

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра математического обеспечения ЭВМ

**Разработка web-приложения для автоматизации покупки
товаров через Интернет**

Бакалаврская работа
Направление 01.03.02 Прикладная математика и информатика
Профиль Системное программирование и компьютерные технологии

Допущено к защите ГЭК _____

Зав. кафедрой _____
(подпись)

д. ф. – м. н. доц. Махортов С. Д.

Обучающийся _____
(подпись)

Субботин Д.А.

Руководитель _____
(подпись)

Коваленко М. О.

Оглавление

| | |
|--|----|
| Введение | 3 |
| Постановка задачи | 4 |
| Теоретическая часть создания web-приложения | 5 |
| Архитектура web-приложения | 5 |
| Создание web-приложения | 8 |
| Выбор языка программирования | 8 |
| Выбор технологии для создания web-приложения | 9 |
| Архитектура REST | 11 |
| IoC-контейнер Ninject | 12 |
| Используемые фреймворки | 13 |
| Работа с базой данных | 16 |
| Система контроля версий | 18 |
| Практическая реализация web-приложения | 19 |
| Используемые инструменты | 19 |
| Проектирование архитектуры приложения | 20 |
| Проектирование методов действий контроллеров | 24 |
| Сквозная функциональность | 26 |
| Внедрение зависимостей | 26 |
| Реализация корзины товаров | 29 |
| Подключение платёжной системы | 32 |
| Обработка заказа | 34 |
| Панель администратора | 35 |
| Графический интерфейс приложения | 37 |
| Заключение | 49 |
| Список литературы | 50 |
| Приложение 1. Внедрение зависимостей | 51 |
| Приложение 2. Контроллеры | 53 |
| Приложение 3. Репозитории | 57 |
| Приложение 4. Реализация товара | 58 |
| Приложение 5. Реализация корзины | 60 |
| Приложение 6. Реализация обработки заказа | 65 |

Введение

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

С ростом активного использования веб-приложения начинают возникать вопросы о его поддержке, добавлении нового функционала к существующему, поэтому необходимо заранее предусматривать такую архитектуру веб-приложения, которую было бы легко сопровождать в дальнейшем – исправлять существующие проблемы и добавлять новую функциональность. Для этого архитектура должна быть слабосвязанной и составляющие веб-приложения не имели жёстко закодированных зависимостей друг от друга, а имели лишь представления того, что они должны делать. Таким образом, изменения в одной части приложения не будут затрагивать другую часть приложения.

В данной работе рассмотрен проект архитектуры на примере создания веб-приложения для автоматизации заказов через Интернет-магазин.

Постановка задачи

Целью работы является создание *web*-приложения для автоматизации продажи игровых аккаунтов, ключей через Интернет. Эту задачу было бы правильно разделить на несколько подзадач:

- 1) Ознакомление с различными технологиями разработки *web*-приложений.
- 2) Разработка структуры приложения и базы данных для хранения информации о продукции магазина, покупках и покупателях.
- 3) Создание *web*-приложения:
 - Реализация каталога товаров разделённых по категориям;
 - Реализация корзины пользователя;
 - Предоставление информации о возможности оплаты;
- 4) Организация способа оплаты и предоставления оплаченного товара.

Теоретическая часть создания web-приложения

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 специальное приложение — 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 komponует ответ на основе исполнения программного кода, работающего на стороне сервера, *Web*-формы, страницы *HTML*, другого содержимого, включая графические файлы. В результате, как уже было сказано, 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 и т.д.

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 *Hyper Text Transfer Protocol*, «протокол передачи гипертекста». В соответствии со спецификацией *OSI*, *HTTP* является протоколом прикладного (верхнего, 7-го) уровня. Актуальная на данный момент версия протокола, *HTTP* 1.1, описана в спецификации *RFC* 2616.

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 *URI HTTP*/Версия, где:

- Метод - название запроса, одно слово заглавными буквами.
- *URI (Uniform Resource Identifier* - 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*.

Создание web-приложения

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, как принято есть и достоинства и недостатки, но выбор пал на уже знакомый язык *C#*.

C# — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой 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 ИТ-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программ до крупных веб-порталов и веб-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 *Microsoft Visual Studio .NET* в феврале 2002 года. Текущей версией языка является версия *C# 6.0*, которая вышла в 20 июля 2015 года вместе с *Visual Studio 2015*.

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, но в тоже время гибких, масштабируемых и расширяемых приложений. И *C#* продолжает активно развиваться, и с каждой новой версией появляется все больше интересных функциональности, как, например, лямбды, динамическое связывание, асинхронные методы и т.д.

Плюсы:

- Мощный инструмент для создания безопасных и мощных приложений, выполняемых в среде *.NET Framework*;
- 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.

Выбор технологии для создания web-приложения

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

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

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 *Web Forms*. В последующих разделах будет показано, каким образом эта новая платформа позволила преодолеть ограничения *Web Forms* и вновь вывести *ASP.NET* на передовой уровень.

Важно различать архитектурный шаблон *MVC* и инфраструктуру *ASP.NET MVC Framework*. Шаблон *MVC* далеко не нов (его появление датируется 1978 г. и связано с проектом *Smalltalk* в *Xerox PARC*), но в наши дни он завоевал огромную популярность в качестве шаблона для веб-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 обновленное представление пользователю. Затем цикл повторяется. Это хорошо укладывается в схему веб-приложений, предоставляемых в виде последовательностей запросов и ответов *HTTP*.
- Веб-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.NET MVC Framework* реализует шаблон *MVC* и при этом обеспечивает существенно улучшенное разделение ответственности. На самом деле в *ASP.NET MVC* внедрён современный вариант *MVC*, который особенно хорошо подходит для веб-приложений.

За счёт принятия и адаптации шаблона *MVC* инфраструктура *ASP.NET MVC Framework* составляет сильную конкуренцию *Ruby on Rails* и аналогичным платформам, выводя модель *MVC* в авангард развития мира *.NET*. Обобщая опыт и наиболее рекомендуемые приёмы, обнаруженные разработчиками, которые используют другие платформы, *ASP.NET MVC* 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.NET MVC Framework*, 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.NET MVC* привлекает простота интеграции с фреймворками и тем, что *MVC* по умолчанию использует *REST*-подход (*Representational state transfer*) для *URL*-адресов (*Uniform Resource Locator*).

Архитектура REST

REST— это стиль архитектуры программного обеспечения для распределённых систем, таких как *World Wide Web*, который, как правило, используется для построения веб-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 однозначно определяется глобальным идентификатором, таким как *URL*. Каждая *URL* в свою очередь имеет строго заданный формат.

Web-сервис – это приложение работающее в *World Wide Web* и доступ к которому предоставляется по *HTTP*-протоколу, а обмен информации идёт с

помощью формата *XML*. Следовательно, формат данных передаваемых в теле запроса будет всегда *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 о объекте.
- *PUT /info/* или *POST /info/ (Create)* – создаёт новый объект. 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}*.

IoC-контейнер Ninject

***Inversion of Control* (инверсия управления)** — это некий абстрактный принцип, набор рекомендаций для написания слабо связанного кода. Суть которого в том, что каждый компонент системы должен быть как можно более изолированным от других, не полагаясь в своей работе на детали конкретной реализации других компонентов.

***Dependency Injection* (внедрение зависимостей)** — это одна из реализаций этого принципа (помимо этого есть ещё *Factory Method*, *Service Locator*).

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 *Ninject*.

Используемые фреймворки

Moq — известная библиотека с открытым исходным кодом, которая была разработана с использованием возможностей C# 3.0 (деревья выражений *LINQ* и лямбда-выражения). Одним 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 вам методы и свойства с использованием лямбда-выражений.

Bootstrap - свободный набор инструментов для создания сайтов и *web*-приложений. Включает в себя *HTML* и *CSS* шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-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 *Twitter Bootstrap*;

- Совместимость с браузерами — *Twitter Bootstrap* совместим с *Mozilla Firefox*, *Yandex Browser*, *Google Chrome*, *Safari*, *Internet Explorer*, *Microsoft Edge* и *Opera*;
- Открытое программное обеспечение — особенность *Twitter Bootstrap*, 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 140px 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 — Описания 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 вставляют 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 `@`.

Работа с базой данных

Поскольку *ASP.NET MVC* 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*, *LINQ to SQL*, *ADO.NET Entity Framework* и *NHibernate*.

Для базы данных будет применяться *SQL Server*, а доступ к ней будет осуществляться с помощью *Entity Framework (EF)* - объектно-ориентированная технология доступа к данным, является *object-relational mapping (ORM)* решением для *.NET Framework* от *Microsoft*. Предоставляет возможность взаимодействия с объектами как посредством *LINQ* в виде *LINQ to Entities*, так и с использованием *Entity SQL*. Для облегчения построения web-решений используется технология *ADO.NET Data Services (Astoria)*, позволяющая строить многоуровневые приложения, реализуя шаблон проектирования *MVC*.

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

- Подход **Model-First**, впервые появившийся в версии *Entity Framework 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 *Entity Framework*. Проектирование модели происходит в графическом дизайнере *EDM* среды *Visual Studio*.
- Подход **Database-First**, появившийся вместе с *Entity Framework*, 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 - *Entity Framework* 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 *Entity Framework* 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 *Entity Framework* 4.1, обычно используется, когда у вас есть уже существующее приложение, содержащее модель данных. Эта модель, как правило, описывается с помощью нескольких классов и кода взаимодействия между этими классами. Теперь можно сначала написать код, описывающий классы-модели, а потом фреймворк автоматически создаст 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.

Система контроля версий

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

Version Control System (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 С. Проект был создан Линусом Торвальдсом в 2005 году для управления разработкой ядра Linux и, как и *GNU/Linux*, является свободным программным обеспечением (ПО), при этом стороннее использование подчиняется лицензии GNU GPL версии 2. Вкратце 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*, *GNU Core Utilities*, *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.

Практическая реализация web-приложения

Используемые инструменты

В качестве архитектурного каркаса web-приложения используется *ASP.NET MVC 5 Framework*.

Хранилищем данных выступит *Microsoft SQL Server*.

В качестве *ORM* фреймворка применяется *Microsoft Entity Framework*.

Для разработки web-приложения использовалось следующее программное обеспечения:

- 1) *Visual Studio 2015 Professional ver.14.0*
 - a. *Microsoft .NET Framework ver.4.6*
- 2) *Microsoft SQL Server Management Studio 2014 ver.12.0*
- 3) *Git ver.2.8.3*

Проектирование архитектуры приложения

При разработке, решение обычно состоит из одного проекта, но, как правило, 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 *Entity Framework*.

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

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

Domain-driven design - 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:

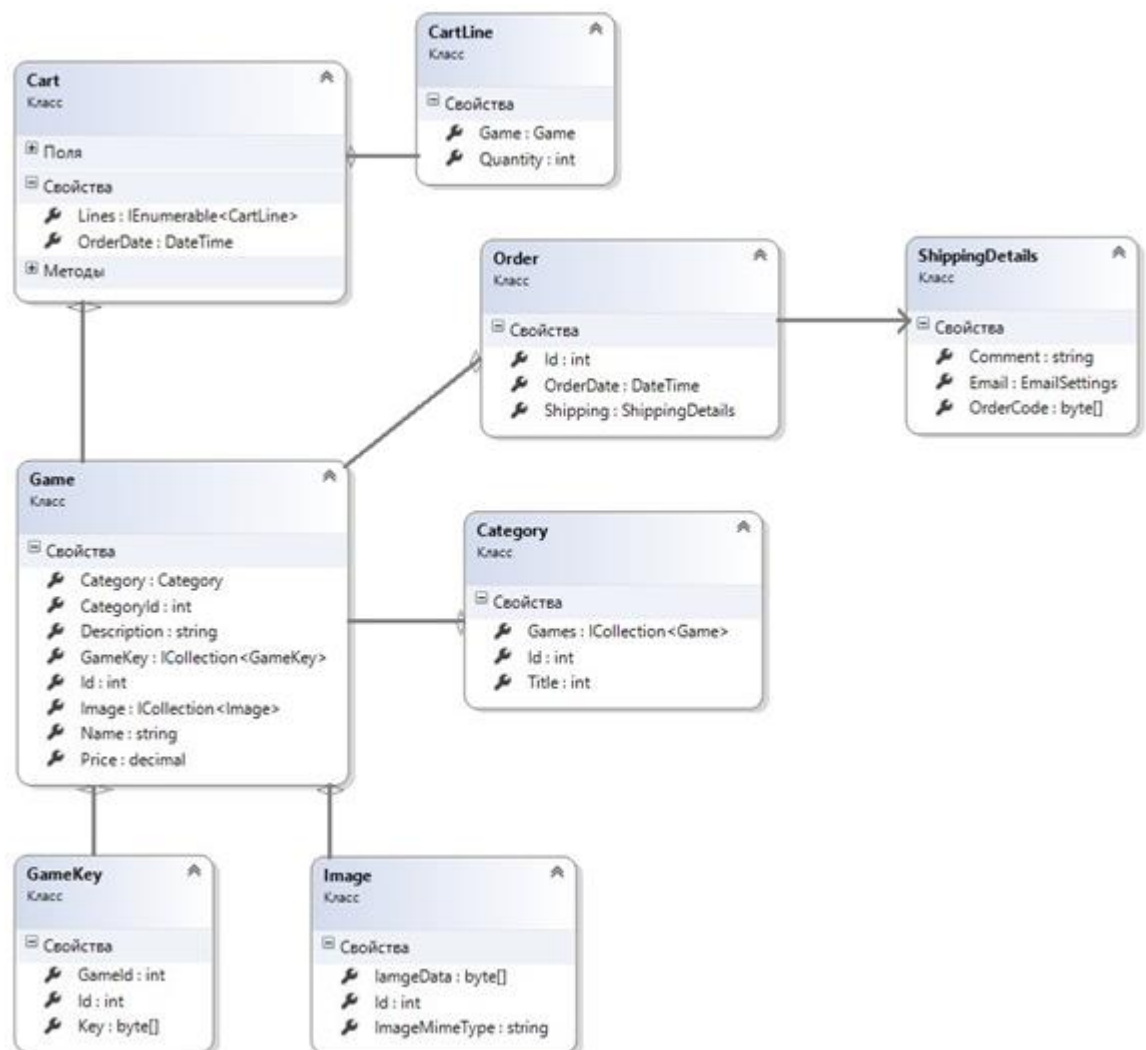


Рис. 2.1 Диаграмма классов

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

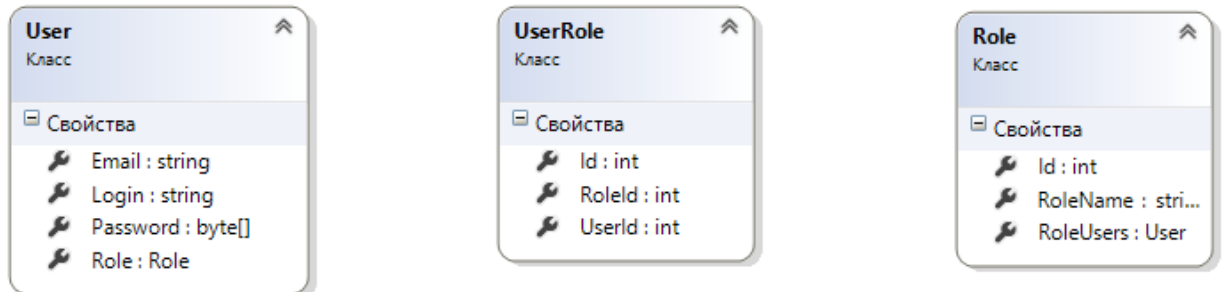


Рис. 2.2 Диаграмма классов пользователей и ролей

Подход *Code First* 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 *Code First* 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)

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 представить в виде таблицы:

Таблица 2.1 Методы действий контроллеров

| Контроллер | Метод действия | Описание действия |
|----------------------------|-----------------------|---|
| <i>GameController</i> | <i>List</i> | Список всех товаров |
| | <i>GetImage</i> | Страница отдельного товара |
| <i>NavigatorController</i> | <i>Menu</i> | Список всех категорий |
| | <i>Edit</i> | CRUD операции для категорий |
| | <i>Delete</i> | |
| | <i>Create</i> | |
| <i>AccountController</i> | <i>Register</i> | Регистрация пользователя |
| | <i>LogOn</i> | Вход пользователя |
| | <i>LogOff</i> | Выход пользователя |
| | <i>PasswordChange</i> | Смена пароля |
| <i>CartController</i> | <i>Checkout</i> | С помощью системы привязки моделей, определяется пуста ли корзина |
| | <i>AddToCart</i> | Добавление товара в корзину |
| | <i>RemoveFromCart</i> | Удаление товара из корзины |

| | | |
|---------------------------|-----------------|--|
| | <i>Summary</i> | Метод для визуализирования представления, который передаёт в качестве представления данных текущий объект, получаемый с использованием специального метода связывателя модели |
| | <i>Index</i> | Отображение содержимого корзины |
| <i>CheckoutController</i> | <i>Address</i> | Заполнение информации о доставке |
| | <i>Payment</i> | Оплата товара |
| | <i>Complete</i> | Завершение оформления заказа |
| <i>AdminController</i> | <i>Index</i> | Отображение товаров |
| | <i>Edit</i> | Функции изменения и сохранения работают через перегрузку метода Edit |
| | <i>Delete</i> | Удаление товара |

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.

Рассмотрим некоторые важные компоненты сквозной функциональности.

Внедрение 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 выноситься 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. *Dependency Injection*)

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 (IoC container)*. 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 *MVC Framework* распознаватель зависимостей применяется для создания экземпляров классов, необходимых для обслуживания запросов. Создавая специальный распознаватель, мы гарантируем, что *MVC Framework* использует *Ninject* всегда, когда должен создаваться тот или иной объект, включая экземпляры контроллеров.

Регистрация распознавателя зависимостей

Простого создания реализации интерфейса недостаточно - инфраструктуре *MVC Framework* необходимо также сообщить на необходимость его использования.

Пакеты *Ninject*, добавленные с помощью *NuGet*, в папке *App_Start* создан файл по имени *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 *MVC Framework*, как показано в примере ниже. Результатом данного оператора является создание шлюза между *Ninject* и поддержкой внедрения зависимостей в *MVC Framework*.

Код использование *Ninject* представлен в приложении.

Реализация корзины товаров

Механизм добавления товаров в корзину реализован, знакомым всем, кто когда-либо совершал покупки в Интернет-магазина.

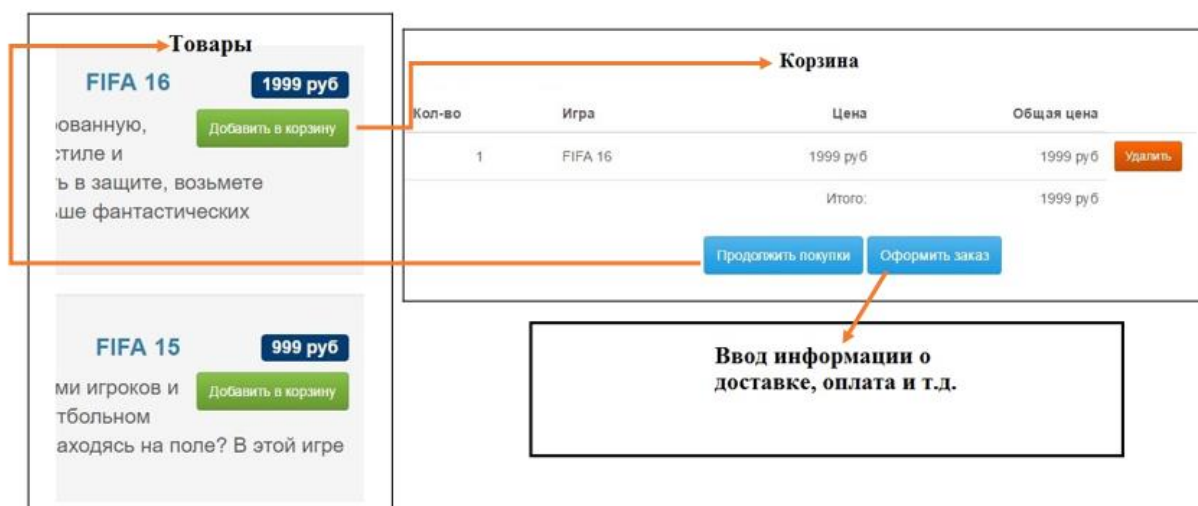


Рис. 2.3 Механизм работы корзины покупателя

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 *MVC Framework* использовала класс *CarttModelBinder*, необходимо в методе *Application_Start()*, файла *Global.asax*, добавить реализованный связыватель:

```
ModelBinders.Binders.Add(typeof(Cart), new CartModelBinder());
```

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.

- Любой класс 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 *CartItemViewModel.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.

Подключение 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 ранее 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:

Яндекс Деньги

Товар, услуга или магазин

Найти

Товары и услуги

Переводы

Банковские карты

Прием платежей

Скидки и бонусы

Настройки

HTTP-уведомления

Этот сервис позволит вам получать уведомления о входящих платежах или переводах на адрес вашего сервера по протоколу HTTPS (HTTP).

Укажите адрес, на который хотите получать уведомления. Поддерживаются протоколы https:// и http://

Показать секрет

Изменить секрет

Секретное слово позволит вам проверять подлинность уведомлений.

Протестировать

☐ Отправлять уведомления

Сохранить

Подробнее

[Документация](#)

Другие способы

[Email-информирование](#)

[Уведомления по смс](#)

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:


```

<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/Car/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: 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.

Обработка заказа

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

В приложении реализован компонент, которому можно обработку информации о заказе. Соблюдая принципы модели *MVC*, был определён интерфейс для этой функциональности, написана его реализацию 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 *.NET Framework*.

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 *SMTP* сервер.

Файлы отвечающие за реализацию работоспособности отправки заказа представлены в приложении.

Панель 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 *MVC 5 Controller* 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 *Visual Studio* 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 *Entity Framework*, *EFGameRepository* отвечает за удаление, добавление данных в контексте данных.

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

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

Так же 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 данные пользователей представлены в приложении.

Графический интерфейс приложения

При открытии приложения, пользователю представляется главная страница сайта (Рис.3.1.).



Рис.3.1. Внешний вид приложения

Здесь он может выбрать интересующий его жанр игр, прочитать их минимальное описание, ознакомиться со стоимостью каждой игры и перейти на следующую страницу списка игра (Рис.3.2).

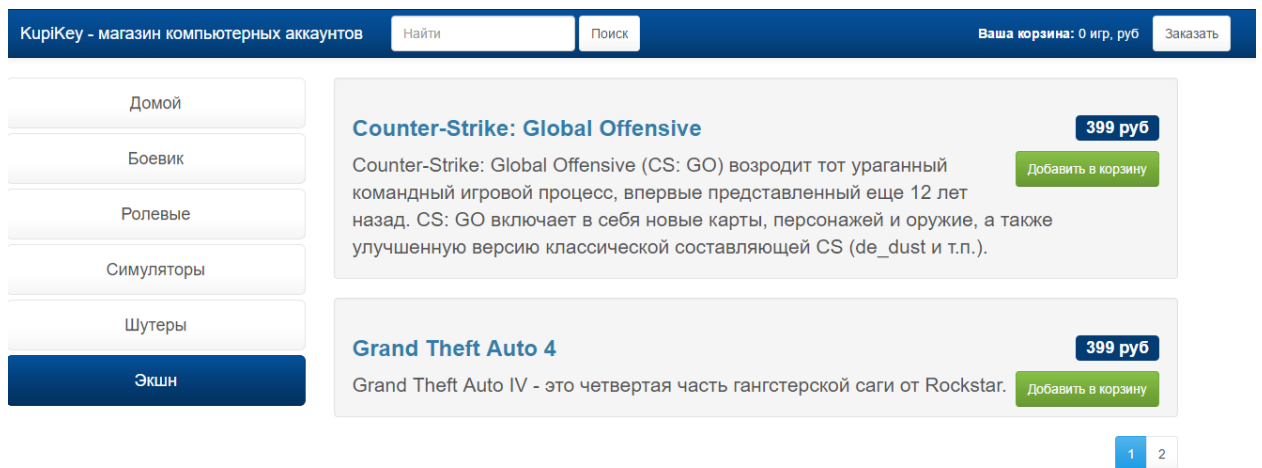


Рис.3.2. Отображение игр по категориям

При добавлении товара в корзину, пользователя перенаправляет на содержимое его корзины, где он имеет возможность редактировать содержимое корзины, ознакомиться с итоговой ценой всех товаров в корзине, а так же продолжить покупки или перейти к оформлению заказа (Рис.3.3.).

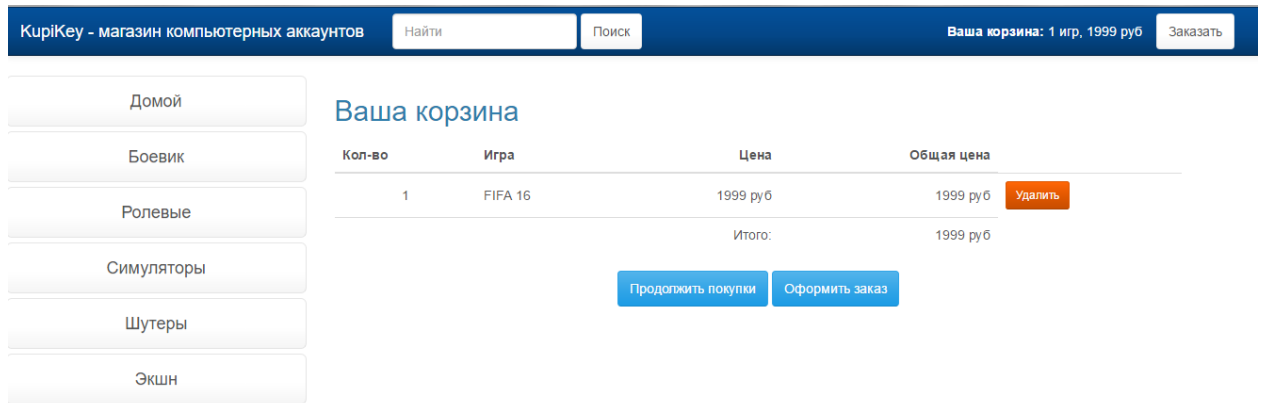


Рис.3.3. Корзина покупателя

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

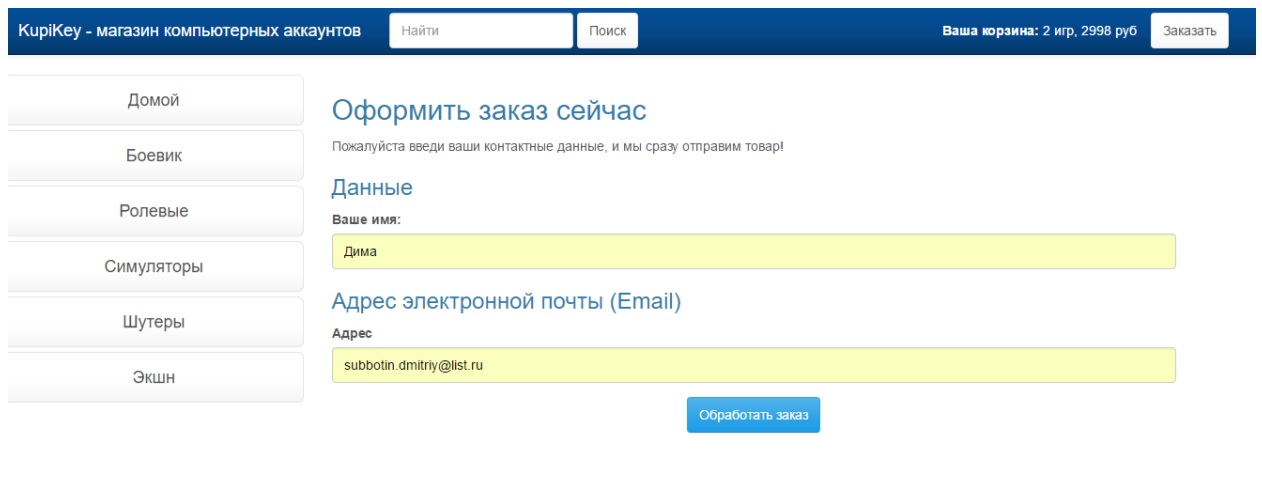


Рис.3.4. Форма оформления заказа

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 (Рис.3.5.).

Яндекс Деньги Товар, услуга или магазин Найти

История
Избранное
Товары и услуги
Переводы
Прием платежей

Оплата заказа 46

Данные платежа → **Подтверждение** → Результат

Название платежа **Оплата заказа 46**

№ счета получателя 410011174743222

Комиссия **0 руб. 01 коп.**

Сумма **1 руб.**

Способ оплаты **Яндекс.Деньги** Банковская карта

Напоминание о платеже **не нужно** ▼

Платежный пароль 🔒 •••••••••••••••• 🔊

Подтвердить

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 (Рис.3.6.).

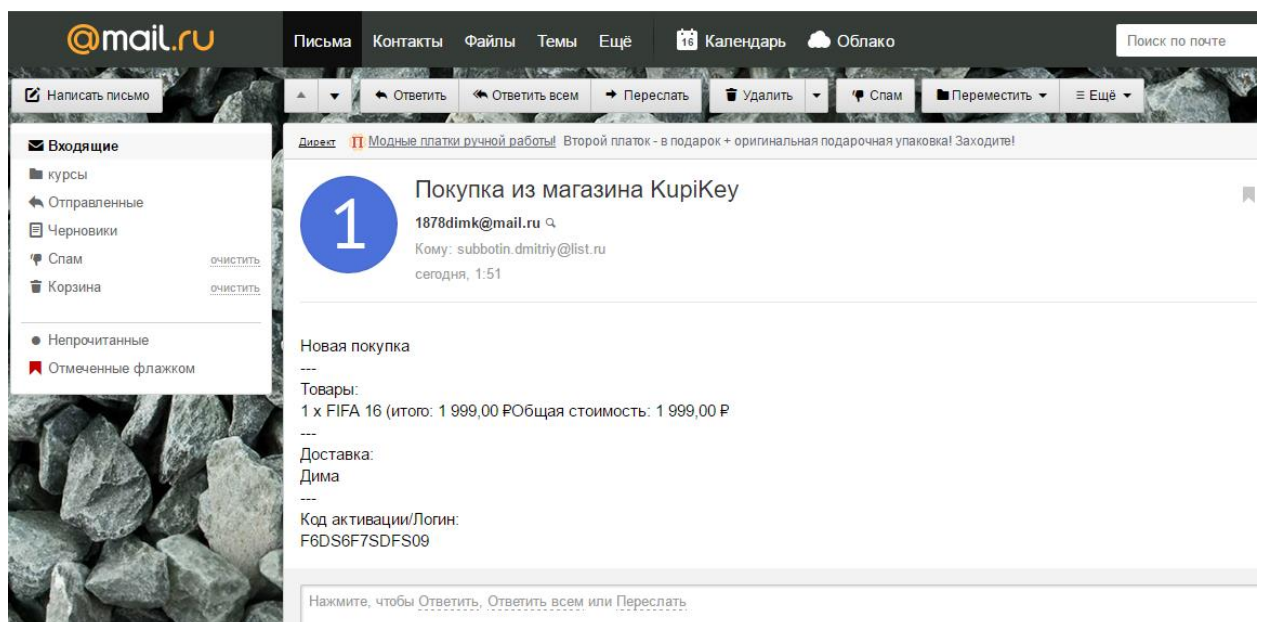


Рис.3.6. Отправка подтверждения заказа на сайт покупателя

После успешной оплаты покупателя переправляет на указанный разработчиком адрес, в данном случае на страницу успешной оплаты(Рис.3.7.).

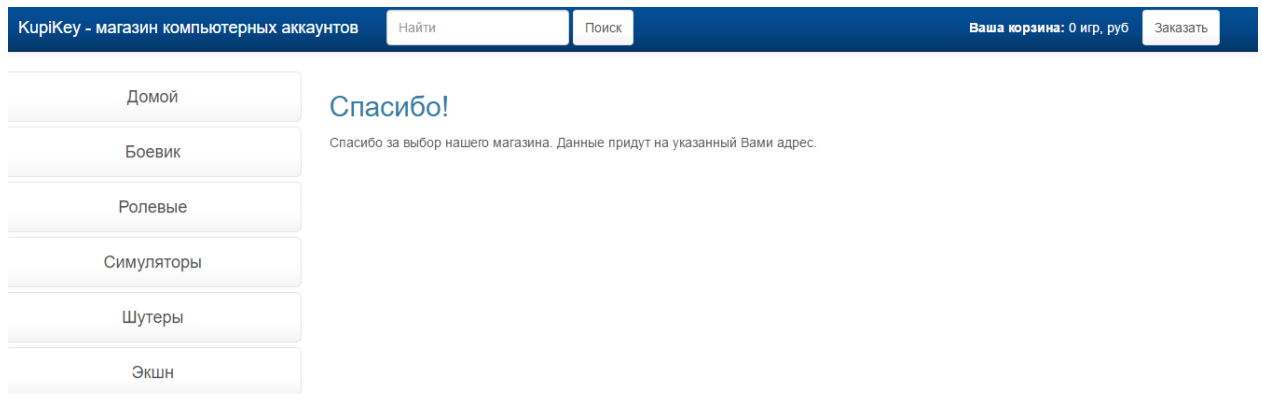


Рис.3.7. Подтверждение заказа

Для входа в панель администратора надо перейти на адрес Admin/Index, ввести данные администратора и пройти авторизацию (Рис.3.8.).

Вход в систему

Пожалуйста, войдите в систему, чтобы получить доступ к админ. панели:

Имя пользователя:

admin

Пароль:

Войти

Рис.3.8. Форма авторизации

При успешной авторизации администратора перейдёт на страницу со списком всех товаров магазина (Рис.3.9.)

Список игр

| ID | Название | Цена | Действия |
|----|----------------------------------|----------|----------|
| 1 | FIFA 16 | 1999 руб | Удалить |
| 2 | FIFA 15 | 999 руб | Удалить |
| 3 | Counter-Strike: Global Offensive | 399 руб | Удалить |
| 4 | Dark Souls III | 1999 руб | Удалить |
| 5 | Fallout 4 | 1999 руб | Удалить |
| 7 | Tom Clancy's The Division | 2099 руб | Удалить |
| 8 | The Elder Scrolls 5: Skyrim | 499 руб | Удалить |
| 9 | Mass Effect 2 | 399 руб | Удалить |
| 11 | Grand Theft Auto 4 | 399 руб | Удалить |
| 12 | Battlefield 3 | 599 руб | Удалить |
| 14 | qwerty | 499 руб | Удалить |

Добавить игру

Рис.3.9. Панель администратора

Напротив каждого элемента из списка формируется кнопка «удалить», при нажатии на неё происходит удаление товара из хранилища данных (Рис.3.10.).

Игра "qwerty" была удалена

Список игр

| ID | Название | Цена | Действия |
|----|--|----------|-------------------------|
| 1 | FIFA 16 | 1999 руб | Удалить |
| 2 | FIFA 15 | 999 руб | Удалить |
| 3 | Counter-Strike: Global Offensive | 399 руб | Удалить |
| 4 | Dark Souls III | 1999 руб | Удалить |
| 5 | Fallout 4 | 1999 руб | Удалить |
| 7 | Tom Clancy's The Division | 2099 руб | Удалить |
| 8 | The Elder Scrolls 5: Skyrim | 499 руб | Удалить |
| 9 | Mass Effect 2 | 399 руб | Удалить |
| 11 | Grand Theft Auto 4 | 399 руб | Удалить |
| 12 | Battlefield 3 | 599 руб | Удалить |

[Добавить игру](#)

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 (Рис. 3.11.).

Редактирование игры «»

Название

Fifa 17

Описание

тест

Категория

Симуляторы

Цена (руб)

2999

Картинка [Выберите файл...](#)

Нет картинки

Сохранить

Отменить изменения и вернуться к списку

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 (рис.3.12.).

Изменения в игре "Fifa 17" были сохранены

Список игр

| ID | Название | Цена | Действия |
|----|--|----------|-------------------------|
| 1 | FIFA 16 | 1999 руб | Удалить |
| 2 | FIFA 15 | 999 руб | Удалить |
| 3 | Counter-Strike: Global Offensive | 399 руб | Удалить |
| 4 | Dark Souls III | 1999 руб | Удалить |
| 5 | Fallout 4 | 1999 руб | Удалить |
| 7 | Tom Clancy's The Division | 2099 руб | Удалить |
| 8 | The Elder Scrolls 5: Skyrim | 499 руб | Удалить |
| 9 | Mass Effect 2 | 399 руб | Удалить |
| 11 | Grand Theft Auto 4 | 399 руб | Удалить |
| 12 | Battlefield 3 | 599 руб | Удалить |
| 16 | Fifa 17 | 2999 руб | Удалить |

[Добавить игру](#)

Рис.3.12. Список игр, после добавления новой игры

При нажатии на название игры, произойдёт переход на страницу редактирования указанного товара, где поля уже будут заполнены имеющейся информацией и показана картинка, если она добавлена (Рис.3.13.).

Редактирование игры «Fifa 17»

Название**Описание****Категория****Цена (руб)****Картинка** [Выберите файл...](#)

Рис.3.13. Форма редактирования товара

Так же используя набор инструментов Twitter Bootstrap и технологию чувствительного дизайна, была проработана мобильная версия приложения, которая показана на рисунках 3.14-3.19.



Рис.3.14. Мобильный вид приложения

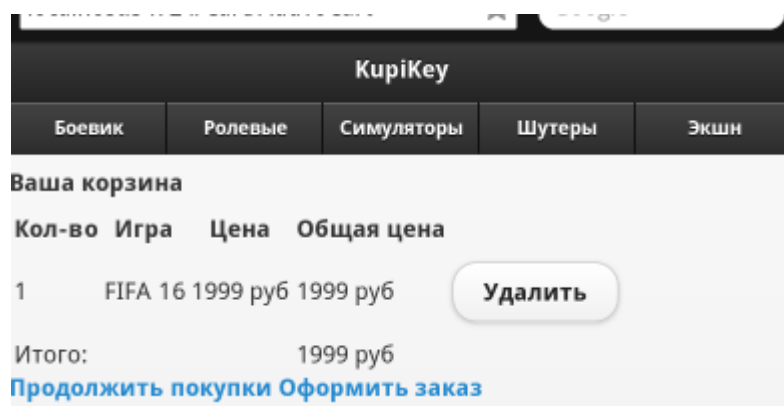


Рис.3.15. Мобильный вид корзины

| Боевик | Ролевые | Симуляторы | Шутеры | Экшн |
|---|---------|------------|--------|------|
| Оформить заказ сейчас Пожалуйста введи ваши контактные данные, и мы сразу отправим товар! | | | | |
| Данные | | | | |
| Ваше имя: | | | | |
| <input type="text"/> | | | | |
| Адрес электронной почты (Email) | | | | |
| Адрес | | | | |
| <input type="text"/> | | | | |
| <input type="button" value="Обработать заказ"/> | | | | |

Рис.3.16. Мобильный вид формы оформления заказа

<http://localhost:54724/Account/Login?ReturnUrl=%2fAdmin%2fIndex>

localhost:54724/Account/Login?ReturnUrl: ★

Вход в систему

Пожалуйста, войдите в систему, чтобы получить доступ к админ. панели:

Имя пользователя:

Пароль:

Рис.3.17. Мобильный вид формы авторизации

| Список игр | | | |
|------------|--|----------|-------------------------|
| ID | Название | Цена | Действия |
| 1 | FIFA 16 | 1999 руб | Удалить |
| 2 | FIFA 15 | 999 руб | Удалить |
| 3 | Counter-Strike: Global Offensive | 399 руб | Удалить |
| 4 | Dark Souls III | 1999 руб | Удалить |
| 5 | Fallout 4 | 1999 руб | Удалить |
| 7 | Tom Clancy's The Division | 2099 руб | Удалить |
| 8 | The Elder Scrolls 5: Skyrim | 499 руб | Удалить |
| 9 | Mass Effect 2 | 399 руб | Удалить |
| 11 | Grand Theft Auto 4 | 399 руб | Удалить |
| 12 | Battlefield 3 | 599 руб | Удалить |
| 16 | Fifa 17 | 2999 руб | Удалить |

Добавить игру

Рис.3.18. Мобильный вид списка товаров

FIFA 15

Описание

FIFA 15 обладает самой реалистичной физикой, моделями игроков и эмоциями, которые когда-либо были представлены в футбольном симуляторе. Хотите узнать, что чувствуют футболисты находясь на поле? В этой игре у вас будет такая возможность.

Категория

Симуляторы

Цена (руб)

999

Картинка [Выберите файл...](#)

PS4




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.

Серверный архитектурный каркас основан на многослойной архитектуре *Model-View-Controller*. 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, *Framework Bootstrap* и движок представление *Razor*.

Список литературы

1. Eric Evans «Domain-Driven Design: Tackling Complexity in the Heart of Software» / Eric Evans – ISBN-10: 0321125215, ISBN-13: 9780321125217, Edition: 1, 30 August 2003;
2. 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;
3. Jeffrey Richter «CLR via C#» / Jeffrey Richter - ISBN-13: 978-0735667457, ISBN-10: 0735667454, Edition 4, 25 November, 2012;
4. Jess Chadwick «Programming Razor» / Jess Chadwick - ISBN-13: 978-1449306762, ISBN-10: 1449306764, Edition 1, 25 September, 2011;
5. Уроки ASP.NET MVC / Хабрахабр : [сайт] – (URL: <http://habrahabr.ru/post/175999/>);
6. Bootstrap : [сайт] – (URL: <https://bootswatch.com/>);
7. Фреймворк Moq : [сайт] – (URL : <http://metanit.com/sharp/mvc5/18.5.php>);
8. Введение в Entity Framework 6 : [сайт] – (URL : metanit.com/sharp/entityframework/1.1.php);
9. Автоматическая миграция БД в Entity Framework : [сайт] – (URL : <http://habrahabr.ru/post/143292>);
10. IoC, DI, IoC-контейнер — Просто о простом / Хабрахабр : [сайт] – (URL : <https://habrahabr.ru/post/131993/>);
11. Уведомления о входящем переводе – Технологии Яндекса : [сайт] – (URL: <https://tech.yandex.ru/money/doc/dg/reference/notification-p2p-incoming-docpage/>);

Приложение 1. Внедрение зависимостей

```

public class NinjectDependencyResolver : IDependencyResolver
{
    private IKernel kernel;

    public NinjectDependencyResolver(IKernel kernelParam)
    {
        kernel = kernelParam;
        AddBindings();
    }

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

    public IEnumerable<object> GetServices(Type serviceType)
    {
        return kernel.GetAll(serviceType);
    }

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

        EmailSettings emailSettings = 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); // для подключения оплаты

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

public static class NinjectWebCommon
{
    private static readonly Bootstrapper bootstrapper = 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 IKernel CreateKernel()
    {
        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
{
    private IGameRepository repository;

    public NavController(IGameRepository repo)
    {
        repository = repo;
    }

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

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

        return PartialView("FlexMenu", categories);
    }
}

public class GameController : Controller
{
    // GET: Game
    private IGameRepository repository;

    public int pageSize = 2; // количество моделей на странице

    public GameController(IGameRepository repo)
    {
        repository = repo;
    }

    public ViewResult 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);
    }

    public FileContentResult GetImage(int gameId)
    {
        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
{
    private IGameRepository repository;
    private IOrderProcessor orderProcessor;
    private IPaymentProcessor paymentProcessor;

    public CartController(IGameRepository repo, IOrderProcessor processor,
IPaymentProcessor paymProcessor/*оплата*/)
    {
        repository = repo;
        orderProcessor = processor;
        paymentProcessor = paymProcessor; // оплата через систему

    }

    public ViewResult Checkout()
    {
        return View(new ShippingDetails());
    }

    [HttpPost]
    public ViewResult Checkout(Cart cart, ShippingDetails shippingDetails)
    {
        if (cart.Lines.Count() == 0)
        {
            ModelState.AddModelError("", "Извините, ваша корзина пуста!");
        }

        if (ModelState.IsValid)
        {
            //paymentProcessor.PaymentOrder(cart, shippingDetails); // оплата через
систему

            //View("Payment");
            orderProcessor.ProcessOrder(cart, shippingDetails); // доставка
            cart.Clear();
            return View("Completed");
        }
        else
        {
            return View(shippingDetails);
        }
    }

    public RedirectToRouteResult AddToCart(Cart cart, int gameId, string returnUrl)
    {
        Game game = repository.Games
            .FirstOrDefault(g => g.GameId == gameId);

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

    public RedirectToRouteResult RemoveFromCart(Cart cart, int gameId, string
returnUrl)
    {
        Game game = repository.Games
            .FirstOrDefault(g => g.GameId == gameId);

        if (game != null)

```

```

        {
            cart.RemoveLine(game);
        }
        return RedirectToAction("Index", new { returnUrl });
    }
    public PartialViewResult Summary(Cart cart)
    {
        return PartialView(cart);
    }

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

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

    public AdminController(IGameRepository repo)
    {
        repository = repo;
    }

    public ViewResult Index()
    {
        return View(repository.Games);
    }

    public ViewResult Edit(int gameId)
    {
        Game game = repository.Games
            .FirstOrDefault(g => g.GameId == gameId);
        return View(game);
    }

    // Перегруженная версия Edit() для сохранения изменений
    [HttpPost]
    public ActionResult 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("Изменения в игре \"{0}\" были  
сохранены", game.Name);
            return RedirectToAction("Index");
        }
        else
        {
            // Что-то не так со значениями данных
            return View(game);
        }
    }

    public ViewResult Create()
    {

```

```

        return View("Edit", new Game());
    }

    [HttpPost]
    public ActionResult Delete(int gameId)
    {
        Game deletedGame = repository.DeleteGame(gameId);
        if (deletedGame != null)
        {
            TempData["message"] = string.Format("Игра \"{0}\" была удалена",
                deletedGame.Name);
        }
        return RedirectToAction("Index");
    }
}

public class AccountController : Controller
{
    IAuthProvider authProvider;
    public AccountController(IAuthProvider auth)
    {
        authProvider = auth;
    }

    public ViewResult Login()
    {
        return View();
    }

    [HttpPost]
    public ActionResult 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("", "Неправильный логин или пароль");
                return View();
            }
        }
        else
        {
            return View();
        }
    }
}

```


Приложение 3. Репозитории

```

public interface IGameRepository
{
    IEnumerable<Game> Games { get; }
    void SaveGame(Game game);
    Game DeleteGame(int gameId);
}

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

    public IEnumerable<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(int gameId)
    {
        Game dbEntry = context.Games.Find(gameId);
        if (dbEntry != null)
        {
            context.Games.Remove(dbEntry);
            context.SaveChanges();
        }
        return dbEntry;
    }
}

```

Приложение 4. Реализация товара

```

public class Game
{
    [HiddenInput(DisplayValue = false)]
    public int GameId { get; set; }

    [Display(Name = "Название")]
    [Required(ErrorMessage = "Пожалуйста, введите название игры")]
    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, укажите категорию для игры")]
    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
    private IGameRepository repository;

    public int pageSize = 2; // количество моделей на странице

    public GameController(IGameRepository repo)
    {
        repository = repo;
    }

    public ActionResult 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);
    }

    public FileContentResult GetImage(int gameId)
    {
        Game game = repository.Games

```

```

        .FirstOrDefault(g => g.GameId == gameId);

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

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

    public PagingInfo PagingInfo { 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.PagedListLinks(Model.PagingInfo, x => Url.Action("List",
        new { page = x, category = Model.CurrentCategory }))
</div>

```

Приложение 5. Реализация корзины

```

public class Cart
{
    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()
    {
        return lineCollection.Sum(e => e.Game.Price * e.Quantity);
    }

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

    public IEnumerable<CartLine> Lines
    {
        get { return lineCollection; }
    }
}

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

public class CartController : Controller
{
    private IGameRepository repository;
    private IOrderProcessor orderProcessor;
    private IPaymentProcessor paymentProcessor;

    public CartController(IGameRepository repo, IOrderProcessor processor,
        IPaymentProcessor paymProcessor/*оплата*/)
    {
        repository = repo;
        orderProcessor = processor;
        paymentProcessor = paymProcessor; // оплата через систему
    }

    public ViewResult Checkout()

```

```

{
    return View(new ShippingDetails());
}

[HttpPost]
public ViewResult Checkout(Cart cart, ShippingDetails shippingDetails)
{
    if (cart.Lines.Count() == 0)
    {
        ModelState.AddModelError("", "Извините, ваша корзина пуста!");
    }
    if (ModelState.IsValid)
    {
        //paymentProcessor.PaymentOrder(cart, shippingDetails);
        // оплата через систему
        //View("Payment");
        orderProcessor.ProcessOrder(cart, shippingDetails); // доставка
        cart.Clear();
        return View("Completed");
    }
    else
    {
        return View(shippingDetails);
    }
}

public RedirectToRouteResult AddToCart(Cart cart, int gameId, string returnUrl)
{
    Game game = repository.Games
        .FirstOrDefault(g => g.GameId == gameId);

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

public RedirectToRouteResult RemoveFromCart(Cart cart, int gameId, string
returnUrl)
{
    Game game = repository.Games
        .FirstOrDefault(g => g.GameId == gameId);
    if (game != null)
    {
        cart.RemoveLine(game);
    }
    return RedirectToAction("Index", new { returnUrl });
}

public PartialViewResult Summary(Cart cart)
{
    return PartialView(cart);
}

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

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

```

```

public object BindModel(ControllerContext controllerContext,
    ModelBindingContext bindingContext)
{
    // Получить объект Cart из сеанса
    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
    return cart;
}
}

```

```

public class CartIndexViewModel
{
    public Cart Cart { get; set; }
    public string returnUrl { get; set; }
}

```

Checkout.cshtml

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

```

@{
    ViewBag.Title = "КупиKey: форма заказа";
}

```

```

<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>Ваше имя:</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="btn btn-primary" type="submit" value="Обработать заказ" />
    </div>
}

```

```

<iframe frameborder="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=1&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 = "КупиKey: ваша корзина";
}

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

<h2>Ваша корзина</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">Общая цена</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("# py6")</td>
                <td class="text-right">
                    @((line.Quantity * line.Game.Price).ToString("# py6"))
                </td>
                <td>
                    @using (Html.BeginForm("RemoveFromCart", "Cart"))
                    {
                        @Html.Hidden("GameId", line.Game.GameId)
                        @Html.HiddenFor(x => x.ReturnUrl)
                        <input class="btn btn-sm btn-warning" type="submit"
value="Удалить" />
                    }
                </td>
            </tr>
        }
    </tbody>
    <tfoot>
        <tr>
            <td colspan="3">Итого:</td>

```

```

        <td class="text-right">
            @Model.Cart.ComputeTotalValue().ToString("# py6")
        </td>
    </tr>
</tfoot>
</table>

<div class="text-center">
    <a class="btn btn-primary" href="@Model.ReturnUrl">Продолжить покупки</a>
    @Html.ActionLink("Оформить заказ", "Checkout", null, new { @class = "btn btn-primary"
    })
</div>

```

Payment.cshtml

```

@model GameStore.Domain.Entities.ShippingDetails
@{
    ViewBag.Title = "Страница оплаты";
    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" @*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="@Model.TotalPrice" @*Сумма*@ 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>

```

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 = "btn btn-default navbar-btn" })
</div>

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

<div class="navbar-text navbar-right">
    <b class="hidden-xs">Ваша корзина:</b>
    @Model.Lines.Sum(x => x.Quantity) итп,
    @Model.ComputeTotalValue().ToString("# py6")
</div>

```


Приложение 6. Реализация обработки заказа

```

public interface IOrderProcessor
{
    void ProcessOrder(Cart cart, ShippingDetails shippingDetails); // отправка на
    почту покупателя
}

public interface IPaymentProcessor
{
    void PaymentOrder(Cart cart, ShippingDetails shippingDetails); // оплата товара
}

public class EmailSettings
{
    public string MailToAddress = "subbotin.dmitriy@list.ru"; //кому
    public string MailFromAddress = "1878dimk@mail.ru"; // от кого
    public bool UseSsl = true;
    public string Username = "1878dimk@mail.ru";
    public string Password = "ilovefencing1994";
    public string ServerName = "smtp.mail.ru";
    public int ServerPort = 587;
    public bool WriteAsFile = true;
    public string FileLocation = @"d:\game_store_emails";
}

public class EmailOrderProcessor : IOrderProcessor
{
    private EmailSettings emailSettings;

    public EmailOrderProcessor(EmailSettings settings)
    {
        emailSettings = settings;
    }

    public void ProcessOrder(Cart cart, ShippingDetails shippingInfo)
    {
        using (var smtpClient = 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; // Можно переключить для сохранения
писем в файл

                smtpClient.PickupDirectoryLocation = emailSettings.FileLocation;
                smtpClient.EnableSsl = true;
            }

            StringBuilder body = new StringBuilder()
                .AppendLine("Новая покупка")
                .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("Общая стоимость: {0:c}", cart.ComputeTotalValue())

```

```

        .AppendLine("")
        .AppendLine("---")
        .AppendLine("Доставка:")
        .AppendLine(shippingInfo.Name)
        .AppendLine("---")
        .AppendLine("Код активации/Логин:")
        .AppendLine("F6DS6F7SDFS09");

        MailMessage mailMessage = new MailMessage(
            emailSettings.MailFromAddress, // От кого
            shippingInfo.Email,           // Кому
            "Покупка из магазина KupiKey", // Тема
            body.ToString());             // Тело
письма

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

        smtpClient.Send(mailMessage);
    }
}

public class ShippingDetails
{
    [Required(ErrorMessage = "Укажите как вас зовут")]
    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">
            
        </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="btn btn-success" value="Добавить в корзину" />
        </div>
    }
    <span class="lead">@Model.Description</span>
</div>

```