ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК
КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

РАЗРАБОТКА ИНТЕРНЕТ – МАГАЗИНА С УЧЕТОМ СПЕЦИФИКИ ОРГАНИЗАЦИИ ООО «ФЛИП»

Выпускная квалификационная работа
обучающегося по направлению подготовки
02.03.02 Фундаментальная информатика и информационные технологии
очной формы обучения, группы 07001301
Боднар Лейлы Мирвайсовны

Научный руководитель
к.т.н., доцент Муромцев В.В.

БЕЛГОРОД 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном информационном обществе каждая компания должна иметь собственный представительский сайт в сети Интернет, который обеспечит информационную поддержку существующего бизнеса. С помощью *web*-сайта фирмы решают такие задачи, как представление компании в сети Интернет, расширение потенциальной аудитории потребителей, поддержка бренда, повышение узнаваемости, информирование общественности и др. Разработка сайтов для компаний является актуальной и востребованной сферой деятельности, т.к. сайт фирмы в сети Интернет представляет собой достаточно дешевый и массовый способ рекламы, дает возможность потенциальным и существующим клиентам легко получать информацию о товарах и услугах компаний, их деловых интересах, что может помочь найти новых заказчиков и партнеров по бизнесу, а, следовательно, будет способствовать увеличению объема продаж и рентабельности предприятия.

С развитием интернет-технологий всё чаще наблюдаются примеры перехода настольных приложений в *web*. Уже сейчас разумнее создавать *web*-приложения для решения каких-либо задач, чем *WindowsForms*приложение, если не требуется мощности клиентской машины. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому *web*-приложения являются межплатформенными сервисами.

С ростом активного использования *web*-приложения начинают возникать вопросы о его поддержке, добавлении нового функционала к существующему, поэтому необходимо заранее предусматривать такую архитектуру *web*-приложения, которую было бы легко сопровождать в дальнейшем – исправлять существующие проблемы и добавлять новую функциональность. Для этого архитектура должна быть слабосвязанной и

составляющие *web*-приложения не должны иметьжёстко закодированных зависимостей друг от друга, а иметь лишь представления того, что они должны делать. Таким образом, изменения в одной части приложения не будут затрагивать другую часть приложения.

Объектом исследования данного дипломного проекта является процесс создания интернет – магазина для *ООО «ФЛИП»*. Магазин занимается розничной торговлей чехлов для мобильных телефонов, смартфонов, продажей планшетов. На данный момент интернет-продажи продукции предприятия упали по сравнению с оборотом 2013-2014 гг., и появилась необходимость в создании дополнительного источника прибыли.

Целью работыявляется создание *web*-приложения, представляющего собой интернет – магазин по продаже игровых приложений и ключей, которое поможет предприятию ООО «ФЛИП» расширить области продаж и увеличить прибыль. Для достижения поставленной цели необходимо решить следующие задачи:

1) обосновать выбор технологии разработки *web*-приложения;

2) разработать структуру приложения и базы данных для хранения информации о продукции магазина, покупках и покупателях;

3) разработать *web*-приложение:

• выполнить реализацию каталога товаров, разделённых по категориям;

• выполнить реализацию корзины пользователя;

4) выполнить реализацию способа оплаты и предоставления оплаченного товара.

Дипломная работа состоит из введения, трех глав, заключения и списка использованных источников. Объем работы – 44листа, в работе содержится 27 рисунков и 1 таблица.

В первой главе рассматриваются программы и среды разработки, используемые в работе. Во второй главе рассмотрено проектирование архитектуры *web*-приложения, выделены требования, выдвигаемые к сайту и

выбраны средства реализации приложения. В третьей главе производится реализация и тестирование сайта.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ СОЗДАНИЯ WEB-ПРИЛОЖЕНИЯ

1.1 Архитектура web-приложения

Web-приложения представляют собой особый тип программ, построенных по архитектуре "клиент-сервер". Особенность их заключается в том, что само *Web*-приложение находится и выполняется на сервере – клиент при этом получает только результаты работы. Работа приложения основывается на получении запросов от пользователя (клиента), их обработке и выдачи результата.

Отображением результатов запросов, а также приёмом данных от клиента и их передачей на сервер обычно занимается специальноеприложение - браузер (*Internet Explorer*, *Mozilla*, *Opera* и т. д.). Как известно, одной из функций браузера является отображение данных, полученных из Интернета, в виде страницы, описанной на языке *HTML*, следовательно, результат, передаваемый сервером клиенту, должен быть представлен на этом языке.

На стороне сервера *Web*-приложение выполняется специальным программным обеспечением (*Web*-сервером), который и принимает запросы клиентов, обрабатывает их, формирует ответ в виде страницы, описанной на языке *HTML*, и передаёт его клиенту. Одним из таких *Web*-серверов является *Internet Information Services* (*IIS*) компании *Microsoft*. Это единственный *Web*-сервер, который способен выполнять *Web*-приложения, созданные с использованием технологии *ASP.NET*.

В процессе обработки запроса пользователя *Web*-приложение компонует ответ на основе исполнения программного кода, работающего на стороне сервера, *Web*-формы, страницы *HTML*, другого содержимого, включая графические файлы. В результате

формируется *HTML*-страница, которая и отправляется клиенту. Получается, что результат работы *Web*-приложения идентичен результату запроса к традиционному *Web*-сайту, однако, в отличие от него, *Web*-приложение генерирует*HTML*-код в зависимости от запроса пользователя, а не просто передаёт его клиенту в том виде, в котором этот код хранится в файле на стороне сервера. То есть,*Web*-приложение динамически формирует ответ с помощью исполняемого кода – так называемой исполняемой части.

За счёт наличия исполняемой части, *Web*-приложения способны выполнять практически те же операции, что и обычные *Windows*-приложения, с тем лишь ограничением, что код исполняется на сервере, в качестве интерфейса системы выступает браузер, а в качестве среды, посредством которой происходит обмен данными, Интернет. К наиболее типичным операциям, выполняемым *Web*-приложениями, относятся:

- приём данных от пользователя и сохранение их на сервере;
- выполнение различных действий по запросу пользователя: извлечение данных из базы данных (*БД*), добавление, удаление, изменение данных в *БД*, проведение сложных вычислений;
- аутентификация пользователя и отображение интерфейса системы, соответствующего данному пользователю;
- отображение постоянно изменяющейся оперативной информации.

HTTP - широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам).Аббревиатура*HTTP*расшифровывается как *HyperTextTransferProtocol*, «протокол передачи гипертекста». В соответствии со спецификацией *OSI, HTTP* является протоколом прикладного (верхнего, 7-го) уровня. Актуальная на данный момент версия протокола, *HTTP 2.0*, описана в спецификации *RFC7540*.

Каждое *HTTP*-сообщение состоит из трёх частей, которые передаются в указанном порядке:

1) стартовая строка - определяет тип сообщения (запрос и ответ).

Строка запроса выглядит так:

Метод *URIHTTP/Версия*, где:

- метод - название запроса, одно слово заглавными буквами;
- *URI(UniformResourceIdentifier*- унифицированный идентификатор ресурса) - определяет путь к запрашиваемому документу;
- версия - пара разделённых точкой цифр. Например: 1.0.

2) Заголовки - характеризуют тело сообщения, параметры передачи и прочие сведения;

3) тело сообщения - непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Основным объектом манипуляции в *HTTP* является ресурс, на который указывает *URI*.

1.2 Создание web-приложения

1.2.1 Выбор языка программирования

Существует множество языков программирования, предназначенных для выполнения поставленной задачи. Каждый из них характеризуется уникальным набором операторов и особым синтаксисом. На сегодняшний день из наиболее распространённых языков, используемых для *web*-разработки, являются языки *PHP, Ruby, Python, Java* и *C#.*

У всех языков есть и достоинства, и недостатки, но для разработки *web*-приложения был выбран язык*C#.*

C# – это объектно-ориентированныйязык программирования. Он был разработан в1998-2001 годахгруппой инженеров под руководством Андерса Хейлсберга в компании *Microsoft* как язык разработки приложений для

платформы *Microsoft .NET Framework*и впоследствии был стандартизирован как *ECMA-334* и *ISO/IEC 23270*.

На сегодняшний момент язык программирования *C#* один из самых мощных, быстро развивающихся и востребованных языков в *IT*-отрасли. Сегодня на нем пишутся самые различные приложения: от небольших *WindowsForms* программ до крупных *web*-порталов и *web*-сервисов, обслуживающих ежедневно миллионы пользователей.

По сравнению с другими языками *C#* достаточно молодой, но в то же время он уже прошёл большой путь. Первая версия языка вышла вместе с релизом *MicrosoftVisualStudio .NET* в феврале 2002 года.

C# является языком с Си-подобным синтаксисом и близок в этом отношении к *C++* и *Java*. Поэтому, если вы знакомы с одним из этих языков, то овладеть *C#* будет легче.

C# является объектно-ориентированным и в этом плане много перенял у *Java* и *C++*. Например, *C#* поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных и в тоже время гибких, масштабируемых и расширяемых приложений. *C#* продолжает активно развиваться, и с каждой новой версией появляется все больше интересных функциональностей, как, например, лямбды, динамическое связывание, асинхронные методы и т.д.

Плюсы:

- мощный инструмент для создания безопасных и мощных приложений, выполняемых в среде *.NETFramework*;
- доступность литературы;
- молодой, быстро развивающийся и востребованный язык программирования.

Минусы:

- *.Net* ориентированность;
- платные разработки от компании *Microsoft*.

Так же на выбор повлияла и доступность литературы по языку *C#* и другим технологиям, предоставляемым компанией *Microsoft*для *web*-разработки.

1.2.2 Выбор технологии для создания web-приложения

Выбрав язык и платформу, нужно определиться с выбором *web*-технологии для реализации проекта.

Компания *Microsoft* предлагает несколько платформ для *web*-разработки, одни из них - это технология *ASP.NETMVC.*

В октябре 2007 г. в *Microsoft* анонсировали новую платформу веб-разработки - *MVC*, построенную на основе *ASP.NET* и явно спроектированную непосредственно в ответ на развитие технологий, подобных *Rails*, а также в качестве реакции на критику в адрес *WebForms*. В последующих разделах будет показано, каким образом эта новая платформа позволила преодолеть ограничения *WebForms* и вновь вывести *ASP.NET* на передовой уровень.

Важно различать архитектурный шаблон *MVC* и инфраструктуру *ASP.NETMVCFramework.* Шаблон *MVC* далеко не нов (его появление датируется 1978 г. и связано с проектом *Smalltalk* в *XeroxPARC*), но в наши дни он завоевал огромную популярность в качестве шаблона для *web*-приложений по перечисленным ниже причинам:

- взаимодействие пользователя с приложением *MVC* осуществляется в соответствии с естественным циклом - пользователь предпринимает действие, в ответ на которое приложение изменяет свою модель данных и доставляет обновлённое представление пользователю. Затем цикл повторяется. Это хорошо укладывается в схему *web*-приложений, предоставляемых в виде последовательностей запросов и ответов *HTTP*;
- *web*-приложения, нуждающиеся в комбинировании нескольких технологий (например, баз данных, *HTML*-разметки и исполняемого кода),

обычно разделяются на ряд слоев или уровней. Полученные в результате шаблоны естественным образом вписываются в концепции *MVC*.

Инфраструктура *ASP.NETMVCFramework* реализует шаблон *MVC* и при этом обеспечивает существенно улучшенное разделение ответственности. На самом деле в *ASP.NETMVC* внедрён современный вариант *MVC*, который особенно хорошо подходит для *web*-приложений.

За счёт принятия и адаптации шаблона *MVC* инфраструктура *ASP.NETMVCFramework* составляет сильную конкуренцию *RubyonRails* и аналогичным платформам, выводя модель *MVC* в авангард развития мира *.NET*. Обобщая опыт и наиболее рекомендуемые приёмы, обнаруженные разработчиками, которые используют другие платформы, *ASP.NETMVC* во многих отношениях превзошла даже то, что может предложить *Rails*.

Такой подход имеет несколько существенных преимуществ:

- разделение ответственности;
- широкие возможности тестирования;
- повышенная гибкость и настраиваемость;
- возможность использования уже существующей модели с различными *UI*.

Для выполнения работы была выбрана именно платформа *ASP.NETMVCFramework,* поскольку она гибкая, придерживается принципа разделения ответственности, что значительно облегчает разработку отдельных компонентов в изоляции. Так же изоляция компонентов облегчает сопровождение программы, поскольку изменения в логике, как правило, концентрируются на небольшом числе компонентов.

Помимо всего прочего в *ASP.NETMVC* привлекает простота интеграции с фреймворками и тем, что *MVC* по умолчанию использует *REST*-подход (*Representationalstatetransfer*) для *URL*-адресов (*UniformResourceLocator*).

1.2.3 Архитектура REST

REST- это стиль архитектуры программного обеспечения для распределённых систем, таких как *WorldWideWeb*, который, как правило, используется для построения *web*-служб. Термин *REST* был введён в 2000 году Роем Филдингом, одним из авторов *HTTP*-протокола. Системы, поддерживающие *REST*, называются *RESTful*-системами.

В общем случае *REST* является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации определяетсяглобальным идентификатором, таким как *URL*. Каждый*URL* в свою очередь имеет строго заданный формат.

Web-сервис – это приложение, работающее в *WorldWideWeb* и доступ к которому предоставляется по *HTTP*-протоколу, а обмен информации идёт спомощью формата *XML*. Следовательно, формат данных, передаваемых в теле запроса,всегда будет*XML*.

Для каждой единицы информации (*info*) определяется 5 действий:

- *GET /info/ (Index)* – получает список всех объектов. Как правило, это упрощённый список, т.е. содержащий только поля идентификатора и названия объекта, без остальных данных.
- *GET /info/{id} (View)* – получает полную информацию об объекте.
- *PUT /info/* или *POST /info/ (Create)* – создаётновыйобъект. Данные передаются в теле запроса без применения кодирования, даже *urlencode*.
- *POST /info/{id}* или *PUT /info/{id} (Edit)*–изменяет данные с идентификатором *{id}*, возможно заменяет их.
- *DELETE /info/{id} (Delete)* – удаляет данные с идентификатором *{id}*.

1.2.4 IoC-контейнер Ninject

InversionofControl (инверсия управления)– это абстрактный принцип, набор рекомендаций для написания слабо связанного кода, суть

которого в том, что каждый компонент системы должен быть максимально изолированным от других, не полагаясь в своей работе на детали конкретной реализации других компонентов.

DependencyInjection (внедрение зависимостей) – это одна из реализаций этого принципа (помимо этого есть ещё *FactoryMethod*, *ServiceLocator*).

IoC-контейнер– это какая-то библиотека, фреймворк, которая позволит упростить и автоматизировать написание кода с использованием данного подхода на столько, на сколько это возможно. Их довольно много, в этой работе используется *Ninjec*t.

1.2.5 Используемые фреймворки

Moq – известная библиотека с открытым исходным кодом, которая была разработана с использованием возможностей *C# 3.0* (деревья выражений *LINQ* и лямбда-выражения). Один из принципов библиотеки звучит как «с очень низким порогом входа получить хорошие возможности для рефакторинга». *Moq* достаточно проста в изучении и использовании и позволяет создавать «моки» на необходимые методы и свойства с использованием лямбда-выражений.

Bootstrap- свободный набор инструментов для создания сайтов и *web*-приложений. Он включает в себя*HTML* и *CSS*шаблоны оформления длятипографики,*web*-форм, кнопок, меток, блоков навигации и прочих компонентов *web*-интерфейса, включая *JavaScript*-расширения.

Bootstrap использует самые современные наработки в области *CSS* и *HTML*, поэтому необходимо быть внимательным при поддержке старых браузеров.

Основные преимущества *Bootstrap*:

- экономия времени –*Bootstrap* позволяет сэкономить время и усилия, используя шаблоны дизайна и классы, и сконцентрироваться на других разработках;
- высокая скорость – динамичные макеты *Bootstrap* масштабируются на разные устройства и разрешения экрана без каких-либо изменений в разметке;
- гармоничный дизайн – все компоненты платформы *Bootstrap*используют единый стиль и шаблоны с помощью центральной библиотеки. Дизайн и макеты веб-страниц согласуются друг с другом;
- простота в использовании – платформа проста в использовании, пользователь с базовыми знаниями *HTML* и *CSS* может начать разработку с *TwitterBootstrap*;
- совместимость с браузерами –*TwitterBootstrap* совместим с *MozillaFirefox, YandexBrowser, GoogleChrome, Safari, InternetExplorer, MicrosoftEdge* и *Opera*;
- открытое программное обеспечение – особенность *TwitterBootstrap*, которая предполагает удобство использования, посредством открытости исходных кодов и бесплатной загрузки.

Основные инструменты *Bootstrap:*

- сетки – заранее заданные размеры колонок, которые можно сразу же использовать, например ширина колонки 140*px* относится к классу .*span2* (.*col-md-2* в третьей версии фреймворка), который можно использовать в *CSS* описании документа;
- шаблоны –фиксированный или резиновый шаблон документа;
- типографика– описание шрифтов, определение некоторых классов для шрифтов, таких как код, цитаты и т. п.;
- медиа – представляет некоторое управление Изображениями и Видео;

- таблицы –средства оформления таблиц, вплоть до добавления функциональности сортировки;
- формы –классы для оформления форм и некоторых событий, происходящих с ними;
- навигация –классы оформления для Табов, Вкладок, Страничности, Меню и Тулбара;
- алерты–оформление диалоговых окон, Подсказок и Всплывающих окон.

Razor– это название механизма визуализации, который был введён *Microsoft* в версии *MVC 3* и переработан в версии *MVC 4* (хотя изменения были относительно небольшими). Механизм визуализации обрабатывает контент *ASP.NET* и ищет инструкции, которые обычно вставляет динамический контент в вывод, отправляемый браузеру. В *Microsoft* поддерживаются два механизма визуализации: механизм *ASPX*, работающий с дескрипторами <% и %>, которые были основной опорой разработки *ASP.NET* в течение многих лет, и механизм *Razor*, имеющий дело с областями контента, которые обозначены с помощью символа @.

1.3 Разработка базы данных

Поскольку *ASP.NETMVC* построен поверх *.NET 4.5*, при разработке уровня доступа к данным в приложении могут использоваться любые популярные инфраструктуры доступа к данным, в том числе *ADO.NET, LINQtoSQL, ADO.NETEntityFramework* и *NHibernate*.

Для базы данных будет применяться *SQL Server*, а доступ к ней будет осуществляться с помощью *EntityFramework (EF)*–объектно-ориентированная технология доступа к данным, является *object-relationalmapping (ORM)* решением для *.NET Framework* от *Microsoft*. Предоставляет возможность взаимодействия с объектами как посредством *LINQ* в виде *LINQ toEntities*, так и с использованием *Entity SQL*. Для

облегчения построения *web*-решений используется технология *ADO.NET DataServices (Astoria),* позволяющая строить многоуровневые приложения, реализуя шаблон проектирования *MVC*.

EntityFramework предполагает три возможных способа взаимодействия с базой данных:

- подход **Model-First**, впервые появившийся в версии *EntityFramework 4*, применяется разработчиками, которые не хотят использовать инструменты *СУБД* для создания и управления базами данных, а также не хотят вручную настраивать классы модели *EDM*. Фактически это самый простой подход при работе с *EntityFramework*. Проектирование модели происходит в графическом дизайнере *EDM* среды *VisualStudio*;
- подход **Database-First**, появившийся вместе с*EntityFramework*, позволяет писать приложения для существующих баз данных. Базы данных в реальных приложениях довольно быстро становятся сложными, и пытаться создать модель для существующей базы данных, которую могут понять разработчики, довольно трудно. Ещё тяжелее написать код использования модели, в котором происходит взаимодействие с базой данных. Во многих отношениях, подход *Database-First* является противоположностью подходу *Model-First*. При подходе *Database-First* база данных уже существует, поэтому разработчик должен знать, где расположена база данных, а также иметь информацию об имени базы данных. Тем не менее, разработчик не должен понимать внутреннюю работу базы данных –*EntityFramework* по-прежнему скрывает внутреннюю реализацию из поля зрения. При этом подходе, рабочий процесс создания модели начинается с создания и проектирования базы данных. После генерации сущностных классов модели из существующей базы данных, работа с *EntityFramework* аналогична подходам *Code-First* и *Model-First*. Это означает создание объекта класса контекста и использование этого объекта для выполнения необходимых задач;

- подход **Code-First**, который впервые появился в *EntityFramework 4.1*, обычно используется, когдаужеесть существующее приложение, содержащее модель данных. Эта модель, как правило, описывается с помощью нескольких классов и кода взаимодействия между этими классами.

Теперь сначала пишется код, описывающий классы-модели, а потомфреймворк автоматически создает*БД* по такому коду.

Самое приятное, что виды отношений будут распознаны – достаточно определить просто ссылки на объекты для 1:1, *ICollection* для отношения 1:n, и взаимные *ICollection*для m:n. В этом случае промежуточная таблица также создастся автоматически. Для обеспечения «ленивой» загрузки хватит ключевого слова *virtual* в определении свойства. Этот подход и был выбран при разработке приложения. Инфологическая (физическая) модель базы данных представлена в приложении 7.

1.4 Разработка системы контроля версий

Для удобства и избегания бытовых проблем при написании данного *web*-приложения, была использована система контроля версий.

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

В процессе работы была использована система контроля версий *Git.*

Git– это распределённая система управления версиями файлов. Код программы написан в основном на языке C. Проект был создан ЛинусомТорвальдсом в 2005 году для управления разработкой ядра *Linux* и, как и *GNU/Linux*, является свободным программным обеспечением (*ПО*), при этом стороннее использование подчиняется лицензии *GNUGPL* версии 2. Коротко данное соглашение можно охарактеризовать как *ПО* со свободным кодом, которое должно развиваться открыто, т.е. любой программист вправе

продолжить совершенствование проекта на любом его этапе. За своё недолгое время существования данная система была введена многими ведущими разработчиками. *Git* используется в таких известных *Linux*-сообществу проектах, как *Gnome, GNUCoreUtilities, VLC, Cairo, Perl, Chromium, Wine*.

Особенностью *Git* является то, что работа над версиями проекта может происходить не в хронологическом порядке. Разработка может вестись в нескольких параллельных ветвях, которые могут сливаться и разделяться в любой момент проектирования.

Git– довольно гибкая система, и область её применения ограничивается не только сферой разработки. Например, журналисты, авторы технической литературы, администраторы, преподаватели вузов вполне могут использовать её в своём роде деятельности. К таковым задачам можно отнести контроль версий какой-либо документации, доклада, домашних заданий.

1.5 Информация об организации

В дипломном проекте рассмотрим предприятие ООО «ФЛИП», расположенное по адресу г. Нововоронеж. В основном оно ориентируется на средний класс покупателей, а также на лиц от 12 лет и старше. Организационная структура рассматриваемого магазина довольно проста, поскольку владелец предприятия является, как руководителем, так и продавцом в своем магазине. Все бухгалтерские и налоговые расчеты также выполняет сам. С 2013 года ООО «ФЛИП» занимается продажей телефонов, смартфонов, планшетов, а также мобильными аксессуарами методом дропшиппинг (*dropshipping*),который позволяет основателю интернет-магазина заниматься реализацией товаров поставщика напрямую к покупателю, а прибыль интернет-магазина составляет разницу между оптовой ценой товара, установленной поставщиком и розничной ценой, по

которой в итоге продается товар в интернет-магазине, за вычетом накладных расходов. Однако, в виду большой конкуренции в данной сферереализация данной продукции перестала приносить желаемый доход. В связи с чем в конце 2016 года владелец предприятия решил расширить сферу продаж и обратить внимание на развивающуюся в данный момент в России сферу компьютерных игр.

Данная на рис. 1.1 модель описывает основную функцию интернет-магазина игр- функцию автоматизации продажи товаров через интернет:

Рис. 1.1. Контекстная диаграмма функциональной модели

Выходной поток один - это «Отправленный товар». Управляющий потоков также один - это «законодательство РФ», которое влияет на деятельность интернет-магазина.

Для процесса «Продажа товаров через интернет» можно указать следующие входные потоки:

1) данные клиента;
2) запросы клиента.

На рис. 1.2 показан второй уровень процесса «Продажа товаров через интернет», который делится на 2подпроцесса:

1) товары, добавленные в корзину;

2) подтверждение заказа.

Рис. 1.2. Второй уровень функциональной модели

Процесс «Подтверждение заказа» декомпозируется на 2подпроцесса, как представлено на рис. 1.3:

1) подсчет стоимости заказа и оплата;

2) подтверждение оплаты пользователем.

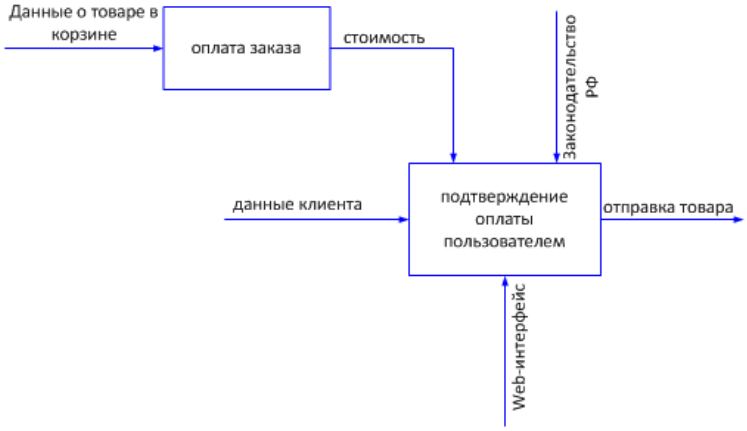


Рис. 1.3. Процесс «Подтверждение заказа»

2 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ WEB-ПРИЛОЖЕНИЯ

2.1 Используемые инструменты

В качестве архитектурного каркаса *web*-приложения используется *ASP.NETMVC5Framework.*

Хранилищем данных выступит *MicrosoftSQLServer*.

В качестве *ORM*фреймворка применяется *MicrosoftEntityFramework.*

Для разработки *web*-приложения использовалось следующее программное обеспечения:

1. *Visual Studio 2015 Enterprise ver.14.0*
2. *Microsoft .NET Framework ver.4.6*
3. *Microsoft SQL Server Management Studio 2014 ver.12.0*
4. *Git ver.2.8.3*

2.2 Проектирование архитектуры приложения

Обычно,при разработке, решение состоит из одного проекта, но для лучшей гибкости и изоляции бизнес-модели от пользовательского интерфейса, компоненты решения разносятся по разным проектам. При этом общее корневое пространство имён применяется единым образом во всех проектах.Каждый проект должен иметь уникальное подпространство имён, которое соответствует имени сборки. При разработке сайта-магазина для продажи игровых ключей и аккаунтов функциональность приложения была разбита на 3 проекта.

Проект *GamingKeyStore.WebUI* содержит контроллеры и представления, выступает в качестве пользовательского интерфейса для приложения *GamingKeyStore*.

Проект *GamingKeyStore.Domain* содержит сущности и логику предметной области, настраивается на обеспечение постоянства посредством хранилища, которое создано с помощью инфраструктуры *EntityFramework*.

Проект *GamingKeyStore.Tests* содержит модульные тесты для других двух проектов.

Таким образом при разработке данного приложения реализованы некоторые элементы из существующего набора принципов *Domain-drivendesign* (предметно-ориентированное проектирование).

Domain-drivendesign– это набор принципов и схем, помогающих разработчикам создавать изящные системы объектов. При правильном применении оно приводит к созданию программных абстракций, которые называются моделями предметных областей. В эти модели входит сложная бизнес-логика, устраняющая промежуток между реальными условиями области применения продукта и кодом.

Эта концепция выделяет следующие элементы построения моделей предметной области:

- сущность – объект, который определяется не значением своих атрибутов, а уникальным идентификатором;
- объект-значение – объект, который определяется значением своих атрибутов;
- множество – коллекция объектов, связанных между собой корневой сущностью, так же называемой корнем множества;
- репозиторий – содержит методы получения доменного объекта из хранилища данных. Может быть заменён при использовании альтернативного хранилища данных;
- фабрика – содержит методы для создания доменного объекта.

В соответствии с концептуальной моделью данных можно выделить следующие классы доменных объектов как показано на рис. 2.1:

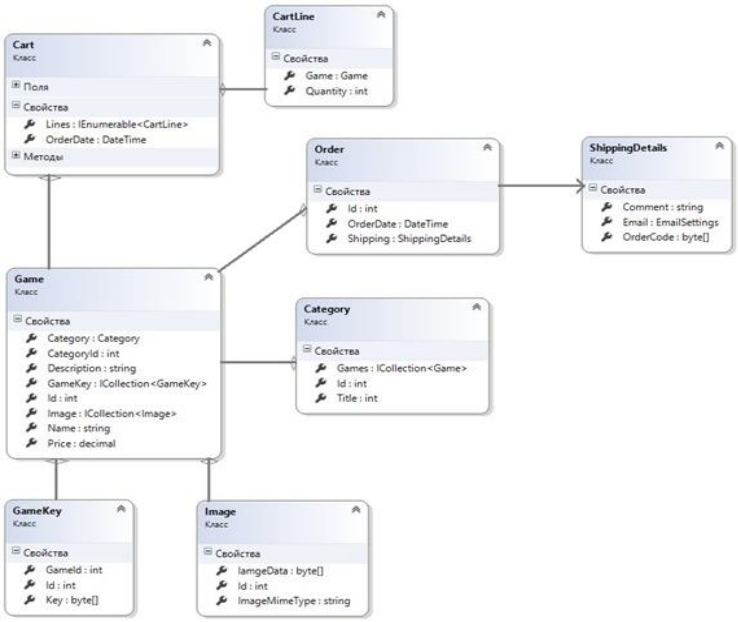


Рис. 2.1. Диаграмма классов

Так же созданы классы *User* и *Role*,как показано на рис. 2.2,для хранения информации о зарегистрированных в системе пользователей (на данный момент зарегистрированным пользователем является только администратор).

Диаграмма взаимодействия между собой администратора и клиента представлена в приложении 8.

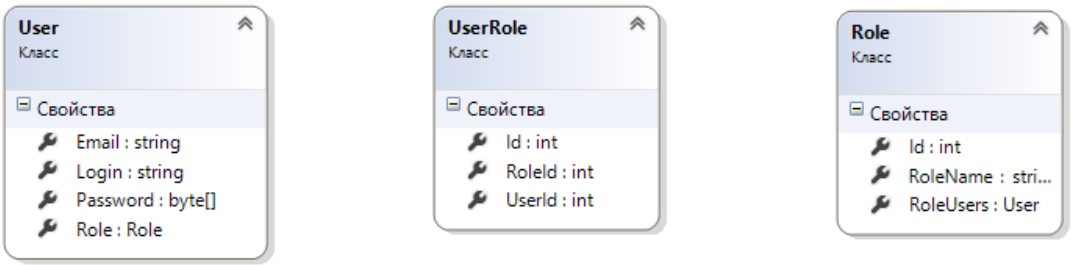


Рис. 2.2. Диаграмма классов пользователей и ролей

Подход *CodeFirst* в *EF* на основе классов, представляющих бизнес-модель, генерирует схему базы данных и создаёт базу данных и её сущности (таблицы, отношения и пр.) при запуске приложения по сгенерированной схеме.

Данный процесс осуществляется по определённым соглашениям, которые автоматически оценивают различные свойства и классы, образующие уровень модели, для того, чтобы выяснить, каким образом информация модели должна быть сохранена, и как отношения между классами модели могут быть эффективно представлены в терминах отношений базы данных.

Например, класс Game отображается в одноименной таблице, а все его свойства представляют собой столбцы в этой таблице. Имена таблиц и столбцов автоматически выводятся из имён классов и их членов.

В основе подхода *CodeFirst* лежит класс *System.Data.Entity.DbContext*, который выступает в качестве шлюза к базе данных и предоставляет все необходимые действия для работы с данными. Для использования класса *DbContext* в *web*-приложении был создан производный от него класс *GameStoreDbContext*, в котором определяются свойства *System.Data.Entity.DbSet<T>*, где *T*– это сущность, которая будет редактироваться и сохраняться в базе данных, в процессе работы *web*-приложения для всех классов, в сохранении и редактировании экземпляров которых нуждается приложение.

В результате была сгенерирована база данных по созданной схеме(полная схема представлена в приложении 1).

2.3 Проектирование методов действий контроллеров

В соответствии с шаблоном MVC, контроллеры располагаются в одном слое, не имеют зависимостей друг от друга, и взаимодействуют только с моделями и представлениями. Поэтому контроллеры с возможными действиями удобно представлены в виде таблицы 2.1:

Таблица 2.1

Методы действий контроллеров

Контроллер	Метод действия	Описание действия
GameController	*List*	Список всех товаров
	GetImage	Страница отдельного товара.
NavigatorController	*Menu*	Список всех категорий.
	Edit	*CRUD* операции для категорий.
	Delete	
	Create	
AccountController	*Register*	Регистрация пользователя.
	LogOn	Вход пользователя.
	LogOff	Выход пользователя.
	PasswordChange	Смена пароля.
CartController	*Checkout*	С помощью системы привязки моделей, определяется пуста ли корзина.
	AddToCart	Добавление товара в корзину.
	RemoveFromCart	Удаление товара из корзины
	Summary	Метод для визуализирования представления, который передаёт в качестве представления данных текущий объект, получаемый с использованием специального метода связывателя модели
	Index	Отображение содержимого корзины
CheckoutController	*Address*	Заполнение информации о доставке
	Payment	Оплата товара
	Complete	Завершение оформления заказа
AdminController	*Index*	Отображение товаров
	Edit	Функции изменения и сохранения работают через перегрузку метода *Edit*

2.4 Сквозная функциональность

Сквозная функциональность – это аспекты дизайна, которые могут применяться ко всем слоям, компонентам и уровням. Это те области, в которых чаще всего делаются ошибки, оказывающие большое влияние на дизайн. Приведем примеры сквозной функциональности:

- аутентификация и авторизация;
- кэширование;
- связь;

- управление конфигурацией;
- управление исключениями;
- логирование;
- валидация.

2.4.1Внедрение зависимостей

Контроллеры в шаблоне проектирования *MVC* предназначены для предварительной подготовки модели с последующей передачей их в представление. Это единственная их ответственность. Контроллеры не являются переиспользуемыми компонентами, поэтому они не должны содержать бизнес-логику. Вся бизнес-логика должна выносится в слой сервисов, в модель данных.

Таким образом получается, что контроллеры имеют внешние зависимости в виде сервисных компонентов модели данных и репозиториев. Для ослабления связи между компонентами воспользуемся Внедрением Зависимостей (англ. *DependencyInjection*).

Используя внедрение зависимостей, объект лишь предоставляет свойство, которое в состоянии хранить ссылку на нужный тип сервиса. А при создании объекта, ссылка на реализацию нужного типа сервиса автоматически устанавливается средствами среды. Происходит «инверсия управления».

Чтобы эффективно управлять разрешением зависимостей извне, обычно используют *контейнер инверсии управления* (*IoCcontainer*). Он позволяет регистрировать зависимые типы и зависимости к ним, строит граф зависимостей, и позволяет создавать объекты с уже разрешенными зависимостями.

Обеспечение работы базовой функциональности *Ninject* осуществляется в три этапа:

- первый этап заключается в подготовке *Ninject* к использованию. Для этого создаётся экземпляр ядра *Ninject*, который представляет собой объект, ответственный за распознавание зависимостей и создание новых объектов. Когда возникает потребность в каком-либо объекте, вместо применения ключевого слова *new* производится обращение к ядру;
- второй этап процесса состоит в конфигурировании ядра *Ninject* с целью предоставления сведений о том, какие объекты реализации должны применяться для каждого интерфейса, с которым придётся работать;
- последний этап – это действительное использование *Ninject*, что делается посредством метода *RegisterServices*() ядра.

Настройка внедрения зависимостей *MVC.*

В результате выполнения трёх шагов, описанных выше, в *Ninject* настраивается информация о том, экземпляр какого класса реализации должен быть создан для удовлетворения запросов интерфейса.

В последующих разделах будет показано, как внедрить *Ninject* в пример приложения *MVC*, что позволит упростить контроллер, расширить влияние библиотеки *Ninject* на всё приложение и вынести конфигурацию за рамки контроллера.

Создание распознавателя зависимостей.

Первое изменение, которое будет внесено, связано с созданием специального распознавателя зависимостей. В инфраструктуре *MVCFramework* распознаватель зависимостей применяется для создания экземпляров классов, необходимых для обслуживания запросов. Создавая специальный распознаватель, мы гарантируем, что *MVCFramework* использует *Ninject* всегда, когда должен создаваться тот или иной объект, включая экземпляры контроллеров.

Регистрация распознавателя зависимостей.

Простого создания реализации интерфейса недостаточно – инфраструктуре *MVCFramework* необходимо также сообщить о необходимости его использования.

Пакеты *Ninject* добавлены с помощью *NuGet*, в папке *App_Start*находится файл с именем*NinjectWebCommon.cs*, в котором определены методы, вызываемые автоматически при запуске приложения; целью является интеграция в жизненный цикл запросов *ASP.NET*.

В метод *RegisterServices*() класса *NinjectWebCommon* добавляется оператор, который создает экземпляр класса *NinjectDependencyResolver* и вызывает статический метод *SetResolver*(), определенный классом *System.Web.Mvc.DependencyResolver*, для регистрации распознавателя в инфраструктуре *MVCFramework*. Результатом данного оператора является создание шлюза между *Ninject* и поддержкой внедрения зависимостей в *MVCFramework*.

Код использования*Ninject*представлен в приложении 2.

2.5Реализация корзины товаров

Механизм добавления товаров в корзину представлен на рис. 2.3:

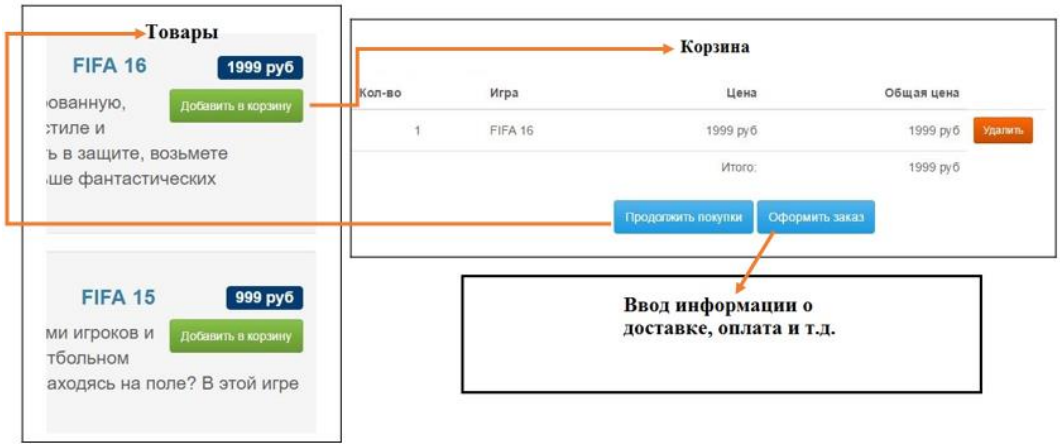


Рис. 2.3. Механизм работы корзины покупателя

Корзина для покупок является частью предметной области приложения, поэтому для представления корзины была создана сущность*Cart* в модели предметной области.

Класс *Cart* использует класс *CartLine*, который отвечает за товар, выбранный пользователем, а также приобретаемое его количество. Были определены методы:

- добавления и удаления элемента из корзины;
- вычисления общей стоимости элементов в корзине;
- очистки корзины путём удаления всех элементов.

Также реализовано свойство, которое позволяет обратиться к содержимому корзины с использованием *IEnumerable*<*CartLine*>. Все это было довольно легко реализовано с помощью кода C# и небольшой доли кода *LINQ*.

Так же для обработки нажатия кнопок «добавить в корзину», «удалить из корзины» создан контроллер *CartController*. В нём реализованы соответствующие методы *AddToCart*()и *RemoveFromCart*(). Их работа обеспечивается, используя средство состояния сеанса *ASP.NET* в методе *GetCart*().

Инфраструктура *ASP.NET* поддерживает удобное средство сеансов, которое использует *cookie*-наборы или переписывание *URL*, чтобы ассоциировать вместе множество запросов от определённого пользователя с целью формирования отдельного сеанса просмотра. С данным средством связано состояние сеанса, позволяющее ассоциировать данные с сеансом. Это идеально подходит для класса *Cart*.

Нужно чтобы каждый пользователь имел собственную корзину, и эта корзина сохранялась между запросами. Данные, связанные с сеансом, удаляются по истечении времени существования сеанса (что обычно происходит, когда пользователь не выдаёт запрос в течение заданного периода), а это значит, что управлять хранением или жизненным циклом объектов *Cart* не понадобится.

Так же придерживаясь концепции, основанной на параметрах методов действий, был создан специальный связыватель модели*CartBModelBinder*путём реализации интерфейса

System.Web.Mvc.IModelBinder, который получает объект *Cart*, содержащийся внутри данных сеанса. В результате чего появилась возможность создавать объект *Cart*и передавать его в виде параметра метода действий класса *CartController*.

Для того чтобы инфраструктура*MVCFramework*использовала класс *CarttModelBinder*, необходимо в методе *Application_Start*(), файла *Global.asax*, добавить реализованный связыватель: *ModelBinders.Binders.Add(typeof(Cart), newCartModelBinder());*

Применение такого специального связывателя модели характеризуется несколькими преимуществами:

- Разделения логики создания *Cart* и логики контроллера, что позволяет изменять способ хранения объектов *Cart* без необходимости в модификации самого контроллера.
- Любойкласс контроллера, который работает с объектами *Cart*, может просто объявить их как параметры метода действия и воспользоваться специальным связывателем модели.

За отображение содержимого корзины отвечает класс модели *CartIndexViewModel.cs*, метод *Index*() класса *CartController.cs* и представление *Index.cshtml*.

В представлении *Index.cshtml*реализован следующий функционал:

- проход по элементам в корзине с добавлением строки для каждого элемента;
- расчёт суммарной стоимости для этой строки;
- итоговая стоимость всех элементов корзины.

Так же был добавлен виджет с информацией о содержимом корзины и реализован в виде действия, вывод которого встраивается в компоновку*Razor*.

Необходимость виджета в том, что без него в корзину можно было попасть только через добавление товара, а теперь в любое время, щёлкнув на него.

Файлы, отвечающие за реализацию работоспособности корзины, представлены в приложении 5.

2.6Подключение платёжной системы

Для подключения платёжной системы был выбран сервис Яндекс-деньги, который предоставляет возможность оплаты с помощью счета в Яндекс-деньгах или с помощью карт *Visa* и *MasterCard*.

Требуется создать платёжную форму, которая по нажатию кнопки отправит на сервис Яндекса, где пользователь произведёт непосредственную оплату, на раннее заданные реквизиты.

После оплаты сервис отправит покупателя на заданную разработчиком страницу, адрес сайта. В случае успешного платежа приходит уведомление по *HTTP*, далее происходит обработка и дальнейшая работа приложения.

В первую очередь происходит подключение *HTTP*-уведомления, как показано на рис. 2.4:

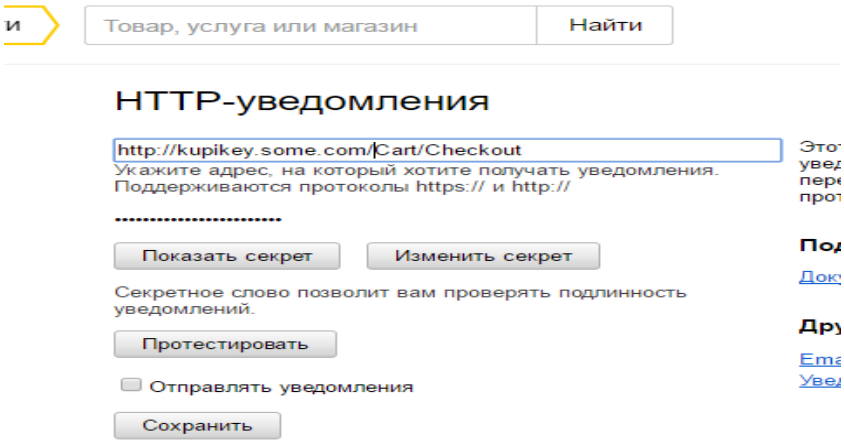


Рис. 2.4. Настройка уведомления на сервисе Яндекса

Для настройки уведомления, требуется указать адрес ресурса, который отвечает за обработку приходящего уведомления от сервиса Яндекс, в данном случае это /*Cart*/*Checkout*.

Секретное слово требуется для проверки уведомлений в целях безопасности.

Для перехода на сервер Яндекса было создано представление для вывода формы с информацией о заказе, как представлено на рис. 2.5:

```
<div>
    <form method="POST" action="https://money.yandex.ru/quickpay/confirm.xml">
        <input name="label" value="@Номер заказа" type="hidden">
        <input name="receiver" value="НОМЕР КОШЕЛЬКА" type="hidden">
        <input name="quickpay-form" value="shop" type="hidden">
        <input type="hidden" name="targets" value="Оплата заказа @Model.OrderId">
        <input name="sum" value="0,01" maxlength="10" data-type="number" type="text"><br /><br />
        <label for="sum">Способ оплаты: </label><br />
        <input name="successURL" value="http://kupikey.somee.com/Cart/Checkout" type="hidden">
        <input name="quickpay-back-url" value="http://kupikey.somee.com/" type="hidden">
    </form>
</div>
```

Рис. 2.5 Форма отправки данных

Для отправки формы нам надо указать на форме ряд параметров:

- *Receiver* – номер Яндекс кошелька;
- *Quickpay* – тип платежа;
- *Targets* – название платежа;
- *Sum* – сумма оплаты заказа;
- *SuccessURL* – переход на указанный адрес в случае успешной оплаты.

От Яндекса в *http*-уведомлении обрабатывается ряд параметров о платеже. Однако, есть одна проблема: к этому же методу может обратиться любой, и послать какие угодно параметры. Для этого нам и нужно секретное слово. Оно указывается в качестве значения строковой переменной *key*. Оно позволяет сгенерировать хэш пароля с использованием алгоритма *sha1*. Затем мы сравниваем оба хэша, и если они равны, то данные добавляем в *БД*.

2.7Обработка заказа

После успешной оплаты заказа,реализованы функциональные возможности для отправки товара покупателю.

Соблюдая принципы модели *MVC*, был определёнинтерфейс для этой функциональности, написана его реализация и связана с помощью контейнера внедрения зависимости (*Ninject*).

Интерфейс *IOrderProcessor*обрабатывает заказ, отправляя его по электронной почте покупателю.

Для отправки электронной почты применялась встроенная поддержка протокола *SMTP*, доступная в библиотеке *.NETFramework*.

Класс *EmailSettings*хранит всю информацию, нужную для отправки сообщений: учётные данные отправителя (сотрудника магазина), почтовый адрес получателя, использование протокола *ssl*для шифрования, порт, используемый для *SMTP* транзакций и пр.

Класс *EmailOrderProcessor*отвечает за выполнение всех настроек, генерации письма и его отправки через *SMTP*-сервер или в файл для удобства отладки.

Экземпляр класса*System.Net.Mail.MailMessage* представляет собой сообщение электронной почты, в конструктор которого передаются все необходимые, полученные ранее настройки, и производится фактическая отправка через *SMPT*сервер.

Файлы, отвечающие за реализацию работоспособности отправки заказа, представлены в приложении 6.

2.8Панель администратора

По общепринятым нормам, администратор должен иметь возможность использования *CRUD* операций через пользовательский интерфейс, для этого была реализована панель администратора.

Для перехода к панели администратора нужно перейти по адресу /*Admin*/*index* и пройти аутентификацию.

Для поддержки средств администрирования был создан *MVC5Controller* с именем *AdminController*.

В конструкторе *AdminController* объявлена зависимость от интерфейса *IGameRepository*, которую *Ninject* будет распознавать при создании экземпляров. Метод действия *Index*() вызывает метод *View*(), чтобы выбрать стандартное представление для действия, передавая ему в качестве модели представления набор товаров из базы данных.

Для отображения списка товаров было создано представление *Index* с вариантом шаблона *List.*

При использовании формирования шаблонов вида *List* среда *VisualStudio* предполагает, что работа производится с последовательностью *IEnumerable* типа модели представления, поэтому можно просто выбрать в списке форму класса в единственном числе.

Шаблонное представление было отредактировано и добавлена необходимая функциональность, добавлены ссылки на кнопки удаления, добавления и редактирования.

В *AdminController*реализован метод действий *Edit*–этот метод ищет товар с идентификатором, соответствующим значению параметра *gameId*, и передаёт его как объект модели представления методу *View*(). Имея метод действий, создано представление для визуализации *Edit.cshtml*.

Так же, в контроллере был создан перегруженный метод *Edit* для сохранения изменений в хранилище данных, после успешной операции происходит перенаправление на страницу товаров, вызывая метод действия *Index*.

Для сохранения изменений, в хранилище товаров был добавлен метод *SaveGame*в класс *EFGameRepository*. Метод сохранял изменения, если *Id*товара был не нулевой, а иначе создавал новый товар.

Функция *Delete*отвечает за удаление товара из корзины.

После фиксации изменений, на странице товаров появится сообщение типа *TempData,* оповещающее о сохранённых изменениях.

Класс хранилища *EntityFrmework, EFGameRepository*отвечает за удаление, добавление данных в контексте данных.

Для использования средств аутентификации используется класс *System.Web.Security.FormsAuthentication*, а такжефильтр*Authorize* (фильтр – это атрибут *.NET,* который можно применять к методу действий или классу контроллера, чтобы изменить поведение инфраструктуры *MVCFramework*).

IAuthProvider– интерфейс поставщика аутентификации, *FormAuthProvider*– реализация данного интерфейса.

Также созданы:

- *LoginViewModel*– класс модели представления;
- *AccountController*–контроллер, обрабатывающий аутентификацию;
- *Login.cshtml*–представление, запрашивающее учётные данные пользователя.

Файлы, отвечающие за реализацию работоспособности панели администратора и проверку учётных данных пользователей, представлены в приложении 2.

3 ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ ПРОГРАММЫ

3.1 Графический интерфейс приложения

При открытии приложения, пользователю представляется главная страница сайта, представленная на рис. 3.1:

Рис.3.1. Внешний вид приложения

Здесь он может выбрать интересующий его жанр игр, прочитать их минимальное описание, ознакомится со стоимостью каждой игры и перейти на следующую страницу списка игр, как представлено на рис.3.2:

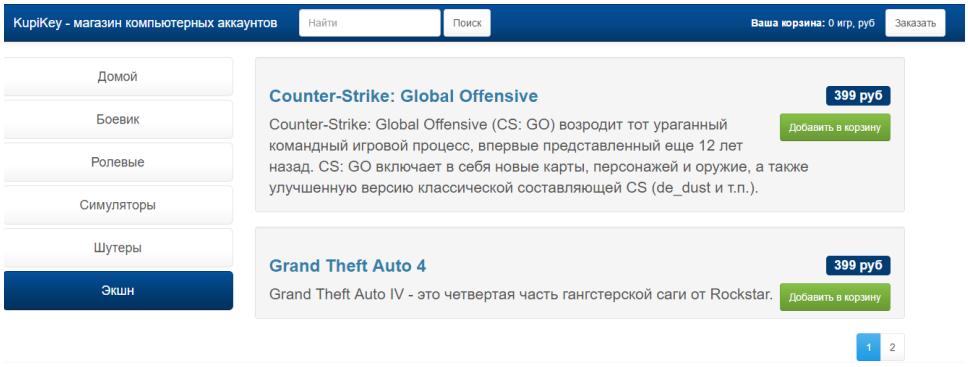


Рис.3.2. Отображение игр по категориям

При добавлении товара в корзину, пользователя перенаправляет на содержимое его корзины, где он имеет возможность редактировать ее

содержимое, ознакомиться с итоговой ценой всех товаров, а также продолжить покупки или перейти к оформлению заказа, как представлено на рис 3.3:

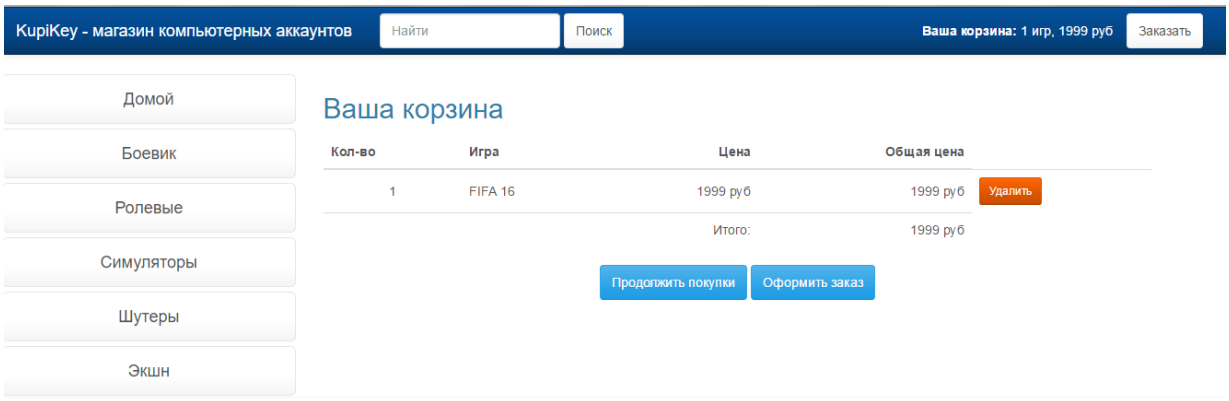


Рис. 3.3. Корзина покупателя

На рис. 3.4 представлена форма оформления заказа. Для покупки товара пользователь должен указать свои имя и почту, при не заполнении полей появится предупреждение.

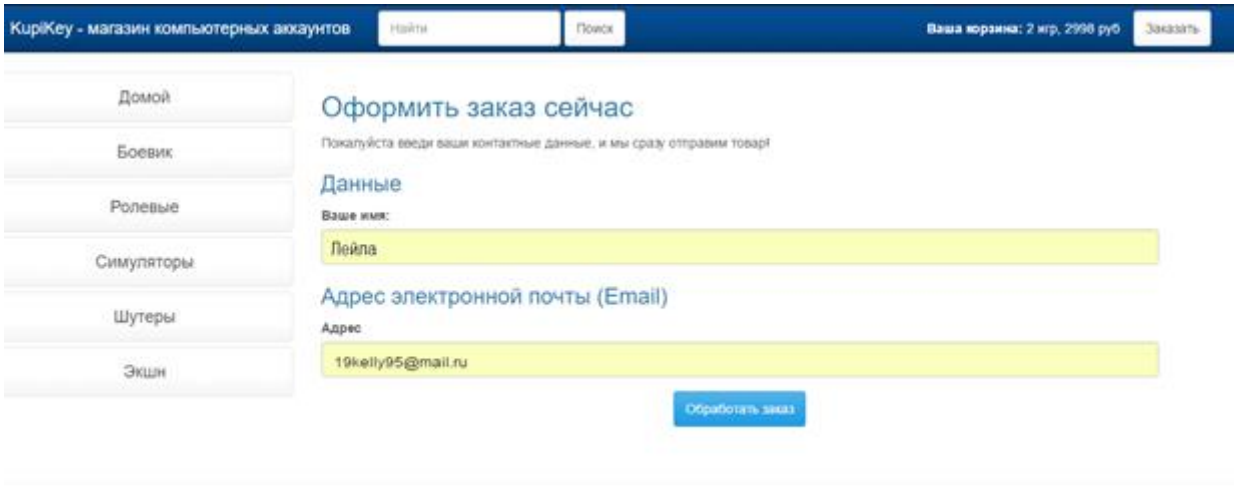


Рис. 3.4. Форма оформления заказа

После нажатия кнопки обработать заказ, пользователя переправляет на страницу сервиса оплаты Яндекс. На форме указывается номер заказа, номер

получателя для перевода средств, и указывается сумма, взятая из итоговой суммы корзины. Нужно указать способ оплаты (карта/Яндекс-деньги) и произвести перевод,как показано на рис. 3.5:

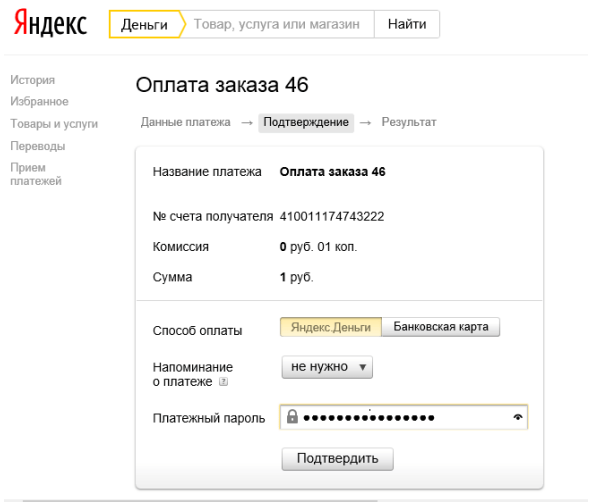


Рис. 3.5. Оплата заказа на сервисе Яндекс

При успешной оплате покупателю на указанную почту придёт оповещение об успешной покупке и код активации купленной игры,как показано на рис. 3.6:

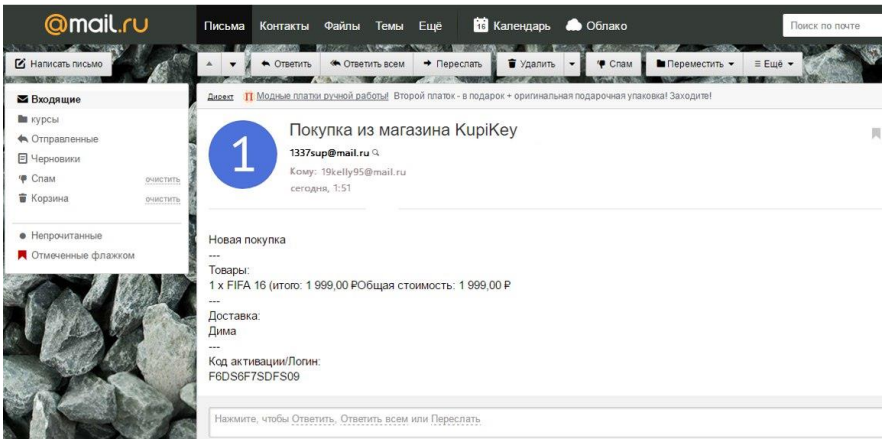


Рис. 3.6. Отправка подтверждения заказа на сайт покупателя

После успешной оплаты покупателя переправляет на указанный разработчиком адрес, в данном случае на страницу успешной оплаты, как указано на рис. 3.7:

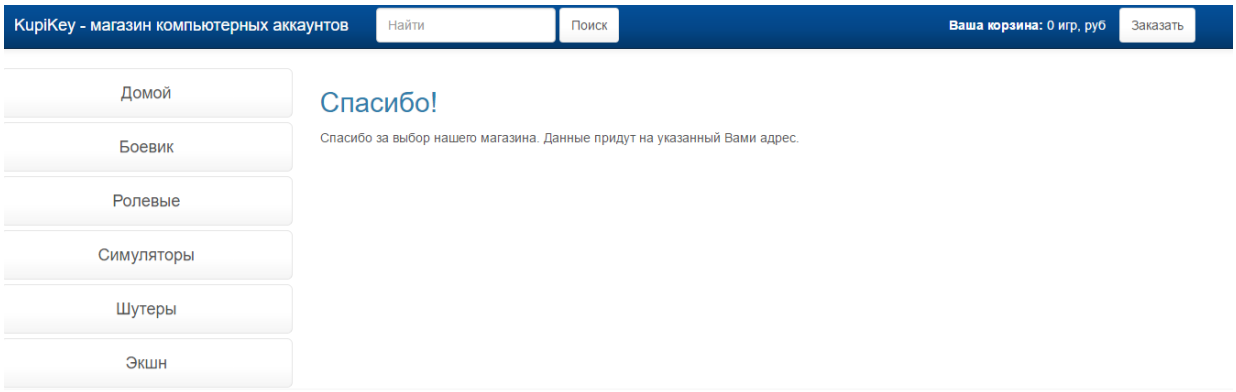


Рис. 3.7. Подтверждение заказа

Как представлено на рис. 3.8, для входа в панель администратора надо перейти на адрес *Admin/Index*, ввести данные администратора и пройти авторизацию:

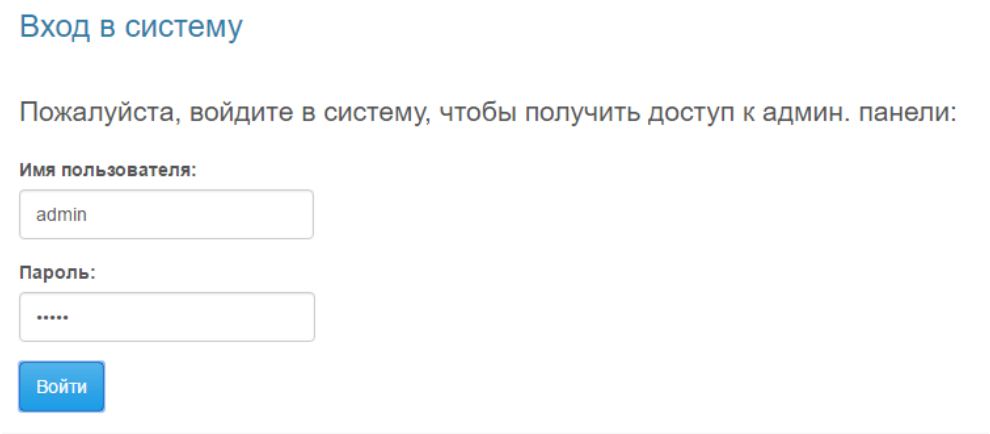


Рис. 3.8. Форма авторизации

При успешной авторизации администратор перейдёт на страницу со списком всех товаров магазина, как представлено на рис. 3.9:

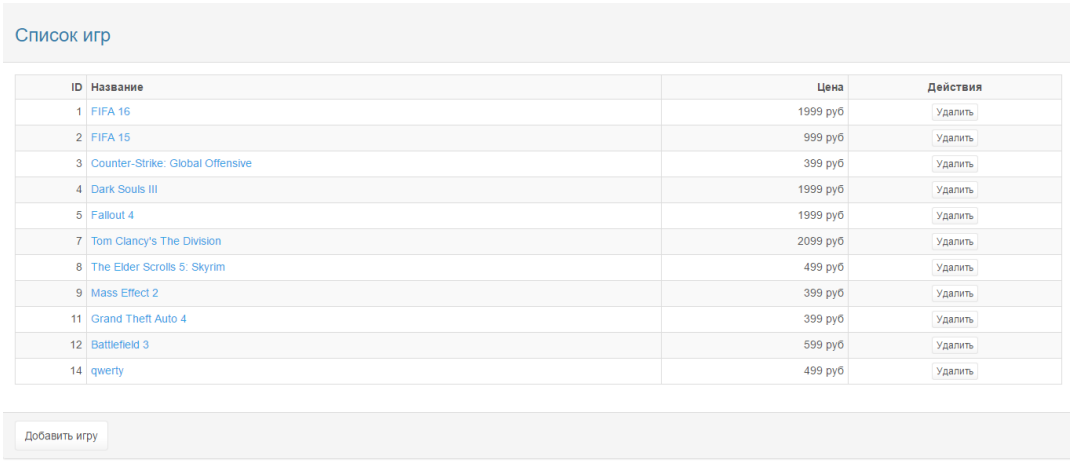


Рис.3.9. Панель администратора

Напротив каждого элемента из списка формируется кнопка «удалить».При нажатии на неё происходит удаление товара из хранилища данных, как показано на рис. 3.10:

Рис.3.10. Удаление игры

Если нажать на кнопку «добавить игру», то пользователя перенаправит на страницу создания нового товара, где ему необходимо заполнить указанные поля и, по желанию, добавить фото из указанного места на компьютере, как показано на рис. 3.11:

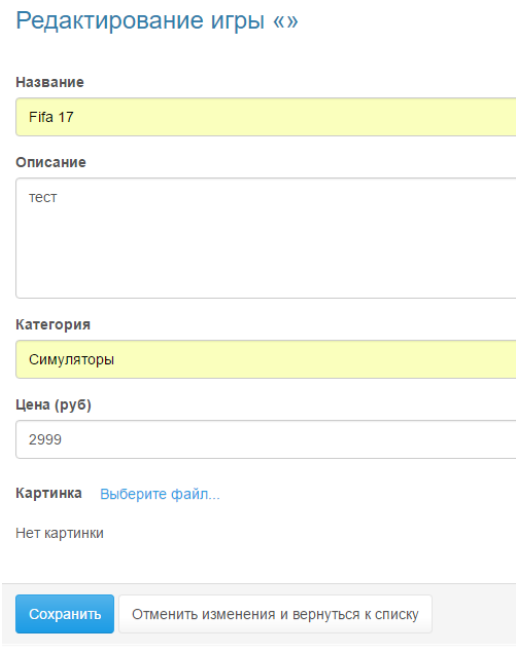


Рис. 3.11. Форма добавления игры

После добавления товара, в список всех товаров будет добавлен новый, а сверху появится сообщение о том, что добавление прошло успешно,как показано на рис. 3.12:

Рис.3.12. Список игр, после добавления новой игры

При нажатии на название игры, произойдёт переход на страницу редактирования указанного товара, как представлено на рис. 3.13, где поля уже будут заполнены имеющейся информацией и показана картинка, если она добавлена:

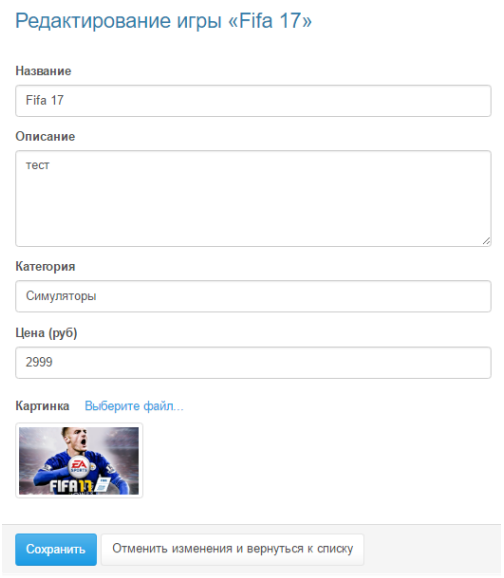


Рис. 3.13. Форма редактирования товара

Также, используя набор инструментов *TwitterBootstrap*и технологию чувствительного дизайна, была проработана мобильная версия приложения.

На рис. 3.14 и рис. 3.15 представлены мобильный вид приложения и мобильный вид корзины:

Рис.3.14. Мобильный вид приложения

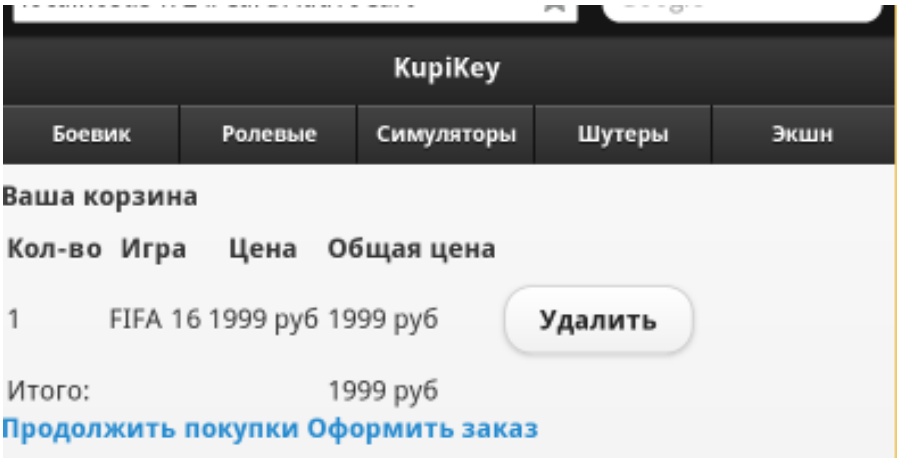


Рис. 3.15.Мобильный вид корзины

На рис. 3.16 и рис. 3.17 представлены мобильный вид оформления заказа и форма аутентификации администратора:

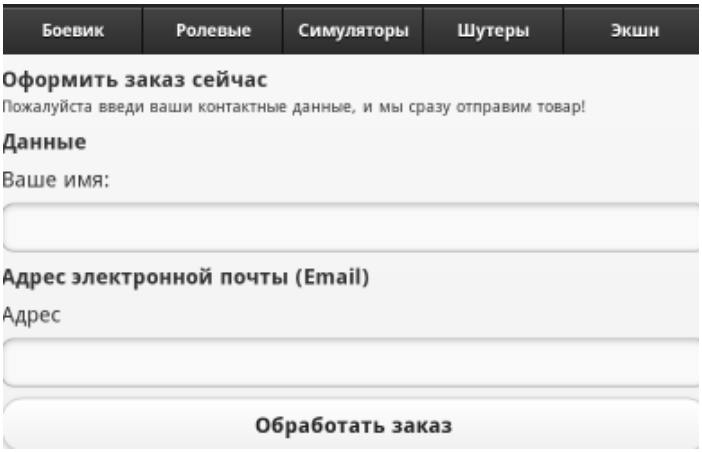


Рис. 3.16. Мобильный вид формы оформления заказа

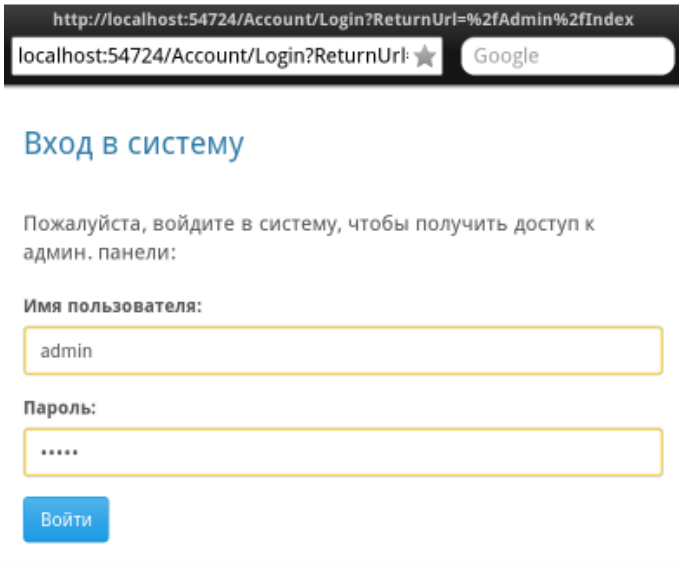


Рис. 3.17. Мобильный вид формы авторизации

На рис. 3.18 и рис. 3.19 представлены мобильный вид формы редактирования списка игр с правами администратора и мобильный вид редактирования товара:

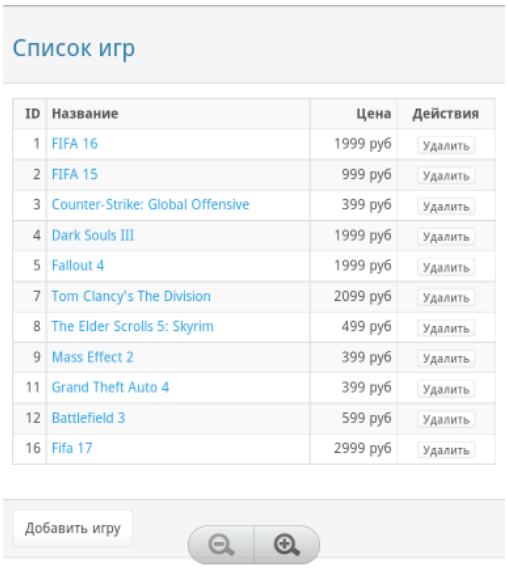


Рис. 3.18. Мобильный вид списка товаров

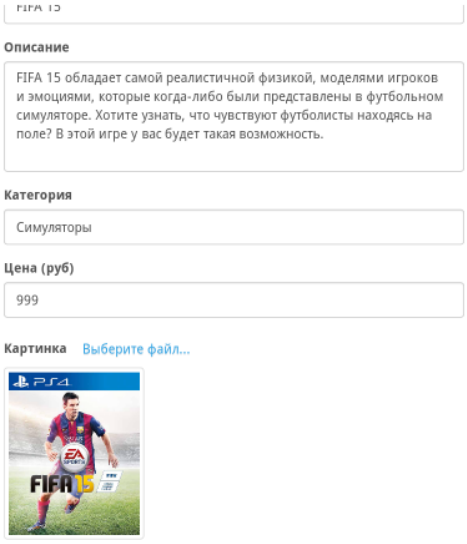


Рис. 3.19. Мобильный вид форма редактирования товаров.

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной работы была спроектирована гибкая архитектура приложения, подготовленная к изменениям и модификациям в будущем. Приложение использует современные технологии, как на клиентской части, так и на серверной.

Рассмотрены проблемы проектирования модели данных и применена методика дизайна архитектуры на основе данных предметной области.

Спроектированный архитектурный каркас позволяет писать слабосвязанные компоненты, которые поддаются как модульному тестированию, так и подходу написания качественного программного обеспечения – разработку через тестирование.

Во время разработки проводилось тестирование создаваемых компонентов.

Изучены технологии привязки моделей и внедрение зависимостей, *FrameworkBootstrap*и движок представление *Razor*.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Adam Frimen «Pro ASP.NET MVC 4» / Adam Frimen - ISBN: 978-5-8459-1867-3, ISBN: 978-1-4302-4236-9, Edition: 4, 16 January;

2. Bootstrap: [сайт] - (URL: https://bootswatch.com/);

3. Eric Evans «Domain-Driven Design: Tackling Complexity in the Heart of Software» / Eric Evans - ISBN-10: 0321125215, ISBN-13: 9780321125217, Edition: 1, 30 August 2012;

4. IoC, DI, IoC-контейнер - Просто о простом / Хабрахабр: [сайт] – (URL: https://habrahabr.ru/post/131993/);

5. Jeffrey Richter «CLR via C#» / Jeffrey Richter - ISBN-13: 978-0735667457, ISBN-10: 0735667454, Edition 4, 25 November, 2012;

6. Jess Chadwick «Programming Razor» / Jess Chadwick - ISBN-13: 978-1449306762, ISBN-10: 1449306764, Edition 1, 25 September, 2012;

7. Автоматическая миграция БД в EntityFramework: [сайт] - (URL: http://habrahabr.ru/post/143292);

8. Введение в Entity Framework 6: [сайт] - (URL: metanit.com/sharp/entityframework/1.1.php);

9. Уведомления о входящем переводе - Технологии Яндекса: [сайт] - (URL: https://tech.yandex.ru/money/doc/dg/reference/notification-p2p-incoming-docpage/)

10. УрокиASP.NETMVC / Хабрахабр: [сайт] - (URL: http://habrahabr.ru/post/175999/);

11. ФреймворкMoq: [сайт] - (URL: http://metanit.com/sharp/mvc5/18.5.php);

Приложение 1

Внедрениезависимостей

```
public class NinjectDependencyResolver : IDependencyResolver
{
	privateIKernel kernel;

	publicNinjectDependencyResolver(IKernelkernelParam)
	{
		kernel = kernelParam;
		AddBindings();
	}

	public object GetService(Type serviceType)
	{
		returnkernel.TryGet(serviceType);
	}

	publicIEnumerable<object>GetServices(Type serviceType)
	{
		returnkernel.GetAll(serviceType);
	}

	private void AddBindings()
	{
		kernel.Bind<IGameRepository>().To<EFGameRepository>();

		EmailSettingsemailSettings = new EmailSettings
		{
			WriteAsFile = bool.Parse(ConfigurationManager
				.AppSettings["Email.WriteAsFile"] ?? "false")
		};

		kernel.Bind<IOrderProcessor>().To<EmailOrderProcessor>()
			.WithConstructorArgument("settings", emailSettings);


	kernel.Bind<IPaymentProcessor>().To<PaymentOrderProcessor>().WithConstructorArgument("settings",
emailSettings); // дляподкюченияоплаты

		kernel.Bind<IAuthProvider>().To<FormAuthProvider>();

	}
}

public static class NinjectWebCommon
   {
private static readonlyBootstrapperbootstrapper = new Bootstrapper();
public static void Start()
      {
DynamicModuleUtility.RegisterModule(typeof(OnePerRequestHttpModule));
DynamicModuleUtility.RegisterModule(typeof(NinjectHttpModule));
bootstrapper.Initialize(CreateKernel);
      }

public static void Stop()
      {
bootstrapper.ShutDown();
      }

private static IKernelCreateKernel()
```

```
        {
var kernel = new StandardKernel();
                        kernel.Bind<Func<IKernel>>().ToMethod(ctx => () => new Bootstrapper().Kernel);
            kernel.Bind<IHttpModule>().To<HttpApplicationInitializationHttpModule>();

RegisterServices(kernel);
return kernel;
        }

private static void RegisterServices(IKernel kernel)
        {
System.Web.Mvc.DependencyResolver.SetResolver(new
GameStore.WebUI.Infrastructure.NinjectDependencyResolver(kernel));
        }
    }
```

Приложение 2

Контроллеры

```
public class NavController : Controller
{
privateIGameRepository repository;

publicNavController(IGameRepository repo)
      {
repository = repo;
      }

publicPartialViewResult Menu(string category = null)
      {
ViewBag.SelectedCategory = category;

IEnumerable<string> categories = repository.Games
            .Select(game =>game.Category)
            .Distinct()
            .OrderBy(x => x);

returnPartialView("FlexMenu", categories);
      }
}

public class GameController : Controller
{
     // GET: Game
privateIGameRepository repository;

publicintpageSize = 2; // количествомоделейнастранице

publicGameController(IGameRepository repo)
      {
repository = repo;
      }

publicViewResult List(string category, int page = 1)
      {
GamesListViewModel model = new GamesListViewModel
         {
            Games = repository.Games
               .Where(p => category == null || p.Category == category)
               .OrderBy(game =>game.GameId)
               .Skip((page - 1) * pageSize)
               .Take(pageSize),
PagingInfo = new PagingInfo
            {
CurrentPage = page,
ItemsPerPage = pageSize,
TotalItems = category == null ?
repository.Games.Count() :
repository.Games.Where(game =>game.Category == category).Count()
            },
CurrentCategory = category

         };
return View(model);
      }

publicFileContentResultGetImage(intgameId)
      {
```

```
            Game game = repository.Games
                .FirstOrDefault(g =>g.GameId == gameId);

if (game != null)
            {
return File(game.ImageData, game.ImageMimeType);
            }
else
            {
return null;
            }
        }
}

public class CartController : Controller
{

privateIGameRepository repository;
privateIOrderProcessororderProcessor;
privateIPaymentProcessorpaymentProcessor;

publicCartController(IGameRepository repo, IOrderProcessor processor, IPaymentProcessorpaymProcessor/*оплата*/)
        {
repository = repo;
orderProcessor = processor;
paymentProcessor = paymProcessor; // оплатачерезсистему
        }

publicViewResult Checkout()
        {
return View(new ShippingDetails());
        }

        [HttpPost]
publicViewResult Checkout(Cart cart, ShippingDetailsshippingDetails)
        {
if (cart.Lines.Count() == 0)
            {
ModelState.AddModelError("", "Извините, вашакорзинапуста!");
            }

if (ModelState.IsValid)
            {
                //paymentProcessor.PaymentOrder(cart, shippingDetails);// оплатачерезсистему
                //View("Payment");
orderProcessor.ProcessOrder(cart, shippingDetails); // доставка
cart.Clear();
return View("Completed");
            }
else
            {
return View(shippingDetails);
            }
        }

publicRedirectToRouteResultAddToCart(Cart cart, intgameId, string returnUrl)
        {
            Game game = repository.Games
                .FirstOrDefault(g =>g.GameId == gameId);

if (game != null)
            {
cart.AddItem(game, 1);
            }
returnRedirectToAction("Index", new { returnUrl });
        }
```

```
publicRedirectToRouteResultRemoveFromCart(Cart cart, intgameId, string returnUrl)
        {
            Game game = repository.Games
                .FirstOrDefault(g =>g.GameId == gameId);

if (game != null)
            {
cart.RemoveLine(game);
            }
returnRedirectToAction("Index", new { returnUrl });
        }
publicPartialViewResult Summary(Cart cart)
        {
returnPartialView(cart);
        }

publicViewResult Index(Cart cart, string returnUrl)
        {
return View(new CartIndexViewModel
            {
                Cart = cart,
ReturnUrl = returnUrl
            });
        }
}

[Authorize]
public class AdminController : Controller
{
IGameRepository repository;

publicAdminController(IGameRepository repo)
        {
repository = repo;
        }

publicViewResult Index()
        {
return View(repository.Games);
        }

publicViewResult Edit(intgameId)
        {
            Game game = repository.Games
                .FirstOrDefault(g =>g.GameId == gameId);
returnView(game);
        }

        // Перегруженная версия Edit() для сохранения изменений
[HttpPost]
publicActionResult Edit(Game game, HttpPostedFileBase image = null)
        {
if (ModelState.IsValid)
            {
if (image != null)
                {
game.ImageMimeType = image.ContentType;
game.ImageData = new byte[image.ContentLength];
image.InputStream.Read(game.ImageData, 0, image.ContentLength);
                }
repository.SaveGame(game);
TempData["message"] = string.Format("Измененияв игре \"{0}\" былисохранены", game.Name);
returnRedirectToAction("Index");
            }
else
```

```
            {
                // Что-то не так со значениями данных
return View(game);
            }
        }

publicViewResult Create()
        {
return View("Edit", new Game());
        }

        [HttpPost]
publicActionResult Delete(intgameId)
        {
            Game deletedGame = repository.DeleteGame(gameId);
if (deletedGame != null)
            {
TempData["message"] = string.Format("Игра \"{0}\" былаудалена",
deletedGame.Name);
            }
returnRedirectToAction("Index");
        }
}


public class AccountController : Controller
{
IAuthProviderauthProvider;
publicAccountController(IAuthProviderauth)
        {
authProvider = auth;
        }

publicViewResult Login()
        {
return View();
        }

        [HttpPost]
publicActionResult Login(LoginViewModel model, string returnUrl)
        {

if (ModelState.IsValid)
            {
if (authProvider.Authenticate(model.UserName, model.Password))
                {
return Redirect(returnUrl ?? Url.Action("Index", "Admin"));
                }
else
                {
ModelState.AddModelError("", "Неправильныйлогинилипароль");
return View();
                }
            }
else{return View();}
        }
}
```

Приложение 3

Репозитории

```
public interface IGameRepository
{
IEnumerable<Game> Games { get; }
voidSaveGame(Game game);
      Game DeleteGame(intgameId);
}

public class EFGameRepository : IGameRepository
{
EFDbContext context = new EFDbContext();

publicIEnumerable<Game> Games
      {
get { return context.Games; }
      }

public void SaveGame(Game game)
      {
if (game.GameId == 0)
context.Games.Add(game);
else
         {
            Game dbEntry = context.Games.Find(game.GameId);
if (dbEntry != null)
            {
dbEntry.Name = game.Name;
dbEntry.Description = game.Description;
dbEntry.Price = game.Price;
dbEntry.Category = game.Category;
dbEntry.ImageData = game.ImageData;
dbEntry.ImageMimeType = game.ImageMimeType;
            }
         }
context.SaveChanges();
      }

public Game DeleteGame(intgameId)
      {
         Game dbEntry = context.Games.Find(gameId);
if (dbEntry != null)
         {
context.Games.Remove(dbEntry);
context.SaveChanges();
         }
returndbEntry;
      }
}
```

Приложение 4

Реализациятовара

```
publicclassGame
{
[HiddenInput(DisplayValue = false)]
publicintGameId { get; set; }

    [Display(Name = "Название")]
    [Required(ErrorMessage = "Пожалуйста, введитеназваниеигры")]
public string Name { get; set; }

    [DataType(DataType.MultilineText)]
    [Display(Name = "Описание")]
[Required(ErrorMessage = "Пожалуйста, введите описание для игры")]
public string Description { get; set; }

    [Display(Name = "Категория")]
    [Required(ErrorMessage = "Пожалуйста, укажитекатегориюдляигры")]
public string Category { get; set; }

    [Display(Name = "Цена (руб)")]
    [Required]
[Range(0.01, double.MaxValue, ErrorMessage = "Пожалуйста, введите положительное значение для цены")]
public decimal Price { get; set; }


public byte[] ImageData { get; set; }
public string ImageMimeType { get; set; }
}

public class GameController : Controller
{
    // GET: Game
privateIGameRepository repository;

publicintpageSize = 2; // количествомоделейнастранице

publicGameController(IGameRepository repo)
    {
repository = repo;
    }

publicViewResult List(string category, int page = 1)
    {
GamesListViewModel model = new GamesListViewModel
        {
            Games = repository.Games
                .Where(p => category == null || p.Category == category)
                .OrderBy(game =>game.GameId)
                .Skip((page - 1) * pageSize)
                .Take(pageSize),
PagingInfo = new PagingInfo
            {
CurrentPage = page,
ItemsPerPage = pageSize,
TotalItems = category == null ?
repository.Games.Count() :
repository.Games.Where(game =>game.Category == category).Count()
            },
CurrentCategory = category
```

```
        };
return View(model);
    }

publicFileContentResultGetImage(intgameId)
    {
        Game game = repository.Games
            .FirstOrDefault(g =>g.GameId == gameId);

if (game != null)
        {
return File(game.ImageData, game.ImageMimeType);
        }
else
        {
return null;
        }
    }
}

public class GamesListViewModel
{
publicIEnumerable<Game> Games { get; set; }

publicPagingInfoPagingInfo { get; set; }

public string CurrentCategory { get; set; }
}
```

Game.cs

```
@using GameStore.WebUI.Models
@using GameStore.WebUI.HtmlHelpers
@model GamesListViewModel

@{
ViewBag.Title = "Товары";
}

@foreach (var p in @Model.Games)
{
@Html.Partial("GameSummary", p)
}

<div class="btn-group pull-right">
@Html.PageLinks(Model.PagingInfo, x =>Url.Action("List",
    new { page = x, category = Model.CurrentCategory }))
</div>
```

Приложение 5

Реализация корзины

```
publicclassCart
{
private List<CartLine>lineCollection = new List<CartLine>();

public void AddItem(Game game, int quantity)
      {
CartLine line = lineCollection
            .Where(g =>g.Game.GameId == game.GameId)
            .FirstOrDefault();

if (line == null)
         {
lineCollection.Add(new CartLine
            {
               Game = game,
               Quantity = quantity
            });
         }
else
         {
line.Quantity += quantity;
         }
      }

public void RemoveLine(Game game)
      {
lineCollection.RemoveAll(l =>l.Game.GameId == game.GameId);
      }

public decimal ComputeTotalValue()
      {
returnlineCollection.Sum(e =>e.Game.Price * e.Quantity);

      }
public void Clear()
      {
lineCollection.Clear();
      }

publicIEnumerable<CartLine> Lines
      {
get { return lineCollection; }
      }
}

public class CartLine
{
public Game Game { get; set; }
publicint Quantity { get; set; }
}

public class CartController : Controller
{

privateIGameRepository repository;
privateIOrderProcessororderProcessor;
privateIPaymentProcessorpaymentProcessor;

publicCartController(IGameRepository repo, IOrderProcessor processor, IPaymentProcessorpaymProcessor/*оплата*/)
```

```
        {
repository = repo;
orderProcessor = processor;
paymentProcessor = paymProcessor; // оплатачерезсистему
        }

publicViewResult Checkout()
        {
return View(new ShippingDetails());
        }

        [HttpPost]
publicViewResult Checkout(Cart cart, ShippingDetailsshippingDetails)
        {
if (cart.Lines.Count() == 0)
            {
ModelState.AddModelError("", "Извините, вашакорзинапуста!");
            }
if (ModelState.IsValid)
            {
                //paymentProcessor.PaymentOrder(cart, shippingDetails);
// оплатачерезсистему
//View("Payment");
orderProcessor.ProcessOrder(cart, shippingDetails); // доставка
cart.Clear();
return View("Completed");
            }
else
            {
return View(shippingDetails);
            }
        }

publicRedirectToRouteResultAddToCart(Cart cart, intgameId, string returnUrl)
        {
            Game game = repository.Games
                .FirstOrDefault(g =>g.GameId == gameId);

if (game != null)
            {
cart.AddItem(game, 1);
            }
returnRedirectToAction("Index", new { returnUrl });
        }

publicRedirectToRouteResultRemoveFromCart(Cart cart, intgameId, string returnUrl)
        {
            Game game = repository.Games
                .FirstOrDefault(g =>g.GameId == gameId);
if (game != null)
            {
cart.RemoveLine(game);
            }
returnRedirectToAction("Index", new { returnUrl });
        }
publicPartialViewResult Summary(Cart cart)
        {
returnPartialView(cart);
        }

publicViewResult Index(Cart cart, string returnUrl)
        {
return View(new CartIndexViewModel
            {
                Cart = cart,
ReturnUrl = returnUrl
```

```
        });
    }
}

public class CartModelBinder : IModelBinder
{
privateconst string sessionKey = "Cart";

public object BindModel(ControllerContextcontrollerContext,
ModelBindingContextbindingContext)
    {
        // Получитьобъект Cart изсеанса
        Cart cart = null;
if (controllerContext.HttpContext.Session != null)
        {
cart = (Cart)controllerContext.HttpContext.Session[sessionKey];
}

        // Создать объект Cart если он не обнаружен в сеансе
if (cart == null)
        {
cart = new Cart();
if (controllerContext.HttpContext.Session != null)
            {
controllerContext.HttpContext.Session[sessionKey] = cart;
}
        }

        // Возвратить объект Cart
returncart;
}
}


public class CartIndexViewModel
{
public Cart Cart { get; set; }
public string ReturnUrl { get; set; }
}
```

Checkout.cshtml

```
@model GameStore.Domain.Entities.ShippingDetails

@{
ViewBag.Title = "KupiKey: формазаказа";
}

<h2>Оформить заказ сейчас</h2>
<p>Пожалуйста введи ваши контактные данные, и мы сразу отправим товар!</p>


@using (Html.BeginForm())
{
@Html.ValidationSummary();
<h3>Данные</h3>
<div class="form-group">
<label>Вашеимя:</label>
    @Html.TextBoxFor(x =>x.Name, new { @class = "form-control" })
</div>

<h3>Адрес электронной почты (Email)</h3>
foreach (var property in ViewData.ModelMetadata.Properties)
  {
if (property.PropertyName != "Name" &&property.PropertyName != "GiftWrap")
    {
```

```
<div class="form-group">
<label>@(property.DisplayName ?? property.PropertyName)</label>
@Html.TextBox(property.PropertyName, null, new { @class = "form-control" })
</div>
        }
    }

<div class="text-center">
<input class="btnbtn-primary" type="submit" value="Обработатьзаказ" />
</div>
<iframeframeborder="0" allowtransparency="true" scrolling="no"
src="https://money.yandex.ru/embed/small.xml?account=410014310415262&quickpay=small&yamon
ey-payment-type=on&button-text=02&button-size=l&button-
color=orange&targets=%D0%9E%D0%BF%D0%BB%D0%B0%D1%82%D0%B0+%D1%82%D0%BE%D0%B2%D0%B0%D1%80%
D0%BE%D0%B2&default-
sum=1999&mail=on&successURL=http%3A%2F%2Flocalhost%3A54724%2FCart%2FCheckout" width="196"
height="54"></iframe>
}
```

Completed.cshtml

```
@{
ViewBag.Title = "Заказ обработан";
}

<h2>Спасибо!</h2>
<p>Спасибо за выбор нашего магазина. Данные придут на указанный Вами адрес.</p>
```

Index.cshtml

```
@model GameStore.WebUI.Models.CartIndexViewModel

@{
ViewBag.Title = "KupiKey: вашакорзина";
}

<style>
    #cartTable td {
vertical-align: middle;
    }
</style>

<h2>Вашакорзина</h2>

<table id="cartTable" class="table">
<thead>
<tr>
<th>Кол-во</th>
<th>Игра</th>
<th class="text-right">Цена</th>
<th class="text-right">Общаяцена</th>
</tr>
</thead>
<tbody>
@foreach (var line in Model.Cart.Lines)
        {
<tr>
<td class="text-center">@line.Quantity</td>
<td class="text-left">@line.Game.Name</td>
<td class="text-right">@line.Game.Price.ToString("# руб")</td>
<td class="text-right">
@((line.Quantity * line.Game.Price).ToString("# руб"))
</td>
<td>
@using (Html.BeginForm("RemoveFromCart", "Cart"))
```

```
                    {
@Html.Hidden("GameId", line.Game.GameId)
@Html.HiddenFor(x =>x.ReturnUrl)
<input class="btnbtn-smbtn-warning" type="submit" value="Удалить" />
                    }
</td>
</tr>
        }
</tbody>
<tfoot>
<tr>
<td colspan="3" class="text-right">Итого:</td>
<td class="text-right">
@Model.Cart.ComputeTotalValue().ToString("# руб")
</td>
</tr>
</tfoot>
</table>

<div class="text-center">
<a class="btnbtn-primary" href="@Model.ReturnUrl">Продолжитьпокупки</a>
@Html.ActionLink("Оформитьзаказ", "Checkout", null, new { @class = "btnbtn-primary" })
</div>
```

Payment.cshtml

```
@model GameStore.Domain.Entities.ShippingDetails
@{
ViewBag.Title = "Страницаоплаты";
    Layout = "~/Views/Shared/_AdminLayout.cshtml";
}

<h2>Оплата товара</h2>

<div>
<div>
<form method="POST" action="https://money.yandex.ru/quickpay/confirm.xml">
<input name="label" value="@Model.OrderID" @*Номерзаказа*@ type="hidden">
<input name="receiver" value="НОМЕРКОШЕЛЬКА" type="hidden">
<input name="quickpay-form" value="shop" type="hidden">
<input type="hidden" name="targets" value="Оплатазаказа@Model.OrderID">
<input name="sum" value="@Model.TotalPrice" @*Сумма*@maxlength="10" data-type="number"
type="text"><br /><br />
<label for="sum">Способоплаты: </label><br />
<input name="successURL" value="http://kupikey.somee.com/Cart/Checkout" type="hidden">
<input name="quickpay-back-url" value="http://kupikey.somee.com/" type="hidden">
</form>
</div>
```

Summary.cshtml

```
@model GameStore.Domain.Entities.Cart

<div class="navbar-right hidden-xs">
@Html.ActionLink("Заказать", "Index", "Cart",
new { returnUrl = Request.Url.PathAndQuery },
new { @class = "btnbtn-default navbar-btn" })
</div>

<div class="navbar-right visible-xs">
```

```
<a href=@Url.Action("Index", "Cart", new { returnUrl = Request.Url.PathAndQuery })
class="btnbtn-default navbar-btn">
<span class="glyphiconglyphicon-shopping-cart"></span>
</a>
</div>

<div class="navbar-text navbar-right">
<b class="hidden-xs">Вашакорзина:</b>
@Model.Lines.Sum(x =>x.Quantity) игр,
@Model.ComputeTotalValue().ToString("# руб")
</div>
```

Приложение 6

Реализация обработки заказа

```
publicinterfaceIOrderProcessor
{
voidProcessOrder(Cartcart, ShippingDetailsshippingDetails); // отправканапочтупокупателя
}

public interface IPaymentProcessor
{
voidPaymentOrder(Cart cart, ShippingDetailsshippingDetails);// оплататовара
}

public class EmailSettings
    {
public string MailToAddress = "subbotin.dmitriy@list.ru"; //кому
public string MailFromAddress = "1878dimk@mail.ru"; // откого
publicboolUseSsl = true;
public string Username = "1878dimk@mail.ru";
public string Password = "ilovefencing1994";
public string ServerName = "smtp.mail.ru";
publicintServerPort = 587;
publicboolWriteAsFile = true;
public string FileLocation = @"d:\game_store_emails";
    }

public class EmailOrderProcessor : IOrderProcessor
    {
privateEmailSettingsemailSettings;

publicEmailOrderProcessor(EmailSettings settings)
        {
emailSettings = settings;
        }

public void ProcessOrder(Cart cart, ShippingDetailsshippingInfo)
        {
using (varsmtpClient = new SmtpClient())
            {
smtpClient.UseDefaultCredentials = false;
smtpClient.Credentials = new NetworkCredential(emailSettings.Username,
emailSettings.Password);
smtpClient.EnableSsl = emailSettings.UseSsl;
smtpClient.Host = emailSettings.ServerName;
smtpClient.Port = emailSettings.ServerPort;

if (emailSettings.WriteAsFile)
                {
smtpClient.DeliveryMethod
                        = SmtpDeliveryMethod.Network; //
Можнопереключитьдялсохраненияписемвфайл
smtpClient.PickupDirectoryLocation = emailSettings.FileLocation;
smtpClient.EnableSsl = true;
                }

StringBuilder body = new StringBuilder()
                    .AppendLine("Новаяпокупка")
                    .AppendLine("---")
                    .AppendLine("Товары:");

foreach (var line in cart.Lines)
```

```
                {

var subtotal = line.Game.Price * line.Quantity;
body.AppendFormat("{0} x {1} (итого: {2:c}",
line.Quantity, line.Game.Name, subtotal);
}

body.AppendFormat("Общаястоимость: {0:c}", cart.ComputeTotalValue())
.AppendLine("")
                    .AppendLine("---")
                    .AppendLine("Доставка:")
                    .AppendLine(shippingInfo.Name)
                    .AppendLine("---")
                    .AppendLine("Кодактивации/Логин:")
                    .AppendLine("F6DS6F7SDFS09");

MailMessagemailMessage = new MailMessage(
emailSettings.MailFromAddress,    // Откого
shippingInfo.Email,                 // Кому
"Покупка из магазина KupiKey",           // Тема
body.ToString());                        // Телописьма

if (emailSettings.WriteAsFile)
                {
mailMessage.BodyEncoding = Encoding.UTF8;
                }

smtpClient.Send(mailMessage);
            }
        }
}

public class ShippingDetails
{
        [Required(ErrorMessage = "Укажитекаквасзовут")]
public string Name { get; set; }

[Required(ErrorMessage = "Вставьте адрес электронной почты")]
[Display(Name = "Адрес")]
public string Email { get; set; }
}
```

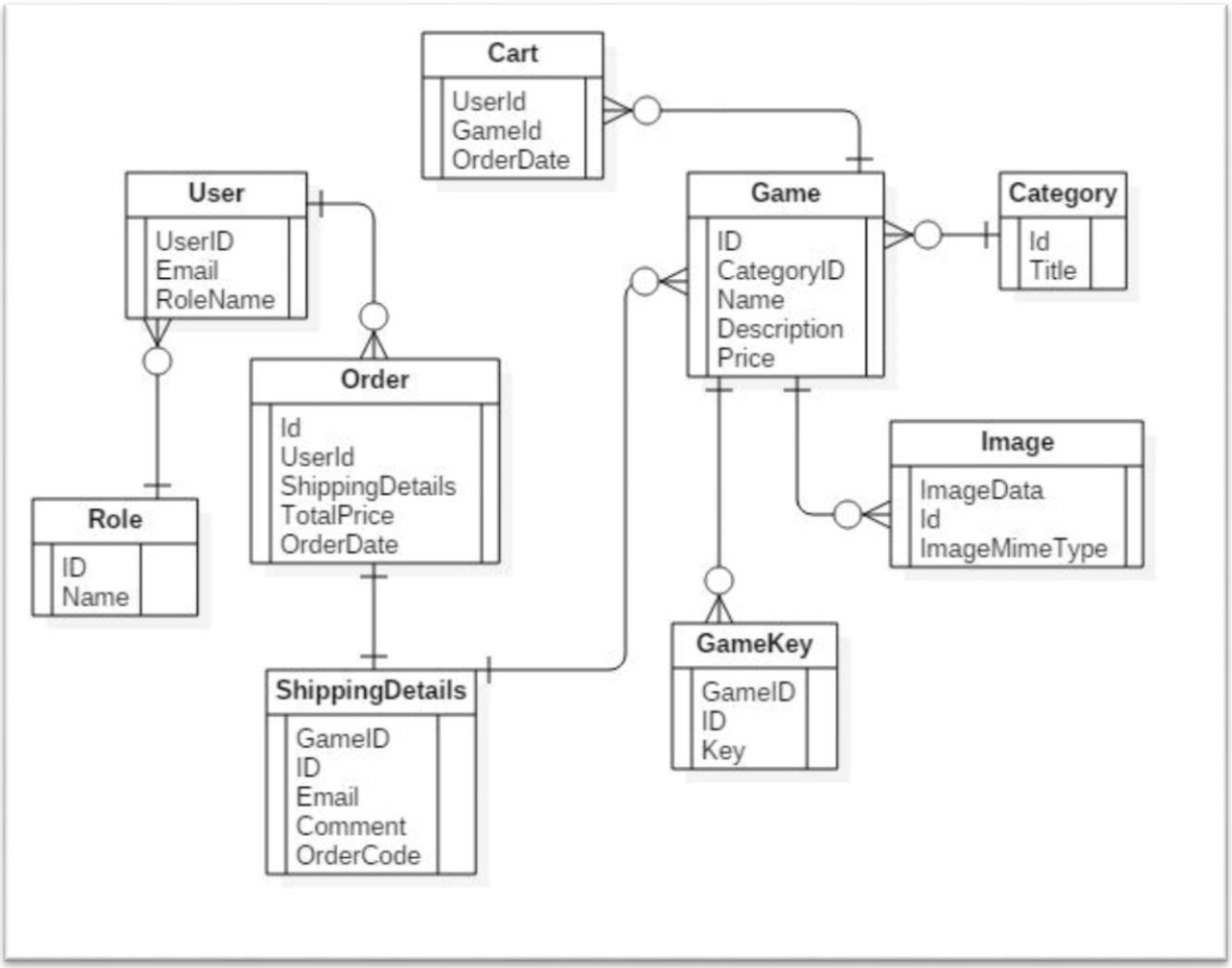
GameSummary.cshtml

```
@model GameStore.Domain.Entities.Game

<div class="well">
 @if (Model.ImageData != null)
    {
<div class="pull-left" style="margin-right: 10px">
<img class="img-thumbnail" width="75" height="75"
src="@Url.Action("GetImage", "Game",
new { Model.GameId })" />
</div>
    }

<h3>
<strong>@Model.Name</strong>
<span class="pull-right label label-info">@Model.Price.ToString("# руб")</span>
</h3>
@using (Html.BeginForm("AddToCart", "Cart"))
    {
<div class="pull-right">
@Html.HiddenFor(x =>x.GameId)
```

```
@Html.Hidden("returnUrl", Request.Url.PathAndQuery)
<input type="submit" class="btnbtn-success" value="Добавитьвкорзину" />
</div>
    }
<span class="lead">@Model.Description</span>
</div>
```

Приложение 7

Физическая модель базы данных



Приложение 8

USE CASE диаграмма