Язык разметки гипертекста HTML должен использоваться только для описания структуры документа, в то время как управление внешним видом Web-страниц должно осуществляться с помощью каскадных таблиц стилей CSS. Следование концепции разделения структуры и представление должно способствовать уменьшению объема кода Web-страниц.

Структура Web-приложения

Web-приложение представляет собой набор HTML-страниц, таблиц стилей, скриптов изображений и других исходных данных, которые могут объединяться и выполняться в различных браузерах.

Общая структура сайта приведена на рисунке 2.1

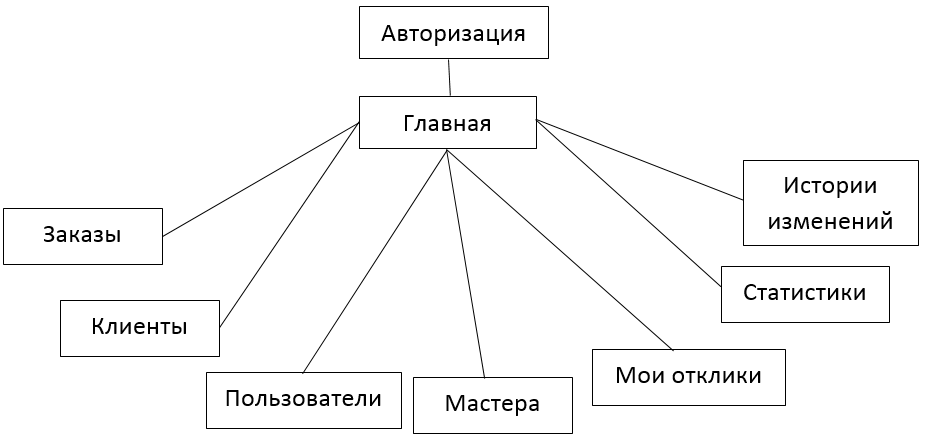


Рисунок 2.1 – Общая структура приложения

Разделы и страницы представляются в виде файлов и каталогов. Необходимо продумать их структуру, способы именования и полное дерево каталогов. Это важный момент, поскольку определяет способы связи между страницами. В приложении используется Model-view-controller -концепция.

Model-view-controller (MVC) - схема использования нескольких шаблонов проектирования, с помощью которых модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные. Данная схема проектирования часто используется для построения архитектурного каркаса, когда переходят от теории к реализации в конкретной предметной области. [11]

Концепция MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

модель предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.

представление, вид - отвечает за отображение информации (визуализацию). Часто в качестве представления выступает [форма (окно)](http://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D0%BD%D0%BE_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) с графическими элементами.

контроллер. Обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Помимо изолирования видов от логики приложения, концепция MVC существенно уменьшает сложность больших приложений. Код получается гораздо более структурированным, и, тем самым, облегчается поддержка, тестирование и повторное использование решений.

Проектирование макета web-приложения

Дизайн сайта будет «резиновый», так как большая часть информации будет представлена текстом и для удобства восприятия был выбран именно этот тип. Также дизайн сайта выполнен блочной вёрсткой, так как она удобна и легка в редактировании.

Макет сайта представляет собой хедер, футер и содержимое для контента между ними.

Макет web-приложения представлен на рисунке 2.3.

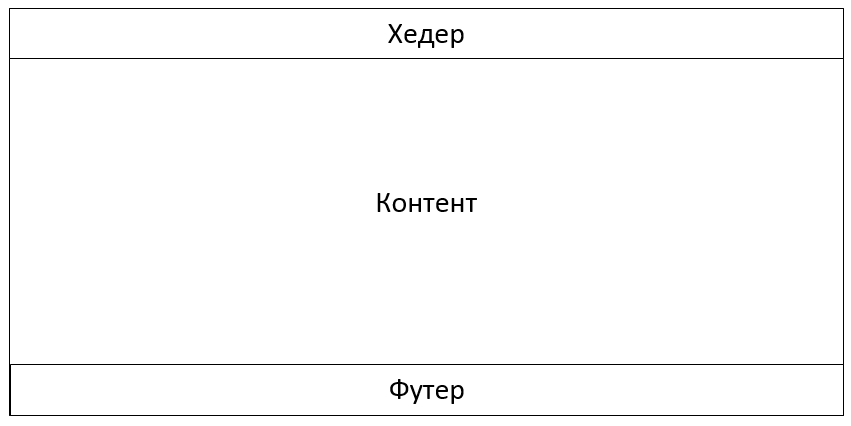


Рисунок 2.3 – Макет web-приложения

Программно-технические средства, необходимые для разработки приложения

Верстка страниц веб-сайта будет производиться в рамках идеологии разделения структуры и представления. Язык разметки гипертекста HTML должен использоваться только для описания структуры документа, в то время как управление внешним видом веб-страниц должно осуществляться с помощью каскадных таблиц стилей (CSS).

При разработке приложения будут использоваться следующие технологии:

Инструментами разработки будут являться:

операционная система Windows 10,

среда программирования Visual Studio Code,

язык программирования JavaScript,

СУБД MSSql,

язык верстки HTML,

фреймворк Material,

фреймворк Angular,

язык проектирования UML,

среда исполнения Node.JS,

язык программирования TypeScript;

Операционная система Windows 10 появилась относительно недавно – она стала доступной с 29 июля 2015 года. Именно на этой операционной системе присутствует среда разработки Microsoft Visual Studio Code, удобная для разработки приложений на языках HTML, JavaScript, TypeScript в среде исполнения Node.Js.

Microsoft Visual Studio Code − это набор инструментов для создания программного обеспечения: от планирования до разработки пользовательского интерфейса, написания кода, тестирования, отладки, анализа качества кода и производительности, развертывания в средах клиентов и сбора данных телеметрии по использованию.

При работе с Visual Studio Code, кроме стандартных библиотек, для реализации графической части поставленной задачи, будет использоваться фреймворк для создания сайтов и веб-приложений Angular.

UML – язык проектирования на его основе проектируются диаграммы программного средства, например диаграмма реляционной базы данных, на основе которой строится сама база данных приложения.

Angular – это платформа для создания мобильных и десктопных веб-приложений. Цель данного шаблона расширение браузерных приложений на основе шаблона MVC, а также упрощение тестирования и разработки.

Node.JS – это открытая, общедоступная, кроссплатформенная среда исполнения кода на языке JavaScript вне веб браузера.Node.JS позволяет разработчикам использовать JavaScript для создания инструментария через командную строку и писать серверные скрипты.

TypeScript – язык программирования, представленный Microsoft в 2012 году и позиционируемый как средство разработки веб-приложений, расширяющее возможности JavaScript.

JavaScript – мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили. Является реализацией стандарта ECMAScript. JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений.

Material – это язык дизайна для веб- и мобильных приложений, который был разработан Google в 2014 году. Material Design упрощает разработчикам настройку UI, сохраняя при этом удобный интерфейс приложений. С Material Design вы получаете хорошо организованный формат и гибкость, чтобы выразить свой бренд и стиль.

MSSql – система управления реляционными базами данных, разработанная корпорацией Microsoft.

HTML – стандартизированный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML. Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства.

Защита и сохранность данных

Защита информации представляет собой деятельность по предотвращению утечки защищаемой информации, несанкционированных и непреднамеренных воздействий на защищаемую информацию.

Для обеспечения защиты от несанкционированного доступа предусмотрено введение системы паролей (система паролей обеспечивает возможность входа в программе только для тех пользователей, которые знают логин и пароль), введение системы разграничения доступа пользователей (программа обеспечивает загрузку разного рода информации, но возможностью пользования данной информацией смогут только те, у кого есть соответствующий логин и пароль. В противном случае, при попытке загрузить информацию, пользователю будет выдано сообщении о несоответствии прав).

Осуществлены меры по защите от “SQL-инъекций”.

Механизмы безопасности системы должны предусматривать защиту от несанкционированного доступа.

Для того что бы войти в Web-приложение пользователю необходимо ввести логин и пароль.

Организация и ведение информационной базы

Реляционная модель основана на математическом понятии отношения, представлением которого является таблица. В реляционной модели отношения используются для хранения информации об объектах, представленных в базе данных. Отношение имеет вид двухмерной таблицы, в которой строки соответствуют записям, а столбцы – атрибутам. Каждая запись должна однозначно характеризоваться в таблице. Для этого используют первичные и вторичные ключи. Достоинством реляционной модели является простота и удобство физической реализации.

Реляционная модель базы данных подразумевает нормализацию всех таблиц данных. Нормализация – это формальный метод анализа отношений на основе их первичного ключа и функциональных зависимостей, существующих между их атрибутами. [6]

База данных соответствует реляционной модели данных, где каждый выделенный в ходе проектировании сущности соответствует таблица. Структура базы данных разрабатываемого программного средства включает одиннадцать таблиц.

Структура таблиц базы данных описана в таблицах 3.2,3.3,3.4,3.5 и 3.6.

Таблица 3.2 ­­­­­­­­­­­­– Структура таблицы Orders

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля | Описание |
| Id | INT | 32 | Идентификатор |
| DeviceId | INT | 32 | Идентификатор устройства |
| OrderDate | DATETIME | 8 | Время загрузки изображения |
| CustomerId | INT | 32 | Идентификатор клиента |
| UserId | INT | 32 | Идентификатор пользователя |

Таблица 3.3 – Структура таблицы Users

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля | Описание |
| Id | INT | 32 | Идентификатор |
| UserName | NVARCHAR | 8060 | Имя пользователя |
| UserPassword | NVARCHAR | 8060 | Пароль пользователя |

Таблица 3.4 – Структура таблицы Customers

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля | Описание |
| Id | INT | 32 | Идентификатор |
| CustomerName | NVARCHAR | 8060 | Имя клиента |
| CustomerPhone | NVARCHAR | 8060 | Телефон клиента |

Таблица 3.5 – Структура таблицы Devices

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля | Описание |
| Id | INT | 32 | Идентификатор |
| DeviceName | NVARCHAR | 8060 | Название устройства |
| OrderId | INT | 32 | Идентификатор заказа |
| In\_Nice\_Condition | BIT | 2 | Состояние устройства |

Таблица 3.6 – Структура таблицы PlayingSpaces

|  |  |  |  |
| --- | --- | --- | --- |
| Имя поля | Тип поля | Размер поля | Описание |
| Id | INT | 32 | Идентификатор |
| DeviceId | INT | 32 | Идентификатор устройства |
| Is\_Empty | BIT | 2 | Состояние места |

Схема базы данных представлена на рисунке 2.4

Рисунок 2.4 – Схема базы данных

Реализация Web-приложения

3.1 Разработка административной части приложения

При разработке административной части Web-приложения, его главная и все второстепенные страницы будут находиться в корневом каталоге «src», где файл «app.ts» будет заглавным файлом (так определено требованиями хостинг-провайдера).

Описание физической структуры административной части Web-приложения:

«app/controllers/AuthController.ts» - контроллер авторизации;

«app/controllers/UsersController.ts» - контроллер функций, связанных с пользователями;

«app/controllers/ClientsController.ts» - контроллер функций, связанных с клиентами;

«app/controllers/OrdersController.ts» - контроллер функций, связанных с заказами;

«app/controllers/DevicesController.ts» - контроллер функций, связанных с устройствами;

«app/controllers/PlayingSpacesController.ts» - контроллер функций, связанных с игровыми местами.

Описание логической структуры административной части Web-приложения:

«Выход» («<a routerLink="/login">») – относительная ссылка на выход из аккаунта web-приложения;

«Заявки» («<a routerLink ="/orders">») – относительная ссылка на страницу с заявками;

«Клиенты» («<a routerLink ="/clients">») - относительная ссылка на страницу с клиентами;

«Пользователи» («<a routerLink ="/users">») – относительная ссылка на страницу с пользователями;

«Заказы» («<a routerLink ="/orders">») - относительная ссылка на страницу заказов.

Разработка клиентской части приложения

При разработке клиентской части Web-приложения, его главная и все второстепенные страницы будут находиться в корневом каталоге «src», где файл «app.component.ts» будет заглавным файлом (так определено требованиями хостинг-провайдера).

Описание используемых функций и процедур

Для реализации графического интерфейса программного средства была создана единая структура страниц и форм, которые содержат все необходимые функции программного средства.

За каждой формой приложения закреплен модуль информационной части. При работе с требуемой формой подключается закрепленный за этой формой функция.

Обязательными функциями программного средства являются просмотр, добавление и изменение данных.

Для выполнения вышеперечисленных функций необходимо открыть через веб-браузер программное средство. Программное средство отобразит форму входа, после чего можно получить доступ ко всем функция приложения.

Данные о клиентах, заказах, пользователях формируются и выводятся в таблицах для удобства пользования графическим интерфейсом программного средства.

Функция вывода информации о клиентах на стороне сервера имеет следующую реализацию:

async get(request:IncomingMessage,response:ServerResponse):Promise<any>{

        try{

            let res:string;

            let info = await Customerservice.get().then(()=>{res = Customerservice.getRes()});

            //console.log(res +'---- INFO');

            setTimeout(() => {

                response.end(res);

            }, 10);

        }catch(err){

            console.log('something is gone wrong');

        }

    }

Функция добавления клиентов реализована следующим образом:

   post(request:IncomingMessage,response:ServerResponse):void{

        try{

                let data = '';

                request.on('data', chunk => {

                    data += chunk.toString();

                    console.log(data);

                });

                request.on('end',()=>{

                    let obj = parse(data);

                    console.log(obj);

                    Customerservice.post(obj);

                    response.end('Ok');

                }

                );

        }catch(err)

        {

            console.log(err);

            console.log("post error")

        }

    }

Функция удаления реализована следующим образом:

delete(request:IncomingMessage,response:ServerResponse):void{

        try{

            let data = '';

                request.on('data', chunk => {

                    data += chunk.toString();

                });

                request.on('end',()=>{

                    let obj = parse(data);

                    console.log(obj);

                    Customerservice.delete(obj);

                    response.end('Ok');

                }

                );

        }catch(err)

        {

            console.log(err);

        }

    }

Функция изменения реализована следующим образом:

put(request:IncomingMessage,response:ServerResponse):void{

        try{

            let data = '';

                request.on('data', chunk => {

                    data += chunk.toString();

                });

                request.on('end',()=>{

                    let obj = parse(data);

                    console.log(obj);

                    Customerservice.put(obj);

                    response.end('Ok');

                }

                );

        }catch(err)

        {

            console.log(err);

        }

    }

Функция авторизации в приложении представлена данным методом на сервере:

post(request: IncomingMessage, response: ServerResponse): void {

        try{

            let data = '';

            request.on('data', chunk => {

                data += chunk.toString();

            });

            request.on('end',async ()=>{

                try{

                    let obj = parse(data);

                    let result:any;

                    console.log(obj);

                    let res =await authService.post(obj).then(()=>{console.log(authService.getUser()); result =authService.getUser();});

                    if(result ==undefined|| result==null||result=='Invalid values'){

                        response.end(JSON.stringify({token:'Invalid values'}));

                    }

                    else{

                        console.log(result+' ---- RESULT');

                        //Create jwt token

                        const signature = 'drcfvtgbyhunjimk,o';

                        const token =  jwt.sign( {UserId: result.UserId} , signature, { expiresIn: '5h' });

                        console.log(JSON.stringify(result)+" --- User found");

                        console.log(token);

                        if(result!=null)response.end(JSON.stringify({token:`${token}`,name:'Authorization'}));

                        else response.statusCode;

                    }

                }catch(err){

                    console.log(err)

                }

            }

            );

    }catch(err)

    {

        console.log(err);

        console.log("post error")

    }

Полный программный код представлен в приложении А.

Описание Web-приложения

4.1 Общие сведения

Данное программное средство предназначено для автоматизации работы администратора игрового центра. Для работы с программным средством пользователю нужна любая операционная система, доступ во всемирную сеть и интернет-браузер. Пользователь может в любой момент времени на любом типе устройства вести учет и принимать новые заказы, а также просматривать и редактировать текущие.

С помощью Web-приложения администратор определяет права доступа, согласно которым пользователю разрешаются выполнять те или иные функции. На страницах выполняется проверка наличия необходимых прав доступа у пользователя, для загрузки страницы.

Верстка осуществлена согласно разработанному макету и обеспечивает удобство пользователя во время работы с сайтом. В процессе верстки учтено, чтобы все элементы дизайна располагались в определенном и логичном порядке, что позволяет экономить время при работе с приложением.

В разработанном программном средстве защита приложения производится при помощи логина и пароля, и сверки о наличии в базе данных введенной информации. Доступ к приложению возможен только после авторизации.

В процессе разработки активно применяется технология JWT — благодаря которой ограничивается несанкционированный доступ к страницам.

4.2 Функциональное назначение

Программное средство предназначено для автоматизации работы сотрудников игрового центра.

Целью разработки программы является снижение бумажной работы, уменьшение количества ошибок при работе с заявками.

В разработанном программном средстве защита программного средства производиться при помощи ввода логина и пароля, а также сверки о наличии в базе данных введенной информации пользователя.

Целевая аудитория Web-приложения представлена администраторами и пользователями.

Пользователи, принадлежащие к группе «Администратор» имеют следующие функции:

добавление, редактирование и удаление заказов;

добавление, редактирование и удаление информации о клиентах;

добавление, редактирование и удаление пользователей приложения.

Пользователи, принадлежащие к группе «Пользователь» имеют следующие функции:

добавление, редактирование заказов;

добавление, редактирование информации о клиентах.

Разработанное программное средство поддерживает кроссбраузерность, а значит будет правильно работать и отображать во всех известных браузерах.

Кроссбраузерность — это возможность приложения одинаково отображаться в разных браузерах.

4.3 Описание разделов web-приложения

При входе в систему под навигационное меню состоит из следующих пунктов: «Заказы», «Клиенты», «Устройства», «Игровые места».

Пункт меню «Заказы» служит для открытия страницы работы с заказами.

Пункт меню «Клиенты» служит для открытия страницы работы с клиентами.

Пункт меню «Устройства» открывает страницу работы с устройствами.

Пункт меню «Игровые места» открывает страницу работы с игровыми местами.

Методика испытаний

5.1 Технические требования

Web-сайт корректно отображается в следующих браузерах:

Mozilla Firefox (версия 2.0 и выше);

Google Chrome (версия 5.0 и выше);

Opera (версия 9.0 и выше).

Минимальная ширина окна браузера, при которой должно обеспечиваться полноценное отображение Web-страниц (без горизонтальной полосы прокрутки), составляет 1024 пикселей.

Для эксплуатации Web-сайта требуется Web-сервер с установленной операционной системой семейства Unix. Размер дискового пространства Web-сервера должен быть достаточным для размещения файлов системы управления, базы данных, а также всех текстовых и графических материалов Web-сайта (не менее 500 Мб).

Функциональное тестирование

В процессе написания программного средства необходимо производить тестирование на правильность его работы. Одной из основных задач тестирования является устранение ошибок, выявленных при проверке работы приложения.

Функциональное тестирование – это тестирование функций приложения на соответствие требованиям.

Рассмотрим функцию авторизации в приложении. Меню добавления клиента и ограничения на заполнение полей показаны на рисунках 5.1, 5.2 и 5.3.

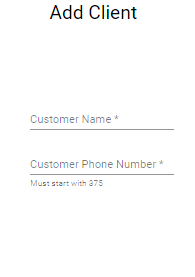


Рисунок 5.1 – Форма добавления клиента

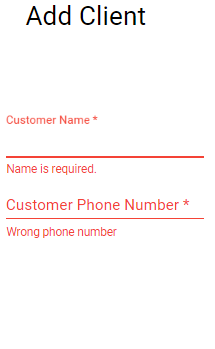


Рисунок 5.2 ­­­­­­­­­­­­­­­­­­­– Форма добавления клиента

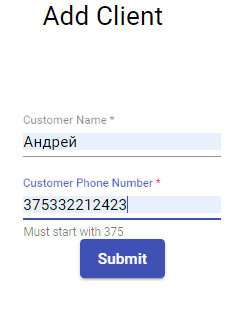


Рисунок 5.3 – Пример правильного заполнения полей

Меню входа в аккаунт пользователя изображено на рисунке 5.4.

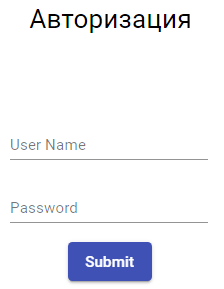


Рисунок 5.4 – Меню входа в аккаунт

Форма добавления устройства, валидация полей и пример правильного их заполнения изображены на рисунках 5.5, 5.6, 5.7.

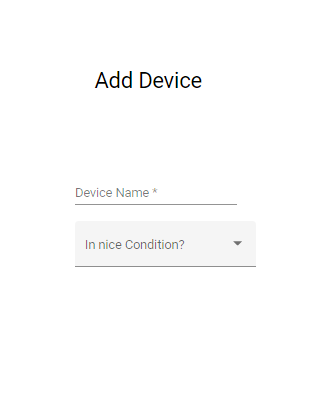


Рисунок 5.5 – Форма добавления устройства

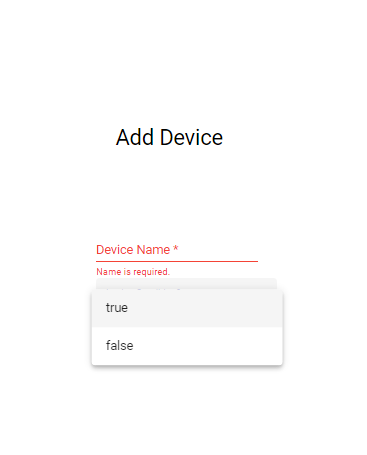


Рисунок 5.6 – Валидация полей формы



Рисунок 5.7 – Пример правильного заполнения полей

Применение

6.1 Назначение программы

Программное средство «Web-приложение игрового центра» предназначено для автоматизации работы игрового центра.

Разработанное Web-приложение позволило автоматизировать ведение базы данных.

Разработанное программное средство имеет интуитивно понятный интерфейс, довольно просто в применении.

Приложение создано с использованием базы данных MSSQL. Оно обладает рядом достоинств, повышающих его эффективность и полезность.

6.2Программно-аппаратное обеспечение сервера и клиента

Для эксплуатации Web-сайта требуется Web-сервер с установленной операционной системой семейства Unix. Размер дискового пространства Web-сервера должен быть достаточным для размещения файлов системы управления, базы данных, а также всех текстовых и графических материалов Web-сайта (не менее 500 Мб).

К конфигурации Web-сервера предъявляются следующие технические требования:

Web-сервер apache версии 2.0.0 или выше;

модуль PHP версии 5.1 или выше;

сервер баз данных MySQL версии 5.1 или выше.

Для эксплуатации программного средства пользователям необходим один из браузеров:

Mozilla Firefox (версия 2.0 и выше);

Google Chrome (версия 5.0 и выше);

Opera (версия 9.0 и выше).

Дополнительных библиотек или настроек на стороне клиента не требуется.

Заключение

В рамках курсового проекта на тему «Web-приложение игрового центра» было разработано программное средство, автоматизирующее работу сотрудников игрового центра.

Для достижения цели были решены следующие задачи:

выполнен объектно-ориентированный анализ и проектирование системы;

выбрано и описано проектирование к разработанному Web-приложению, на базе которой создавалось приложение;

выполнено проектирование макетов для разработанного Web-приложения;

описано созданное Web-приложение;

описана методика проведения испытаний;

описано применение созданного Web-приложения.

В Web-приложении были реализованы следующие функции:

ведение базы данных клиентов, рабочих ресурсов, заказов;

просмотр данных из баз клиентов, устройств, заказов;

изменение записей баз клиентов, рабочих ресурсов, заказов;

Разработка имеет интуитивно понятный графический интерфейс, позволяющий даже с минимальными знаниями компьютера пользоваться приложением.

Программное приложение реализовано в полном объеме и в соответствии с заданными требованиями, полностью отлажено и протестировано. Поставленные задачи выполнены.

Программное средство готово к практическому использованию.

Список информационных источников

1. Багласова, Т.Г. Методические указания по выполнению дипломного проекта для учащихся по специальности 2-40 01 01 «Программное обеспечение технологий» / Т.Г. Багласова. – Минск: КБП, 2017. – 30 c.
2. Багласова, Т.Г. Методические указания по оформлению курсовых и дипломных проектов / Т.Г. Багласова, К.О. Якимович. – Минск: КБП, 2013. – 29 c.
3. Бондарь, А.Г. Microsoft SQL Server 2012 / А.Г. Бондарь. – СПб. : БХВ-Петербург, 2013. – 608 с.
4. Объектно-ориентированный анализ и проектирование с примерами приложений / Гради Буч [и др.]. – 3-е изд. – М.: ООО «И.Д. Вильямс», 2008. – 720 с.
5. Программа и методика испытаний. Требования к содержанию, оформлению и контролю качества: ГОСТ 19.301-2000. – Введ. 01.09.2001. – Минск: Межгос. совет по стандартизации, метрологии и сертификации, 2000. – 14 с.
6. Текст программы. Требования к содержанию, оформлению и контролю качества: ГОСТ 19.401-2000. – Введ. 01.09.2001. – Минск: Межгос. совет по стандартизации, метрологии и сертификации, 2000. – 16 с.
7. Visual Studio 2019 [Электронный ресурс]. – Microsoft, 2019. – Режим доступа: https://visualstudio.microsoft.com/ru/vs. – Дата доступа 14.04.2020.
8. METANIT [Электронный ресурс]. – Режим доступа: https://metanit.com/ - Дата доступа 02.10.2020.
9. Аngular.io [Электронный ресурс] – Режим доступа: https://angular.io/guide/ - Дата доступа 10.11.2020.
10. MaterialAngular.io [Электронный ресурс] – Режим доступа: https://material.angular.io/ - Дата доступа: 25.11.2020.
11. Npmjs.com [Электронный ресурс] – Режим доступа: https://www.npmjs.com/package/mssql - Дата доступа 10.10.2020.

Приложение А

Текст программы

using AutoMapper;

using ImageCloud.Models;

using ImageCloudBLL.DTO;

using ImageCloudBLL.Interfaces;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace ImageCloud.Controllers

{

public class ImageController : Controller

{

IService<ImageDTO> service;

public ImageController (IService<ImageDTO> serv)

{

service = serv;

}

public ActionResult Index()

{

if (Variables.Variables.CurrentUser != null)

{

IEnumerable<ImageDTO> imageDtos = service.GetElements();

List<ImageViewModel> l = new List<ImageViewModel>();

foreach (var element in imageDtos)

{

if(Variables.Variables.CurrentUser.UserRole == "Admin")

{

l.Add(new ImageViewModel { Id = element.Id, ImageDate = element.ImageDate, ImageName = element.ImageName, Picture = element.Picture, UserId = element.UserId });

}

if (Variables.Variables.CurrentUser.UserRole == "Simple User" && Variables.Variables.CurrentUser.Id == element.UserId )

{

l.Add(new ImageViewModel { Id = element.Id, ImageName = element.ImageName, ImageDate = element.ImageDate, Picture = element.Picture, UserId = element.UserId });

}

}

return View(l);

}

return View();

}

[HttpGet]

public ActionResult UploadImage()

{

return View();

}

[HttpPost]

public ActionResult UploadImage(ImageViewModel imageViewModel,HttpPostedFileBase file)

{

var fileName = Path.GetFileName(file.FileName);

file.SaveAs(Server.MapPath("~/Data/Pictures/" + fileName));

ImageDTO uploaded = new ImageDTO { ImageName = imageViewModel.ImageName, ImageDate = DateTime.Now, Picture = "~/Data/Pictures/" + fileName, UserId = Variables.Variables.CurrentUser.Id };

service.Make(uploaded);

return RedirectToAction("Index");

}

[HttpGet]

public ActionResult Delete(int?id)

{

var im = service.GetElement(id);

ImageViewModel image = new ImageViewModel { Id = im.Id, ImageDate= im.ImageDate, ImageName = im.ImageName, Picture = im.Picture, UserId = im.UserId };

return View(image);

}

[HttpPost]

public ActionResult Delete(ImageViewModel imageViewModel)

{

service.Delete(new ImageDTO { Id = imageViewModel.Id, ImageDate = imageViewModel.ImageDate, ImageName = imageViewModel.ImageName, Picture = imageViewModel.Picture, UserId = imageViewModel.UserId });

return RedirectToAction("Index");

}

[HttpGet]

public ActionResult Details(int?id)

{

var im = service.GetElement(id);

return View(new ImageViewModel { Id = im.Id, ImageDate = im.ImageDate, ImageName = im.ImageName, Picture = im.Picture, UserId = im.UserId });

}

public ActionResult About()

{

ViewBag.Message = "Your application description page.";

return View();

}

public ActionResult Contact()

{

ViewBag.Message = "Your contact page.";

return View();

}

}

}

using AutoMapper;

using ImageCloud.Models;

using ImageCloud.Variables;

using ImageCloudBLL.DTO;

using ImageCloudBLL.Interfaces;

using ImageCloudDAL.Entities;

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace ImageCloud.Controllers

{

public class UserController : Controller

{

IService<UserDTO> service;

public UserController(IService<UserDTO> serv)

{

service = serv;

}

public ActionResult Index()

{

IEnumerable<UserDTO> Users = service.GetElements();

List<UserViewModel> users = new List<UserViewModel>();

foreach (var el in Users)

{

users.Add(new UserViewModel { Email = el.Email, Id = el.Id, IsBanned = el.IsBanned, IsEmailVerified = el.IsEmailVerified, Password = el.Password, UserName = el.UserName, UserRole = el.UserRole });

}

return View(users);

}

[HttpGet]

public ActionResult CreateUser()

{

return View();

}

[HttpPost]

public ActionResult CreateUser(UserViewModel userViewModel)

{

bool f(User m) { return m.Email == userViewModel.Email; };

UserDTO UserFormData = new UserDTO();

UserFormData = service.Find(f);

if (UserFormData!=null)

{

if (userViewModel.Email == UserFormData.Email)

{

ModelState.AddModelError("Email", "На данную почту уже зарегистрирован аккаунт ImageCloud");

}

bool f1(User m) { return m.UserName == userViewModel.UserName; };

var el1 = service.Find(f1);

if (el1 != null && userViewModel.UserName == el1.UserName)

{

ModelState.AddModelError("UserName", "Пользователь с таким ником уже зарегистрирован");

}

}

if (string.IsNullOrEmpty(userViewModel.UserName))

{

ModelState.AddModelError("UserName", "Введите имя пользователя");

}

else if (userViewModel.UserName.Length > 20)

{

ModelState.AddModelError("UserName", "ВЫберите имя покороче");

}

if (string.IsNullOrEmpty(userViewModel.Password))

{

ModelState.AddModelError("Password", "Введите пароль");

}

else if (userViewModel.Password.Length < 8)

{

ModelState.AddModelError("Password", "Пароль должен быть не менее 8 символов");

}

if (string.IsNullOrEmpty(userViewModel.Email))

{

ModelState.AddModelError("Email", "Введите адрес электронной почты , на него будет отправлено письмо подтверждения");

}

else if (userViewModel.Email.Length <6)

{

ModelState.AddModelError("Email", "Адрес электронной почты введен некорректно");

}

if (ModelState.IsValid)

{

UserDTO user = new UserDTO { UserRole = "Simple User", Email = userViewModel.Email, Id = userViewModel.Id, IsBanned = userViewModel.IsBanned, IsEmailVerified = userViewModel.IsEmailVerified, Password = userViewModel.Password, UserName = userViewModel.UserName };

service.Make(user);

return RedirectToAction("Index");

}

ViewBag.Messege = "Запрос не прошел валидацию";

return View(userViewModel);

}

[HttpGet]

public ActionResult Delete(int?id)

{

var el = service.GetElement(id);

return View(new UserViewModel { Email = el.Email, Id = el.Id, IsBanned =el.IsBanned, IsEmailVerified = el.IsEmailVerified, Password = el.Password, UserName = el.UserName, UserRole = el.UserRole });

}

[HttpPost]

public ActionResult Delete(UserViewModel user)

{

service.Delete(new UserDTO { Email = user.Email, Id = user.Id, IsBanned = user.IsBanned, IsEmailVerified = user.IsEmailVerified, Password = user.Password, UserName = user.UserName, UserRole = user.UserRole });

return RedirectToAction("Index");

}

[HttpGet]

public ActionResult Edit(int?id)

{

var el = service.GetElement(id);

return View(new UserViewModel { Email = el.Email, Id = el.Id, IsBanned = el.IsBanned, UserRole = el.UserRole, UserName = el.UserName });

}

[HttpPost]

public ActionResult Edit(UserViewModel user)

{

var el = service.GetElement(user.Id);

service.Change(new UserDTO { UserName = user.UserName, UserRole = user.UserRole, IsEmailVerified = el.IsEmailVerified, Email = el.Email, Id = user.Id, IsBanned = user.IsBanned, Password = el.Password });

return RedirectToAction("Index");

}

[HttpGet]

public ActionResult SignIn()

{

return View();

}

[HttpPost]

public ActionResult SignIn(UserViewModel user)

{

if (user.Email == null)

{

ModelState.AddModelError("Email", "Введите эллектронную почту");

}

bool f(User m) { return m.Email == user.Email && m.UserPassword == user.Password; };

var el = service.Find(f);

if(el ==null)

{

ModelState.AddModelError("Email", "Неправильно введена эллектронная почта или пароль");

}

if (el != null && el.IsEmailVerified == false)

{

ModelState.AddModelError("IsEmailVerified", "Ваша почта не подтверждена");

}

if (user.IsBanned == true)

{

ModelState.AddModelError("IsBanned", "Вход на этот сайт данному пользователю был запрещен");

}

if (ModelState.IsValid)

{

Variables.Variables.CurrentUser = new UserV { Email = el.Email, Id = el.Id, Password = el.Password, IsBanned = el.IsBanned, IsEmailVerified = el.IsEmailVerified, UserName = el.UserName, UserRole = el.UserRole };

return RedirectToAction("Index");

}

else

{

return View();

}

}

[HttpGet]

public ActionResult ConfirmEmail(int?id)

{

var el = service.GetElement(id);

el.IsEmailVerified = true;

service.Change(el);

Variables.Variables.CurrentUser = new UserV { IsEmailVerified = el.IsEmailVerified, Email = el.Email, Id = el.Id, IsBanned = el.IsBanned, Password = el.Password, UserName = el.UserName, UserRole = el.UserRole };

return RedirectToAction("Index","Image");

}

[HttpGet]

public ActionResult SignOut()

{

Variables.Variables.CurrentUser = null;

return RedirectToAction("SignIn");

}

}

}

@model IEnumerable<ImageCloud.Models.ImageViewModel>

@{

ViewBag.Title = "Image List";

}

@using ImageCloud.Variables;

@if (Variables.CurrentUser != null)

{

<h2>Images</h2>

<h3>

@Html.ActionLink("Create New", "UploadImage")

</h3>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.ImageName)

</th>

<th>

@Html.DisplayNameFor(model => model.ImageDate)

</th>

@if (Variables.CurrentUser.UserRole == "Admin")

{

<th>

@Html.DisplayNameFor(model => model.UserId)

</th>

}

<th>

@Html.DisplayName("\_\_")

</th>

<th></th>

</tr>

@foreach (var item in Model)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.ImageName)

</td>

<td>

@Html.DisplayFor(modelItem => item.ImageDate)

</td>

@if (Variables.CurrentUser.UserRole == "Admin")

{

<td>

@Html.DisplayFor(modelItem => item.UserId)

</td>

}

<td>

<img src=@Url.Content(item.Picture) class="img-thumbnail" width="50" height="50" />

</td>

<td>

@Html.ActionLink("Details", "Details", new { id = item.Id }) |

@Html.ActionLink("Delete", "Delete", new { id = item.Id })

</td>

</tr>

}

</table>

}

else

{

<h3>Приветствуем вас в новом сервисе ImageCloud</h3>

<h4>Чтобы воспользоваться сервисом необходимо @Html.ActionLink("Зарегистрироваться", "CreateUser", "User") или @Html.ActionLink("Войти", "SignIn", "User") в Аккаунт</h4>

}