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 3](#_Toc43707338)

[1 Объектно-ориентированный анализ и проектирование системы 4](#_Toc43707339)

[1.1 Сущность задачи 4](#_Toc43707340)

[1.2 Проектирование модели 4](#_Toc43707341)

[2 Вычислительная система 7](#_Toc43707342)

[2.1 Требования к аппаратным и операционным ресурсам 7](#_Toc43707343)

[2.2 Инструменты разработки 7](#_Toc43707344)

[3 Проектирование задачи 8](#_Toc43707345)

[3.1 Требования к приложению 8](#_Toc43707346)

[3.2 Концептуальный прототип 8](#_Toc43707347)

[3.3 Организация данных 10](#_Toc43707356)

[3.4 Функции: логическая и физическая организация и элементы управления 11](#_Toc43707358)

[4 Описание программного средства 12](#_Toc43707359)

[4.1 Общие сведения 12](#_Toc43707360)

[4.2 Функциональное назначение 12](#_Toc43707361)

[4.3 Входные и выходные данные 12](#_Toc43707362)

[5 Методика испытаний 13](#_Toc43707363)

[5.1 Технические требования 13](#_Toc43707364)

[5.2 Функциональное тестирование 13](#_Toc43707365)

[6 Применение 17](#_Toc43707367)

[6.1 Назначение программы 17](#_Toc43707368)

[6.2 Условия применения 17](#_Toc43707369)

[Заключение 18](#_Toc43707370)

[Список информационных источников 19](#_Toc43707371)

[Приложение А 20](#_Toc43707372)

*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 данной пояснительной записке содержится шесть разделов:

В первом разделе «Постановка задачи» перечислены основные функции программного средства, исследование предметной области.

Во втором разделе «Вычислительная система» устанавливаются минимальные требования к характеристикам ПК.

В третьем разделе «Проектирование задачи» приведены таблицы с данными о структуре меню, элементах управления.

В четвёртом разделе «Описание программного средства» содержится информация о функциональном назначении, входных и выходных данных.

В пятом разделе «Методика испытаний» содержится информация и скриншоты о функциональном и полном функциональном тестировании программного средства.

В шестом разделе «Применение» представлена информация о условиях применения и назначении программного средства.

В Заключении описана степень соответствия проектных решений заданию, а также причины их несоответствия, если таковые имеются.

Приложение А содержит код программы.

1 Объектно-ориентированный анализ и проектирование системы

## 1.1 Сущность задачи

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 продукции на складе.

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.

## 1.2 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:

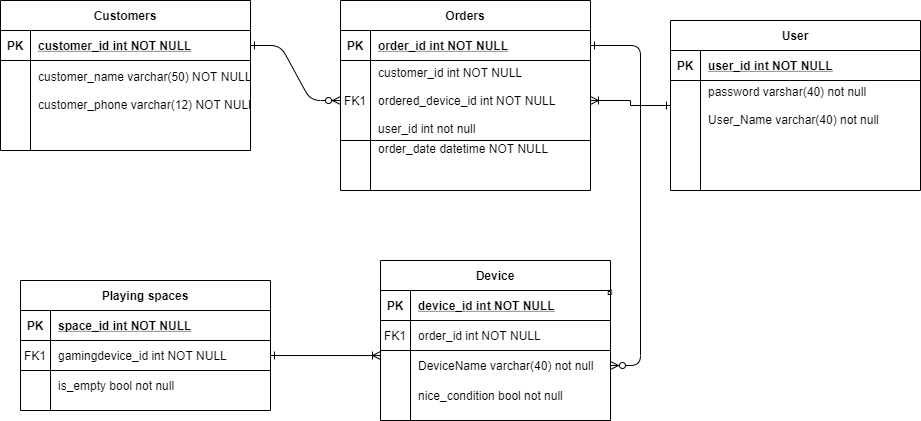
- уникальный номер;

- 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.

2 Вычислительная система

## 2.1 Требования к аппаратным и операционным ресурсам

Программное средство не требовательно к ресурсам компьютера, так как является веб−приложением и единственным требованием является наличие веб−браузера. Поэтому любой компьютер, на котором есть доступ в интернет и возможно запустить интернет−браузер подходит для работы данного программного средства.

Для разработки данного программного средства использовалось устройство со следующими характеристиками:

## 2.2 Инструменты разработки

Инструментами разработки будут являться:

* операционная система Windows 10,
* среда программирования Visual Studio Code,
* язык программирования JavaSctipt,
* язык 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 − 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, 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 – язык программирования

Material – фреймворк Angular, предоставляющий готовые графические решения.

MSSql – язык

HTML – язык верстки

3 Проектирование задачи

3.1 Требования к приложению

Средствами защиты будут являться ввод логина и пароля, а также удаление пользователя при обнаружении несостыковок.

Накладываемыми ограничениями на приложение являются необходимый выход в интернет и веб-браузер.

Главное требование к интерфейсу, чтобы он был простым и интуитивно понятным, также правильно отображался на устройствах с любой диагональю экрана.

3.2 Концептуальный прототип

Концептуальный прототип состоит из описания внешнего пользовательского интерфейса, а именно, элементов управления.

Структура кнопок навигации описана в таблице 3.1.

Таблица 3.1 – Структура меню

|  |  |  |
| --- | --- | --- |
| Уровень | Название | Действие |
| 1 | Главная | Переход на страницу с заказами |
| 1 | Зарегистрироваться | Переход на страницу регистрации |
| 1 | Войти | Переход на страницу входа |
| 1 | Выйти | Выход из учетной записи |

Прототип стартовой страницы изображен на рисунке 3.1.

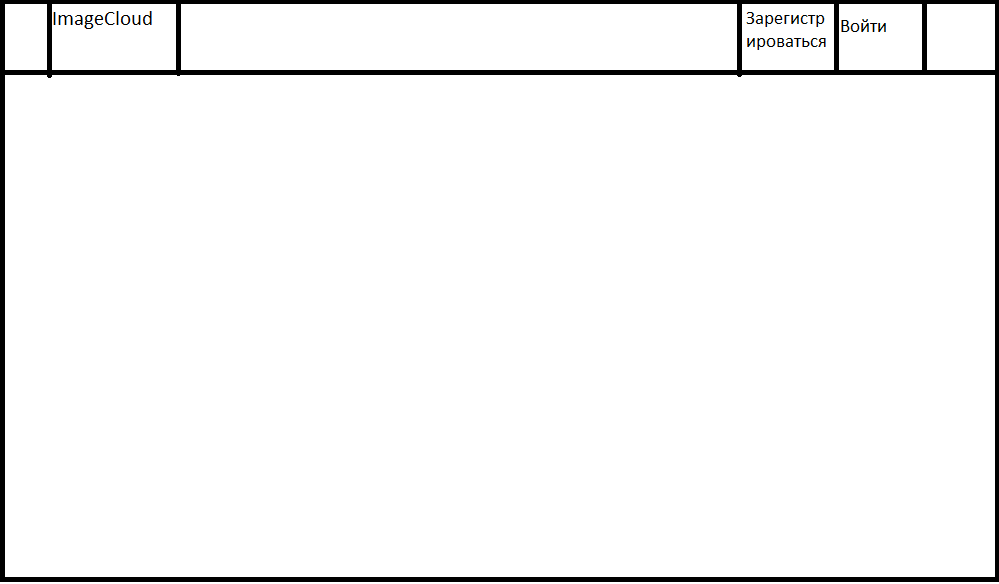


Рисунок 3.1 – Прототип стартовой страницы

Прототип главного меню загрузки изображения изображен на рисунке 3.2.

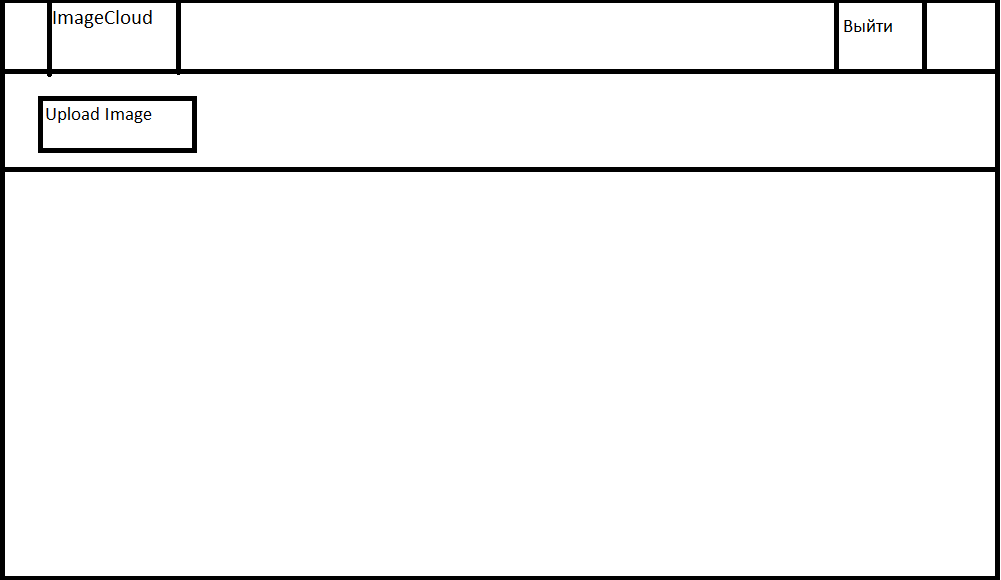


Рисунок 3.2 – Прототип главного меню

Прототип меню загрузки изображения изображен на рисунке 3.3.

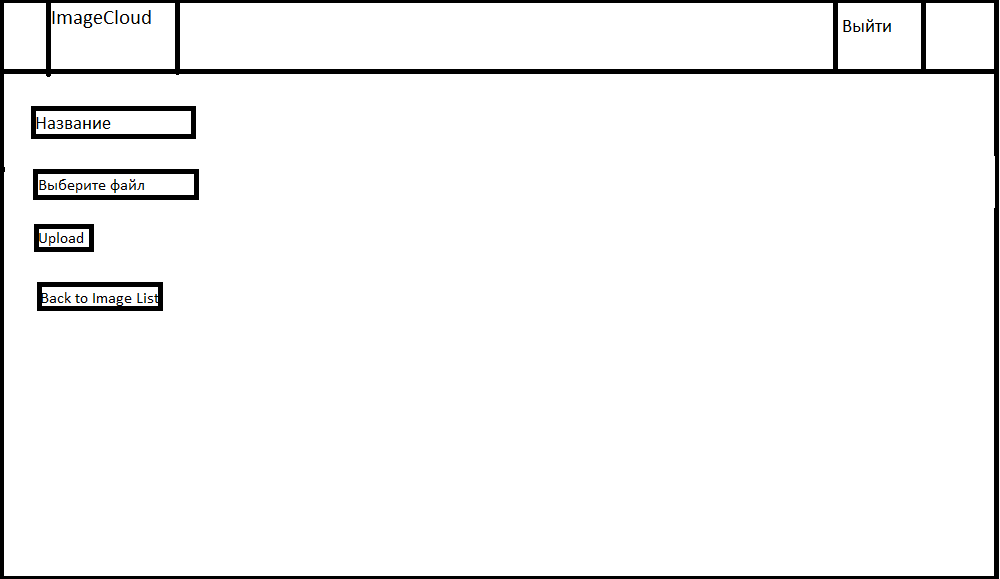


Рисунок 3.3 – Прототип меню загрузки изображения

3.3 Организация данных

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 |

3.4 Функции: логическая и физическая организация и элементы управления

Пример соответствия выполняемых функций и закрепленных за ними элементов управления приведен в таблице 3.7.

|  |  |  |  |
| --- | --- | --- | --- |
| Функция | Компонента меню | Название элемента  интерфейса | Реализация |
| Загрузка изображения | UploadImage() | Upload Image | Метод осуществляющий переход в меню загрузки изображения |
| Удаление изображения | Delete() | Delete | Метод осуществляющий удаление изображения |
| Подробная информация | Details() | Details | Метод осуществляющий подробный вывод информации об изображении |
|  |  |  |  |

Таблица 3.7 ч.1 – Функции и закрепленные за ними элементы управления

|  |  |  |  |
| --- | --- | --- | --- |
| Функция | Компонента меню | Название элемента  интерфейса | Реализация |
| Загрузка изображения | UploadImage() | Upload Image | Метод осуществляющий переход в меню загрузки изображения |
| Удаление изображения | Delete() | Delete | Метод осуществляющий удаление изображения |
| Подробная информация | Details() | Details | Метод осуществляющий подробный вывод информации об изображении |
| Создание аккаунта пользователя | CreateUser() | Зарегистрироваться | Метод осуществляющий запись пользователя в базу данных |
| Удаление аккаунта пользователя | Delete() | Delete | Метод осуществляющий удаление аккаунта пользователя |
| Изменение параметров пользователя | Edit() | Edit | Метод осуществляющий изменение параметров пользователя |
| Вход в аккаунт пользователя | SignIn() | Войти | Метод осуществляющий вход пользователя |

Таблица 3.7 ч.2 – Функции и закрепленные за ними элементы управления

4 Описание программного средства

4.1 Общие сведения

Данное программное средство предназначено для загрузки изображений и последующего хранения их на сервере. Для работы с программным средством пользователю нужна любая операционная система, доступ в всемирную сеть и интернет-браузер. Пользователь может в любой момент времени на любом типе устройства посмотреть свои изображения, но для загрузки изображений необходим компьютер с операционной системой Windows.

4.2 Функциональное назначение

Данный проект является веб−приложением, благодаря которому появляется возможность хранить изображения на сервере.

Средствами защиты данного программного средства являются:

* обязательная авторизация электронной почты пользователя;
* при нарушении правил пользователю может быть заблокирован.

4.3 Входные и выходные данные

Перед использованием данного веб−приложения пользователь обязан зарегистрироваться.

В приложение пользователь загружает изображение любого формата, типа и размера, которое преобразуется в URL ссылку, сохраняется в базе данных и появляется на странице сайта на аккаунте данного пользователя.

5 Методика испытаний

5.1 Технические требования

Для полного функционирования разработанного программного приложения, качественного выполнения всех поставленных задач, требуется ПК следующей конфигурации:

* процессор Intel Core i5-8300H CPU 2.30GHz;
* память ОЗУ 8 ГБ;
* операционная система Windows 10 Pro;
* среда разработки Visual Studio 2017.

5.2 Функциональное тестирование

Меню регистрации пользователя и ограничения на заполнение полей показаны на рисунках 5.1, 5.2 и 5.3.

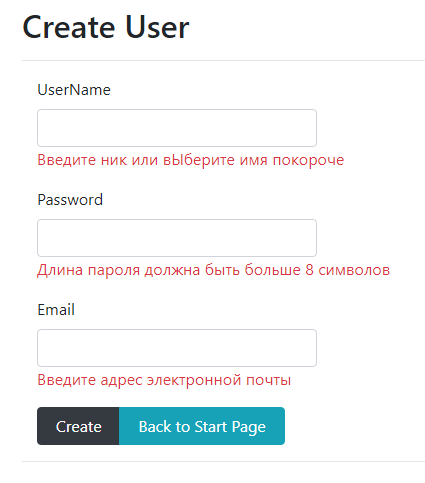


Рисунок 5.1 – Меню регистрации пользователя

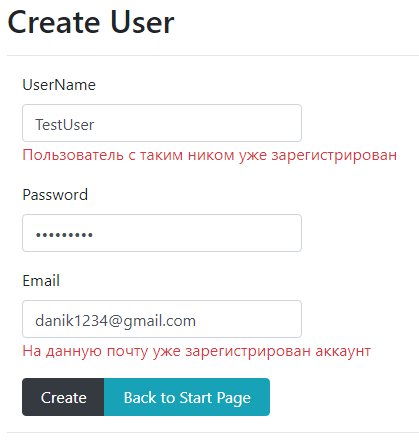


Рисунок 5.2 ­­­­­­­­­­­­­­­­­­­– Меню регистрации пользователя

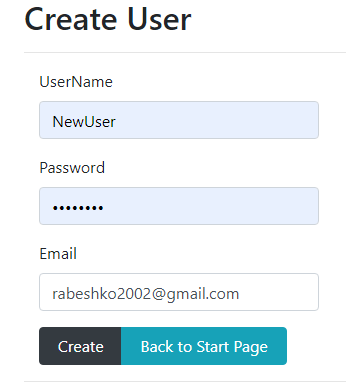


Рисунок 5.3 – Пример правильного заполнения полей

Меню входа в аккаунт пользователя изображено на рисунке 5.4.

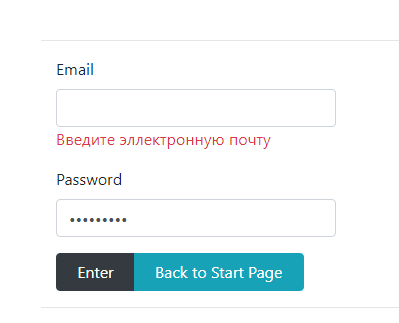


Рисунок 5.4 – Меню входа в аккаунт

При нажатии на кнопку Product Manager происходит переход на интерфейс главного меню, на котором появляются изображения, который изображен на рисунке 5.5.

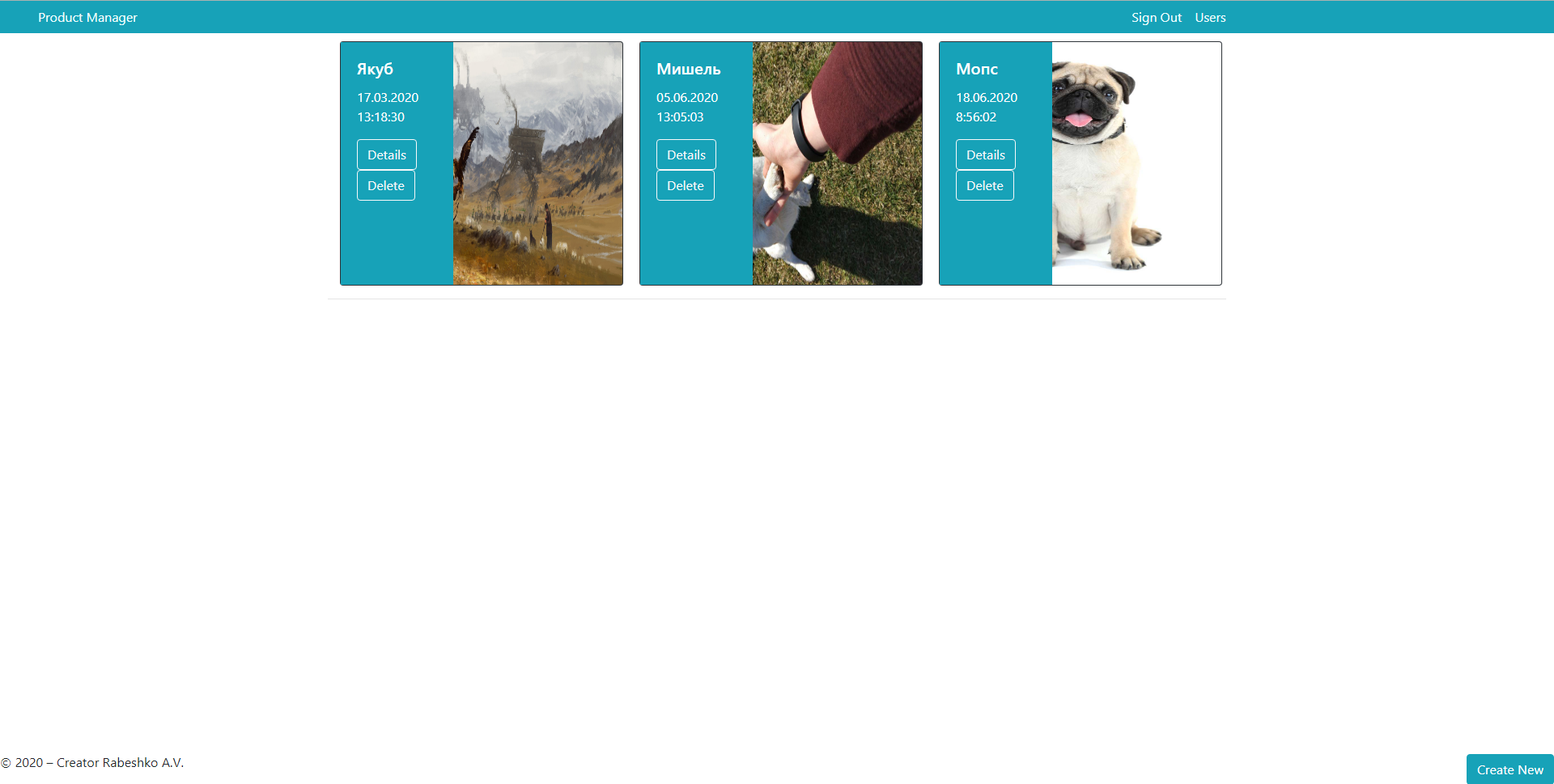


Рисунок 5.5 – Главное меню

После нажатия на кнопку Create New появляется меню создания продукта и загрузки его изображения, его вид, а также пример заполнения изображены на рисунках 5.6 и 5.7.

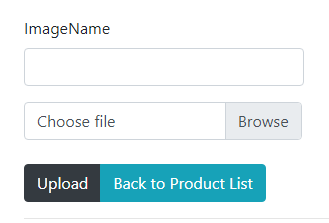


Рисунок 5.6 – Меню создания продукта

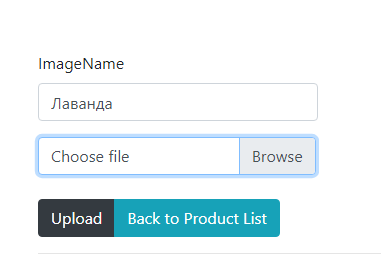


Рисунок 5.7 – Пример заполнения полей

При нажатии на кнопку Details появляется меню подробной информации о продукте изображено на рисунке 5.8.

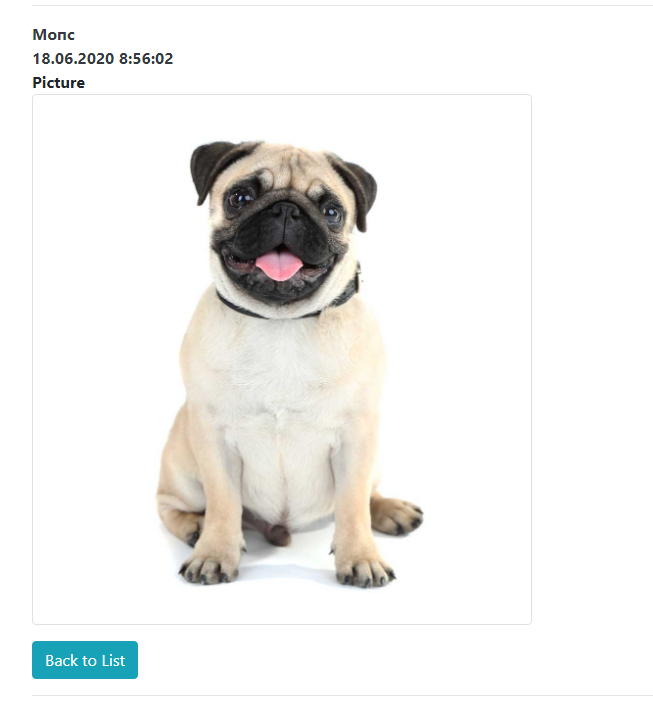


Рисунок 5.8 – Меню подробной информации о продукте

При нажатии на кнопку Delete появляется меню подтверждения операции удаления, оно изображено на рисунке 5.9.

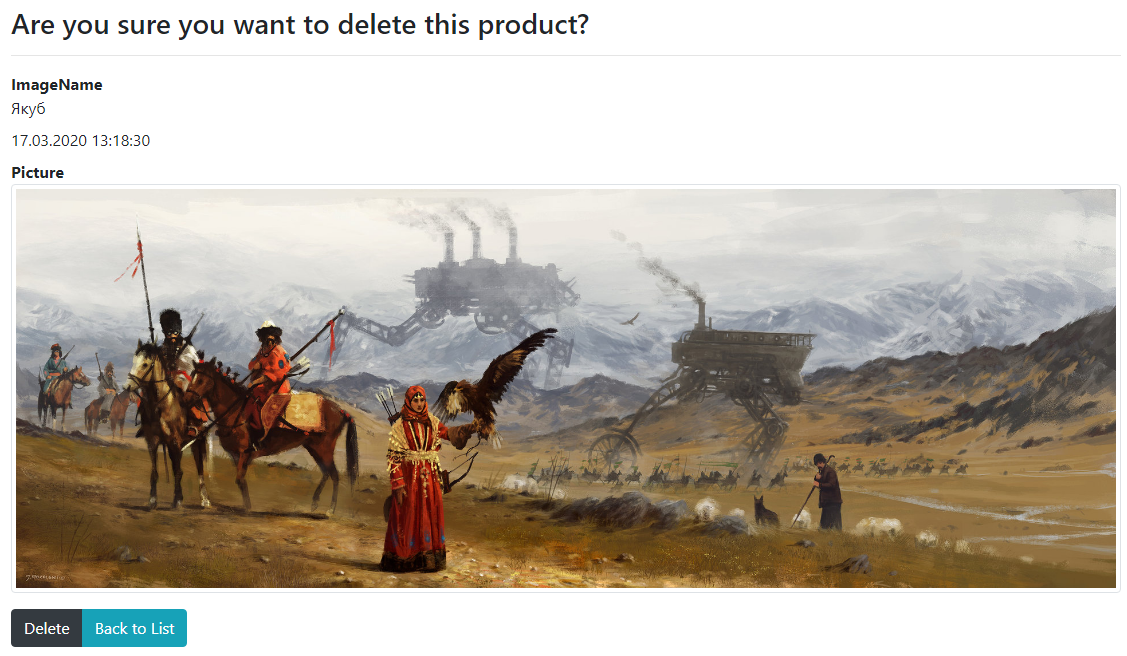


Рисунок 5.9 – Меню подтверждения операции удаления

6 Применение

6.1 Назначение программы

Программное средство предназначено для учета и хранения информации и изображений продукта на сервере. Может применяться различными интернет-магазинами для учета продукции на точках самовывоза. А также индивидуальными предпринимателями для оценки общего состояния склада и магазина в целом.

Главным ограничением является постоянный доступ к всемирной сети чтобы приложение, развернутое на каком-либо сервере, могло иметь возможность обновлять и сохранять данные.

6.2 Условия применения

Главными требованиями необходимыми для использования данного программного средства являются, доступ в интернет, установленный интернет-браузер, наличие персонального компьютера для изменения и сохранения списка продукции или смартфона для простого просмотра изображений и информации о продукции хранящихся на аккаунте на сервере.

Заключение

В рамках курсового проектирования по предмету «Конструирование программ и языки программирования» было разработано программное обеспечение «Автоматизированное рабочее место работника склада».

Программное средство имеет ряд достоинств: простой, понятный и неперегруженный цветами интерфейс, не занимает память на компьютере.

Программное средство имеет недостатки в виде невозможности обработки большого количества запросов одновременно.

В процессе курсового проектирования программное средство прошло функциональное тестирование, которое доказывает корректность выполнения всех функций и методов, отсутствие сбоев со стороны программы.

В процессе разработки программы использовался в большом объеме тематический материал и исследована предметная область разработки аналогичных программных средств.

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. Михнюк, Т.Ф. Охрана труда / Т.Ф. Михнюк. – Минск: ИВЦ Минфина, 2009. – 365 с.
5. Объектно-ориентированный анализ и проектирование с примерами приложений / Гради Буч [и др.]. – 3-е изд. – М.: ООО «И.Д. Вильямс», 2008. – 720 с.
6. Тепляков, С. Паттерны проектирования на платформе .NET / С.Тепляков. – СПб. : Питер, 2015. – 320 с.
7. 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.
8. 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.
9. Visual Studio 2019 [Электронный ресурс]. – Microsoft, 2019. – Режим доступа: https://visualstudio.microsoft.com/ru/vs. – Дата доступа 14.04.2020.
10. METANIT [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/tutorial/> - Дата доступа 02.03.2020.
11. Дж. Рихтер CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. — СПб.: Питер, 2013. — 896 с.: ил. — (Серия «Мастер-класс»).

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>

}