*Федеральное государственное бюджетное образовательное учреждение высшего образования*

*«Владимирский государственный университет*

*имени Александра Григорьевича и Николая Григорьевича Столетовых»*

*Кафедра информационных систем и программной инженерии*

***ПОЯСНИТЕЛЬНАЯ ЗАПИСКА***

***к курсовому проекту по дисциплине   
"Технологии программирования"***

***на тему***

*Проектирование и разработка программной системы   
информационной системы «Интернет-магазин»*

*Выполнил: студент гр. ПРИ-115*

*Обрубов М.О.*

*Принял: доц. кафедры ИСПИ*

*Вершинин В.В.*

*Владимир, 2017*

**Аннотация**

Суть курсовой работы заключается в разработке программной системы «Интернет-магазин». В курсовом проекте описывается проведенная работа по анализу предметной области, выделению необходимого функционала, проектированию, реализации и тестированию программной системы.

Курсовой проект представлен на 66 страницах, рисунков – 17, использованных источников – 3, приложений – 3, иллюстрационный материал на 3 листах формата А1.

The essence of the course work is the development of the software system "Internet shop". In the course project describes the work carried out on the analysis of the subject area, the allocation of the necessary functionality, design, implementation and testing of the software system.

The course project is presented on 66 pages, drawings - 17, used sources - 3, applications - 3, illustrative material on 3 sheets of A1 format.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc502304192)

[1 ПОСТАНОВКА ЗАДАЧИ 5](#_Toc502304193)

[2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 6](#_Toc502304194)

[2.1. Описание предметной области 6](#_Toc502304195)

[2.2. Словарь предметной области 7](#_Toc502304196)

[3 МОДЕЛИРОВАНИЕ СТРУКТУРЫ ОБЪЕКТОВ ПРЕДМЕТНОЙ ОБЛАСТИ И ИХ ВЗАИМОДЕЙСТВИЯ НА КОНЦЕПТУАЛЬНОМ УРОВНЕ 9](#_Toc502304197)

[3.1. Перечень ролей программной системы 9](#_Toc502304199)

[3.2. Диаграмма прецедентов 9](#_Toc502304200)

[3.3. Спецификация прецедентов 10](#_Toc502304201)

[4 ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПРОГРАММНОЙ СИСТЕМЫ 14](#_Toc502304202)

[4.1. Диаграмма классов 14](#_Toc502304204)

[4.2. Диаграмма состояний 18](#_Toc502304205)

[4.3. Диаграмма последовательностей 19](#_Toc502304206)

[5 РЕАЛИЗАЦИЯ ПРОТОТИПА ПРОГРАММНОЙ СИСТЕМЫ 21](#_Toc502304207)

[5.1. Структура проекта 21](#_Toc502304209)

[5.2. Подсистема хранения данных 21](#_Toc502304210)

[5.3. Ролевая модель 22](#_Toc502304211)

[5.4. Внешний сервис 23](#_Toc502304212)

[5.5. Логгирование 23](#_Toc502304213)

[5.6. Контроль версий 24](#_Toc502304214)

[5.7. Пользовательский интерфейс 24](#_Toc502304215)

[6 ТЕСТИРОВАНИЕ СИСТЕМЫ 31](#_Toc502304216)

[ЗАКЛЮЧЕНИЕ 32](#_Toc502304217)

[СПИСОК ИСПОЛЬЗВАННЫХ ИСТОЧНИКОВ 33](#_Toc502304218)

[ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММНОЙ СИСТЕМЫ 34](#_Toc502304219)

[ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ВЕБ-СЕРВИСА 64](#_Toc502304220)

[ПРИЛОЖЕНИЕ В. ИСХОДНЫЙ КОД ЮНИТ-ТЕСТОВ 66](#_Toc502304221)

# ВВЕДЕНИЕ

В 21 веке любой бизнес-процесс нуждается в информатизации. Без внедрения информационных технологий практически каждый процесс будет отставать от своих конкурентов.

В последние года значительные обороты набирают веб-ресурсы. Они позволяют вести бизнес по всему земному шару, так как доступ в интернет есть практически везде. Веб-сайты могут быть созданы либо для удовлетворения потребностей людей, путем предложения им товаров или услуг, либо для автоматизации деятельности каких-либо компаний.

Темой данного курсового проектирования является разработка программной системы для осуществления своей деятельности интернет-магазином. В ходе выполнения курсового проектирования будет спроектировано и реализовано веб-приложение, позволяющее удобно и с меньшими затратами управлять данным бизнесом.

Работа интернет-магазина заключается в взаимодействии с клиентами и предоставлении им товаров или услуг. Помимо этого, в работу интернет-магазина входит большой объем работы в сфере рекламы, маркетинга, удержания и повторного привлечения клиента, но, так как курсовое проектирование рассчитано только на разработку программной системы, то эти темы не могут быть затронуты.

# ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать программную систему для информационной системы «Интернет-магазин», для этого:

* выполнить исследование и анализ предметной области;
* разработать прототип программной системы;
* выполнить моделирование работы программной системы;
* разработать схему;
* реализовать программную систему с использованием выбранных средств и технологий.

Для выполнения работы необходимо использовать:

* методология проектирования и разработки – RUP;
* язык моделирования – UML;
* платформа разработки – ASP.NET.

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

* 1. Описание предметной области

Интернет-магазин – это предприятие по продаже различного рода товаров клиентам с помощью Интернета. Взаимодействие клиента и продавца осуществляется удаленно. Клиент сам выбирает себе необходимые товары и оплачивает их. Продавец же принимает заказ, формирует его и отправляет покупателю любым доступным способом. Далее можно рассмотреть подробнее алгоритм работы интернет магазина и взаимодействия его с клиентом.

Когда клиент попадает на страницу с перечнем товаров, например, «Каталог товаров», он видит все доступные для покупки товары. Они представлены в виде миниатюр (либо списком, либо плитками по несколько в строке). Клиент может выбрать любой понравившийся ему товар и попасть на страницу с его подробным описанием.

На странице с описанием товара может быть размещена любая информация о нём, например, фото товара, его полное описание, размер, цвет, наличие на складе магазина и многое другое. Если товар удовлетворяет клиента, то он выбирает необходимые ему характеристики и нажимает на кнопку «Добавить товар в корзину» и товар добавляется в корзину клиента. Далее таким же образом клиент добавляет другие товары.

Когда клиент «собрал» в корзину все необходимые товары, он приступает к оформлению заказа. Сделать он это может перейдя на страницу с перечнем товаров, добавленных в корзину и нажав кнопку «Оформить заказ». Следует учесть, что при оформлении заказа требуется, чтобы клиент был зарегистрирован и авторизован в системе. После нажатия на кнопку «Оформить заказ», клиент попадает на страницу с данными заказа, там он проверяет существующие данные и добавляет отсутствующие, необходимые для успешного оформления заказа.

После того, как клиент подтверждает введенные данные, заказ отправляется на проверку и подтверждение менеджером интернет-магазина. Менеджер проверяет данные заказа и любым доступным способом уточняет заказ у клиента. Как только менеджер подтверждает заказ, клиент либо оплачивает заказ онлайн, либо ожидает его и оплачивает сотруднику службы доставки. Сам заказ переходит в состояние доставки.

Помимо работы с клиентом, интернет-магазин нужно регулировать изнутри. Этим занимается руководитель или администратор интернет-магазина. Он может:

* управлять товарами, а именно добавлять, изменять или удалять их;
* управлять пользователями, а именно регистрировать новых пользователей, редактировать данные уже существующих или удалять их, а также изменять их роли в системе;
* управлять заказами, а именно изменять или удалять их, если вдруг возникнут ошибки в работе системы.

В рамках данного курсового проектирования будет рассмотрен только тот функциональный минимум, который необходим для работы интернет-магазина.

* 1. Словарь предметной области

Интернет-магазин – предприятие, осуществляющее продажу товаров покупателям в онлайн режиме.

Гость – пользователь, не зарегистрированный или не авторизованный в системе. Гость может авторизоваться или зарегистрироваться и получить дополнительные функции, доступные зарегистрированным пользователям. Гость имеет доступ к списку товаров и может просматривать подробную информацию о них, без возможности дальнейшего взаимодействия (добавления в корзину).

Зарегистрированный пользователь – пользователь, зарегистрированный в системе. Имеет следующие характеристики: логин и пароль для аутентификации и авторизации, роль в системе, адрес электронной почты, номер телефона, фамилию, имя, отчество, а также ряд характеристик, необходимых для функционирования системы. Предполагается, что зарегистрированный пользователь уже авторизован в системе, и функция авторизации ему недоступна.

Покупатель – зарегистрированный пользователь, который может совершать покупки в интернет-магазине. Покупатель имеет адрес доставки и ряд характеристик, необходимых для функционирования системы. Покупателю доступны функции гостя (за исключением функции регистрации и авторизации). Кроме этого покупатель может редактировать личную информацию, оформлять, оплачивать, а также просматривать все свои заказы.

Сотрудник – зарегистрированный пользователь, который обеспечивает функционирование интернет-магазина. Помимо характеристик зарегистрированного пользователя сотрудник имеет: порядковый номер сотрудника и информацию о заработной плате.

Менеджер – сотрудник интернет-магазина, взаимодействующий с покупателями (подтверждает и отправляет заказы). Менеджер имеет ряд функций: просматривать список доступных ему заказов, изменять статус заказа (например, переводить заказ из статуса «Созданный» в статус «Подтвержденный», т.е. подтверждать заказ).

Администратор – сотрудник интернет-магазина, осуществляющий управление основными сущностями интернет-магазина: пользователями, заказами, товарами и т.д.

Корзина – множество товаров, которые покупатель добавил для последующего оформления заказа.

Заказ – сущность, содержащая в себе список товаров, которые заказал (и, возможно, оплатил) покупатель. Помимо списка товаров заказ включает в себя статус (состояние заказа), информацию о покупателе, который сделал заказ и о менеджере, который сопровождает этот заказ.

Товар – вещь, которая покупатель может купить в интернет-магазине. Товар имеет следующие характеристики: наименование, цена, и др.

Оплата – процесс передачи денежных средств за приобретенный товар интернет-магазину.

Наличная оплата – оплата, которая производится при получении заказа.

Безналичная оплата – оплата, которая производится в онлайн режиме одним из доступных способов безналичного расчета при участии внешних платежных сервисов.

# МОДЕЛИРОВАНИЕ СТРУКТУРЫ ОБЪЕКТОВ ПРЕДМЕТНОЙ ОБЛАСТИ И ИХ ВЗАИМОДЕЙСТВИЯ НА КОНЦЕПТУАЛЬНОМ УРОВНЕ

1. 1. Перечень ролей программной системы

Роли проектируемой программной системы перечислены в следующем списке:

1. Гость.
2. Покупатель.
3. Менеджер.
4. Администратор.

Подробное описание каждой роли можно найти в словаре предметной области, а их функции на диаграмме прецедентов.

* 1. Диаграмма прецедентов

В ходе анализа предметной области, изучения структуры и функций проектируемой программной системы была разработана схема функций программной системы, т.н. диаграмма прецедентов. На ней изображены действующие лица (актеры), взаимодействующие с системой, а также функции системы (прецеденты), которые они могут выполнять. Диаграмма прецедентов представлена на рис. 1.

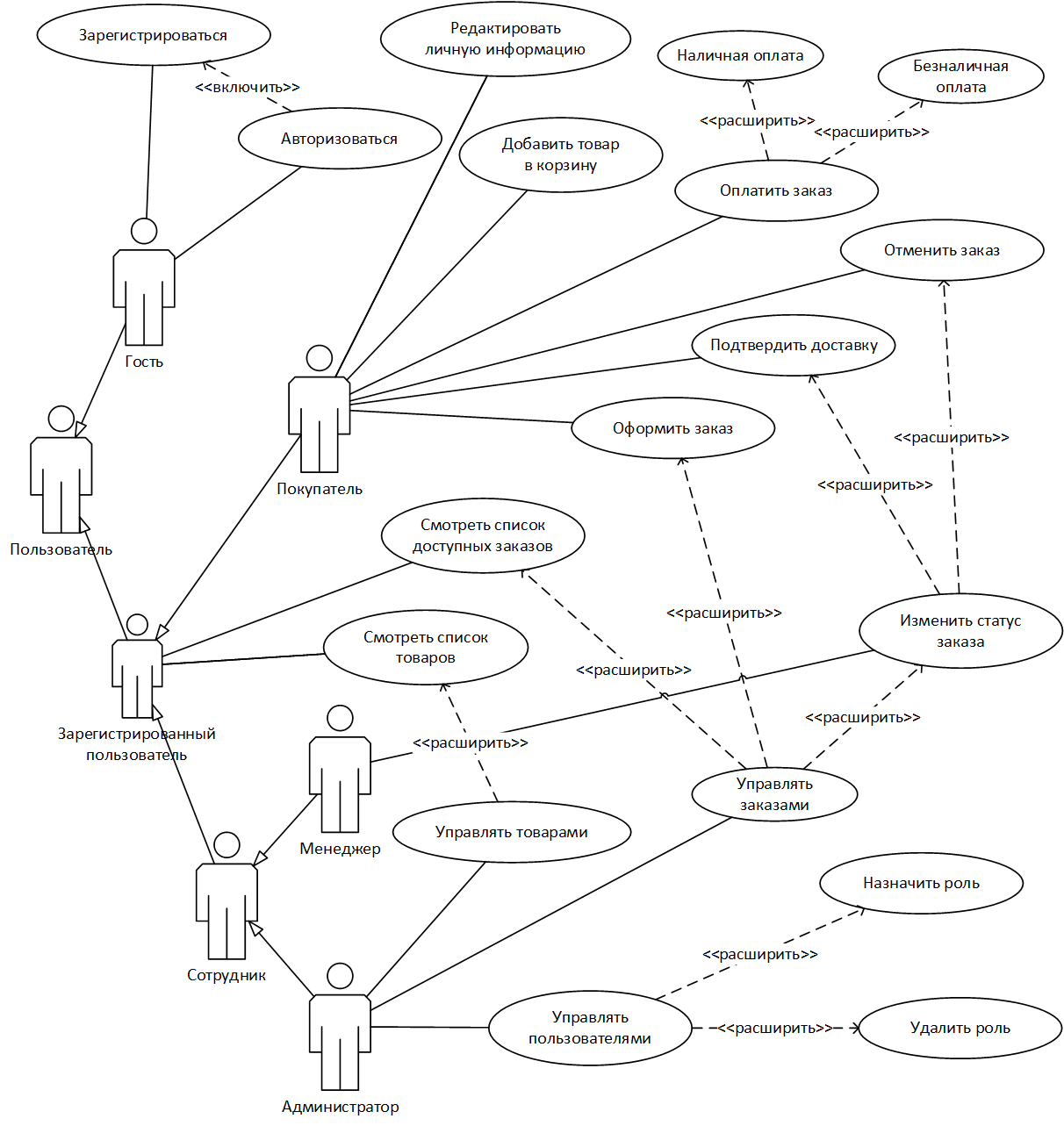


Рисунок 1. Диаграмма прецедентов.

* 1. Спецификация прецедентов

Ниже представлена словесная спецификация нескольких прецедентов программной системы: «Добавить товар в корзину», «Оформить заказ», «Подтвердить заказ» и «Назначить роль».

* + 1. ****Спецификация прецедента «Добавить товар в корзину»****

**Название:** Добавить товар в корзину.

**Краткое описание:** Действующее лицо добавляет выбранный товар в корзину.

**Действующее лицо:** Покупатель.

**Предусловие:** Действующее лицо находится на странице товара, который он хочет добавить в корзину.

**Основной поток:**

1. Действующее лицо нажимает на кнопку «Добавить в корзину».
2. Система сохраняет товар во внутреннем представлении объекта «Корзина».

**Постусловие:** Товар добавлен в корзину.

**Альтернативный поток:** Нет.

* + 1. ****Спецификация прецедента «Оформить заказ»****

**Название:** Оформить заказ.

**Краткое описание:** Действующее лицо оформляет заказ из товаров, находящихся в корзине.

**Действующее лицо:** Покупатель.

**Предусловие:** Действующее лицо находится на странице с корзиной товаров.

**Основной поток:**

1. Действующее лицо нажимает на кнопку «Оформить заказ».
2. Система перенаправляет действующее лицо на страницу оформления заказа.
3. Действующее лицо заполняет все необходимые для оформления заказа поля.
4. Действующее лицо нажимает кнопку «Готово».
5. Система сохраняет заказ в БД со статусом «Создан».

**Постусловие:** Заказ создан и сохранен в системе.

**Альтернативный поток 1:** *не заполнены обязательные поля*

1. На шаге 4 не заполнено хотя бы одно обязательное поле.
2. Система отображает сообщение действующему лицу о том, что не заполнены обязательные поля.
   * 1. Спецификация прецедента «Подтвердить заказ».

**Название:** Подтвердить заказ.

**Краткое описание:** Действующее лицо подтверждает созданный покупателем заказ, и он становится доступным для исполнения.

**Действующее лицо:** Менеджер.

**Предусловие:** Действующее лицо находится на странице с заказами, которые создали покупатели.

**Основной поток:**

1. Действующее лицо нажимает кнопку «Подтвердить заказ», находящуюся в строке необходимого заказа.
2. Система перенаправляет действующее лицо на страницу подтверждения товара.
3. Действующее лицо проверяет детали заказа.
4. Действующее лицо нажимает кнопку «Подтвердить».
5. Система переводит заказ в статус «Подтвержден» и обновляет его в БД. **Постусловие:** Заказ подтвержден.

**Альтернативный поток:** *заказ не верный*

1. На шаге 3 действующее лицо нажимает кнопку «Отклонить».
2. Система переводит заказ в статус «Отклонен».
   * 1. Спецификация прецедента «Назначить роль».

**Название:** Назначить роль.

**Краткое описание:** Действующее лицо назначает роль пользователю, зарегистрированному в системе.

**Действующее лицо:** Администратор.

**Предусловие:** Действующее лицо находится на странице с пользователями системы.

**Основной поток:**

1. Действующее лицо нажимает на кнопку «Изменить роль» в строке выбранного пользователя.
2. Система перенаправляет действующее лицо на страницу с ролями пользователя.
3. Действующее лицо нажимает на кнопку «Добавить роль».
4. Действующее лицо выбирает роль, которую необходимо назначить пользователю.
5. Действующее лицо нажимает кнопку «Добавить».
6. Система сохраняет изменения в БД.

**Постусловие:** Пользователю назначена роль.

**Альтернативный поток:** *нет.*

# ПРОЕКТИРОВАНИЕ СТРУКТУРЫ ПРОГРАММНОЙ СИСТЕМЫ

1. 1. Диаграмма классов

В результате проведения анализа предметной области на основе прецедентной модели были выделены основные сущности предметной области, их атрибуты и поведение, определены взаимосвязи между сущностями и разработана диаграмма классов предметной области (рис. 2).

Ниже приведено словесное описание сущностей системы, их атрибутов и методов.

1. RegisteredUser (Зарегистрированный пользователь)

*Атрибуты:*

* Login – логин пользователя для авторизации в системе;
* Password – пароль для авторизации в системе;
* Name – имя пользователя;
* Surname – фамилия пользователя;
* Birthdate – дата рождения;
* Phone – контактный телефон пользователя;
* Email – адрес электронной почты.

1. Buyer (Покупатель)

*Атрибуты:*

* Goods – корзина;
* Address – адрес доставки.

*Методы:*

* AddToCart – добавить товар в корзину;
* Checkout – оформить заказ;
* PayOnline – оплатить онлайн;
* CancelOrder – отменить заказ;
* ConfirmDelivering – подтвердить доставку заказа;
* EditPersonalInfo – изменить личную информацию.

1. Employee (Сотрудник)

*Атрибуты:*

* MonthSalary – зарплата за месяц.

1. Manager (Менеджер)

*Методы:*

* ConfirmOrder – подтвердить заказ;
* CancelOrder – отменить заказ.

1. Admin (Администратор)

*Методы:*

* AddUser – добавить пользователя;
* MofifyUser – изменить данные пользователя;
* DeleteUser – удалить пользователя;
* ChangeRole – изменить роль пользователя;
* AddGoods – добавить товар;
* ModifyGoods – изменить данные товара;
* DeleteGoods – удалить товар;
* DeleteOrder – удалить заказ.

1. Address (Адрес)

*Атрибуты:*

* Postcode – почтовый индекс;
* Country – страна;
* Region – область;
* City – город;
* Street – улица;
* House – дом;
* Flat – квартира.

1. Order (Заказ)

*Атрибуты:*

* Status – статус заказа;
* Buyer – покупатель;
* Manager – менеджер, ведущий заказ;
* Goods – корзина с товарами;
* DeliveryAddress – адрес доставки;
* PaymentType – тип оплаты;
* IsPayed – оплачен ли заказ;
* Cost – сумма заказа.

1. ShoppingCart (Корзина)

*Атрибуты:*

* Goods – список товаров в корзине.

1. Goods (Товар)

*Атрибуты:*

* Name – наименование;
* Price – цена.

1. OrderStatus (Статус заказа – перечисление)

* Created – создан;
* Confirmed – подтвержден;
* Payed – оплачен;
* Delivering – доставляется;
* Delivered – доставлен;
* Cancelled – отменен.

1. PaymentType (Тип оплаты – перечисление)

* Cash – наличная оплата;
* Cashless – безналичная оплата.

Так как на данном этапе проектировалась диаграмма классов концептуального уровня, то классы, необходимые для функционирования системы, например, классы доступа к данным, на диаграмме классов представлены не будут.

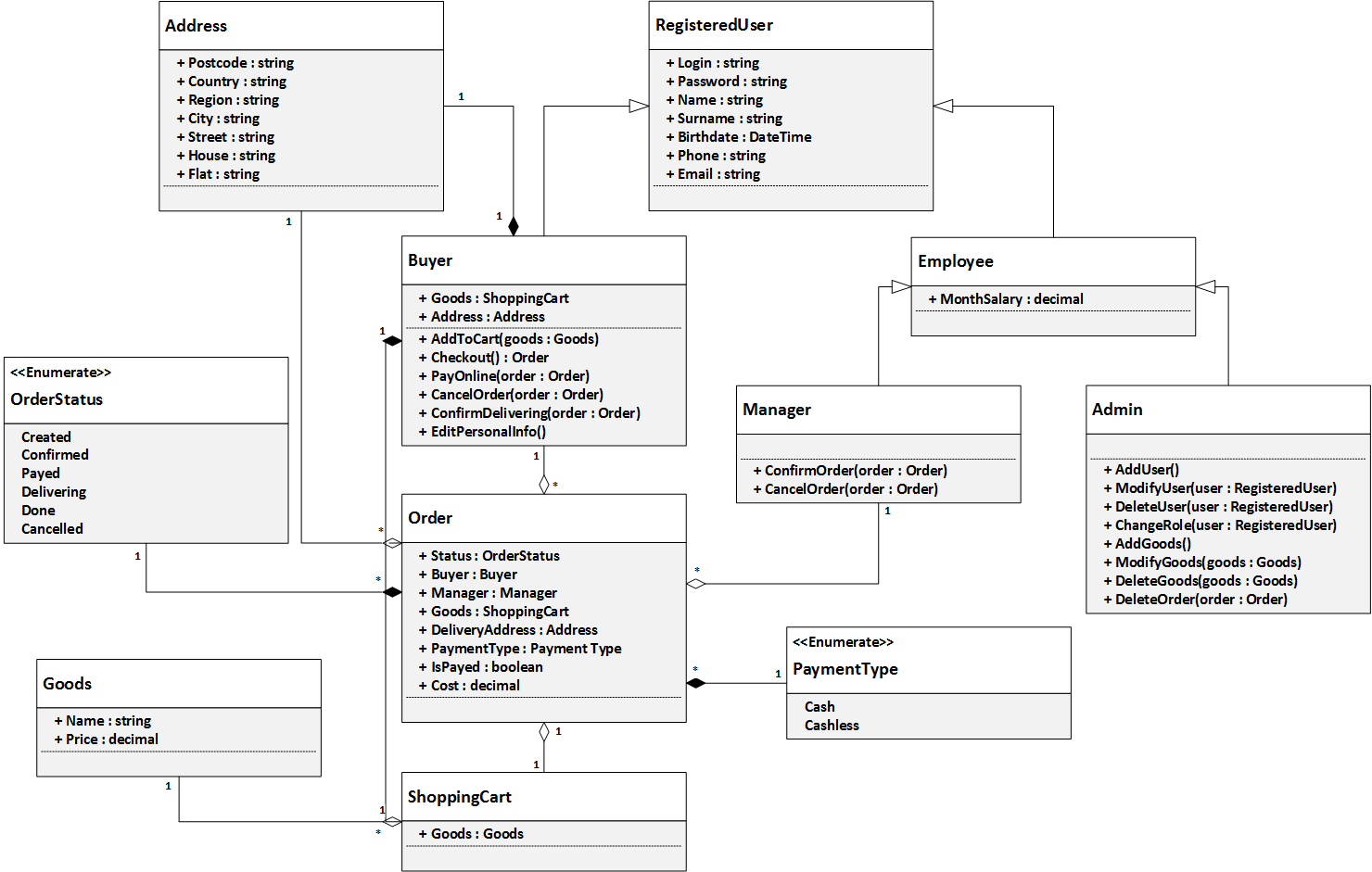


Рисунок 2. Диаграмма классов.

* 1. Диаграмма состояний

Для составления диаграммы состояний был выбран объект «Заказ», так как он является основным объектом бизнес-процессов интернет-магазина и может находиться во многих состояниях.

Ниже представлено словесное описание изменения состояния объекта «Заказ».

1. Покупатель создает заказ, указав в нём все необходимые параметры. Объект «Заказ» создан и сохранен во внутреннем представлении в системе со статусом «Создан».
2. Менеджер интернет-магазина проверяет параметры заказа на корректность (возможность доставки по указанному адресу, наличие товаров на складе и т.д.) и переводит заказ в состояние «Подтвержден», если заказ корректен.
3. В заказе имеются ошибки. Переход к п. 7.
4. Покупатель оплачивает заказ, и система переводит заказ в состояние «Оплачен».
5. Менеджер видит, что заказ оплачен, отправляет покупателю заказ и переводит заказ в статус «Доставляется».
6. Покупатель получает заказ и переводит заказ в статус «Завершен».
7. Заказ переходит в статус «Отменен».

Разработанная диаграмма состояний представлена на рис. 3.

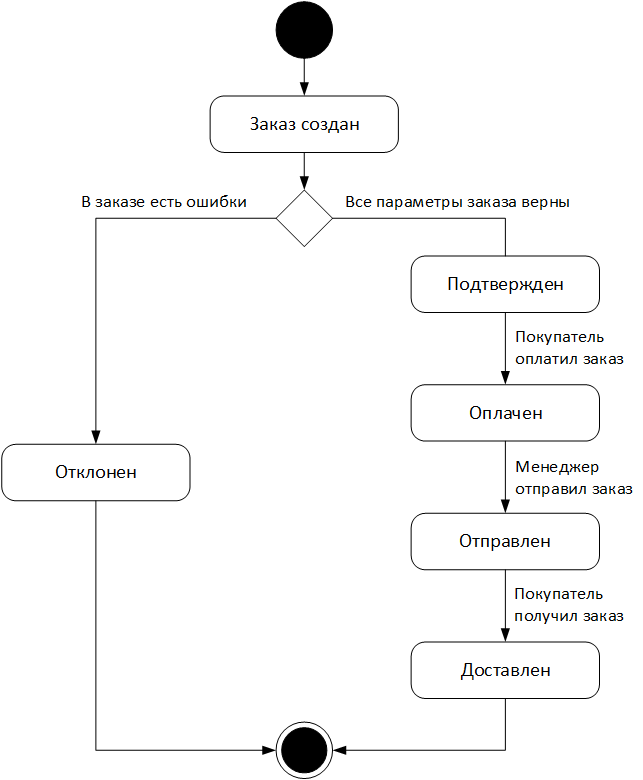


Рисунок 3. Диаграмма состояний.

* 1. Диаграмма последовательностей

Диаграммой последовательностей был описан вариант использования «Оплатить заказ».

На диаграмме: Покупатель – действующее лицо, OrderController – класс, реализующий работу с заказами, OrderDAO – класс, реализующий взаимодействие с базой данных, в частности с сущностью «Заказ», PayController – класс, реализующий работу с внешней платежной системой, Платежная система – внешняя платежная система, с помощью которой осуществляется оплата заказов в интернет-магазине.

Диаграмма последовательностей представлена на рис. 4.

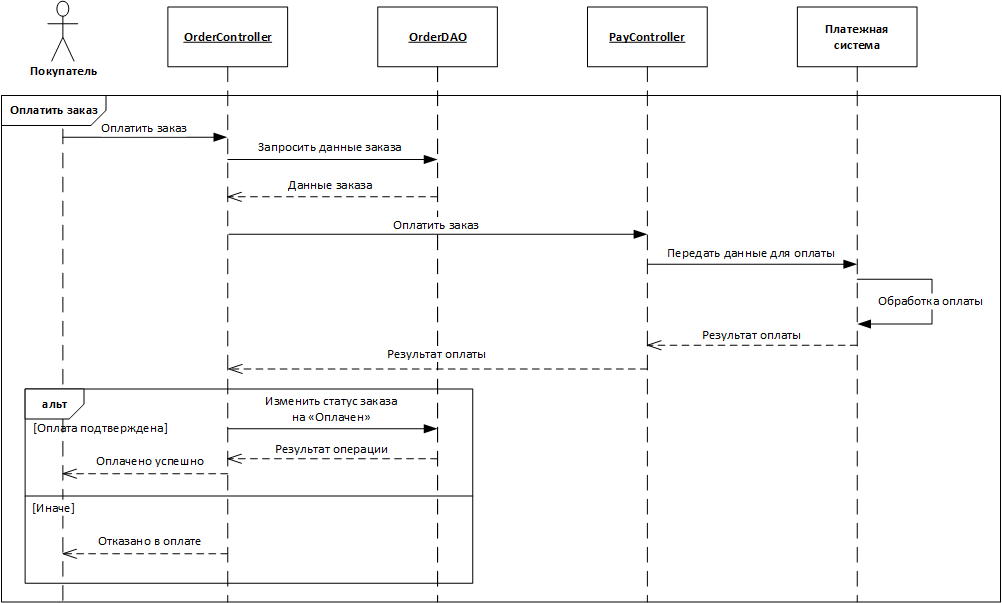


Рисунок 4. Диаграмма последовательностей.

# РЕАЛИЗАЦИЯ ПРОТОТИПА ПРОГРАММНОЙ СИСТЕМЫ

1. 1. Структура проекта

Для реализации программной системы была выбран шаблон проектирования MVC, так как он является наиболее удобным для реализации объектно-ориентированных программных систем.

Шаблон проектирования MVC предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

Модель – предоставляет данные и реагирует на команды контроллера, изменяя свое состояние.

Представление – отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Контроллер – интерпретирует действия пользователя, оповещая модель о необходимости изменений. [1]

Для взаимодействия с подсистемой хранения данных реализован слой доступа к данным – классы DAO (Data Access Object) для основных сущностей. Работа с данными осуществляется с помощью фреймворка ADO.NET Entity Framework, а запросы к БД строятся с помощью технологии LINQ to Entities.

Исходный код проекта веб-приложения представлен в приложении А

* 1. Подсистема хранения данных

При проектировании модели данных было принято решение использовать технологию ADO.NET Entity Framework 6, а в частности подход Database First. Суть этой технологии заключается в том, что сначала проектируется и реализуется структура базы данных, а потом по ней автоматически генерируются классы сущностей, т.н. value-objects.

Для хранения аутентификационных данных пользователей использовалась стандартная база данных и библиотека классов – ASP.NET Identity. К уже существующим сущностям были добавлены сущность предметной области. Диаграмма сущностей базы данных представлена на рис. 5.

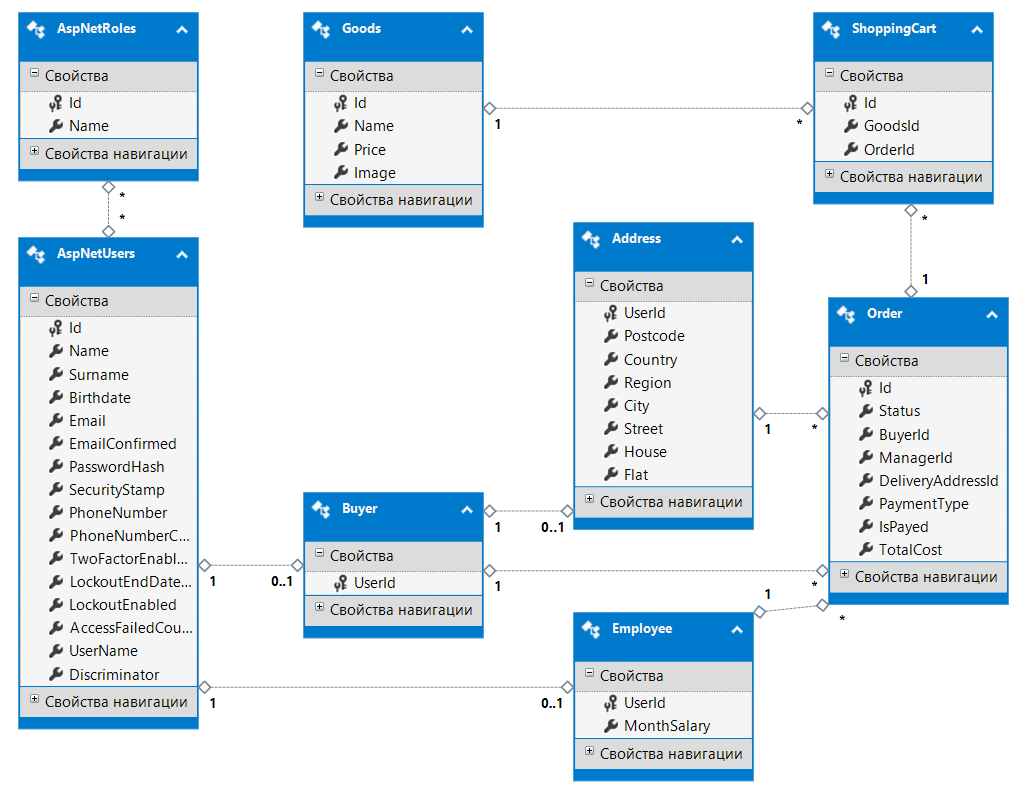


Рисунок 5. Диаграмма сущностей БД.

* 1. Ролевая модель

Контроль за доступом к ресурсам приложения на основании роли авторизованного пользователя был выполнен на нескольких уровнях:

* на уровне карты сайта (доступ к основным разделам сайта ограничен в файле конфигурации карты сайта Mvc.sitemap);
* на уровне контроллеров (с помощью аннотаций [Authorized]);
* на уровне представлений (на основании возвращаемого значения метода User.IsInRole() отображаются те или иные элементы пользовательского интерфейса).
  1. Внешний сервис

Так как интернет-магазину необходим инструмент онлайн-оплаты заказов, то веб-приложению необходима внешняя платежная система. Ввиду того, что в рамках курсового проектирования разрабатывается прототип программной системы, то для имитации работы внешней платежной системы было решено реализовать собственный веб-сервис, работающий по протоколу SOAP (Simple Object Access Protocol).

В подсистеме хранения данных веб-сервиса хранятся данные о счетах клиентов (данные банковских карт и сумма денег на счету). Запросы к базе данных реализованы в классе DAO с помощью ADO.NET Entity Framework и LINQ to Entities.

Скрипт создания таблицы в подсистеме хранения данных веб-сервиса:

CREATE TABLE [dbo].[ClientAccount] (

[CardNumber] NVARCHAR (16) NOT NULL,

[ExpYear] INT NOT NULL,

[ExpMonth] INT NOT NULL,

[CardHolder] NVARCHAR (50) NOT NULL,

[CVC] NVARCHAR (3) NOT NULL,

[Balance] DECIMAL (7, 2) NOT NULL,

PRIMARY KEY CLUSTERED ([CardNumber] ASC)

);

Веб-сервис предоставляет единственный метод, который имитирует онлайн-оплату по данным банковской карты и сумме, которую необходимо списать. В данном методе проверяются данные банковской карты и в случае успешной проверки с баланса пользователя списывается заданная сумма.

Исходный код веб-сервиса полностью представлен в приложении Б.

* 1. Логгирование

В процессе работы любого приложения возникают ошибки. Для того, чтобы своевременно находить причины ошибок и устранять их, нужно иметь информацию о том, как ведет себя приложение в процессе своей работы. Для этого используется логгирование.

Логгирование – это процесс записи информации о происходящих с каким-то объектом (или в рамках какого-то процесса) событиях в журнал (лог-файл). [2]

В ходе разработки веб-приложения в рамках курсового проектирования применялась сторонняя библиотека NLog.

* 1. Контроль версий

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. СКВ даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. [3]

В ходе разработки программной системы использовалась система контроля версий Git и в частности сервис GitHub.

В ходе работы в общей сложности было сделано 60 коммитов.

* 1. Пользовательский интерфейс

Скриншоты пользовательского интерфейса представлены на рис. 6 – 16.

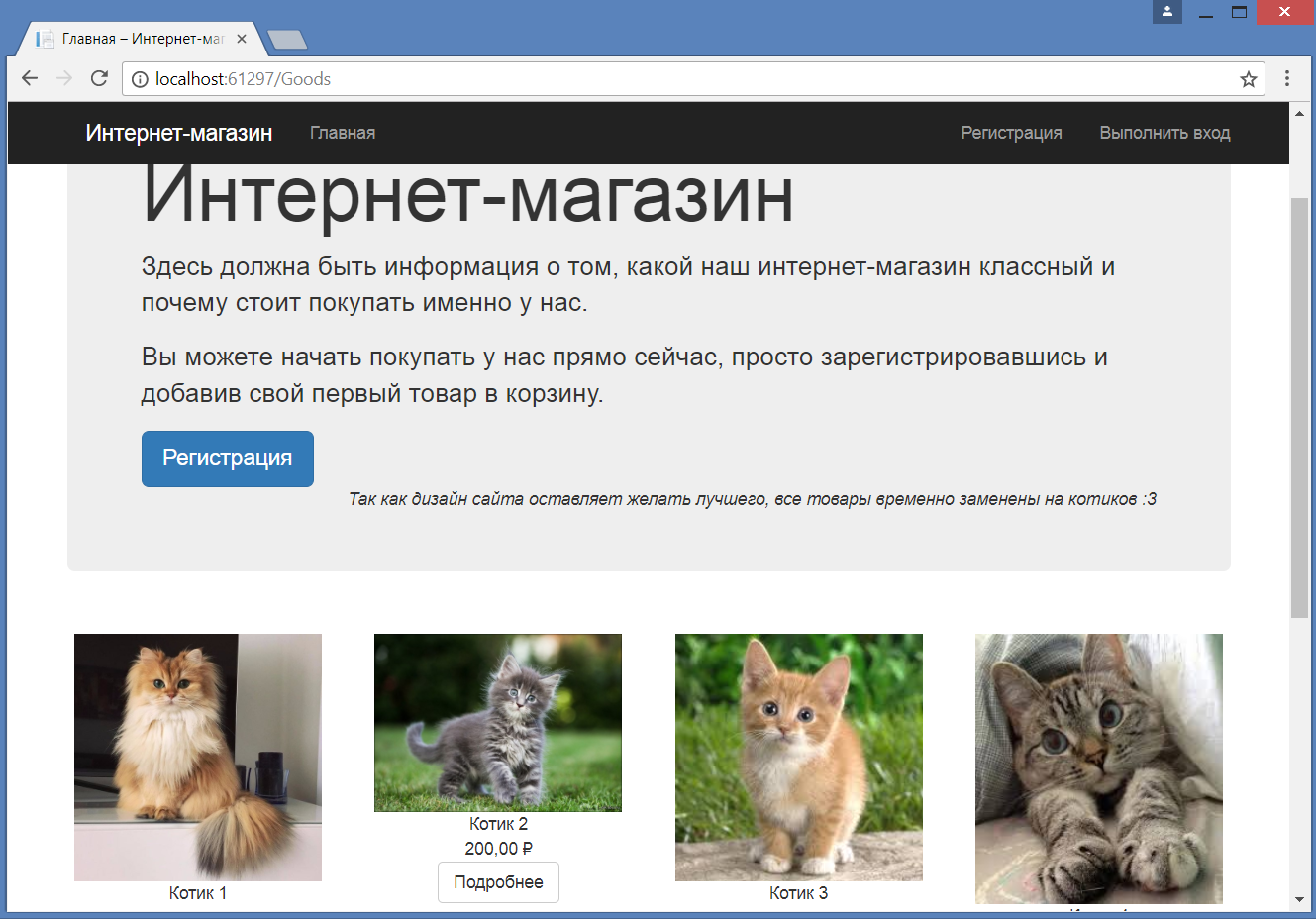


Рисунок 6. Главная страница.

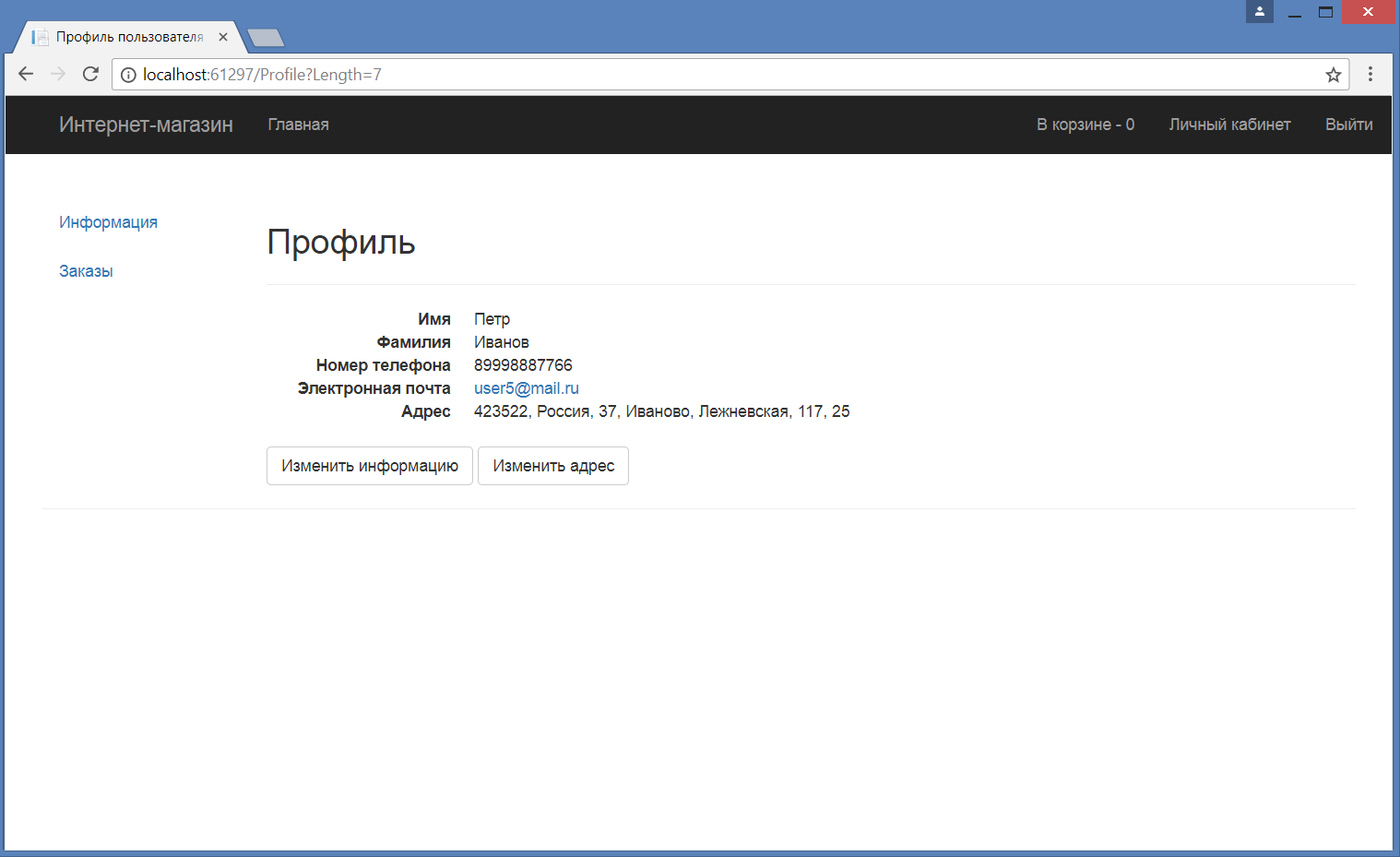


Рисунок 7. Страница профиля пользователя.

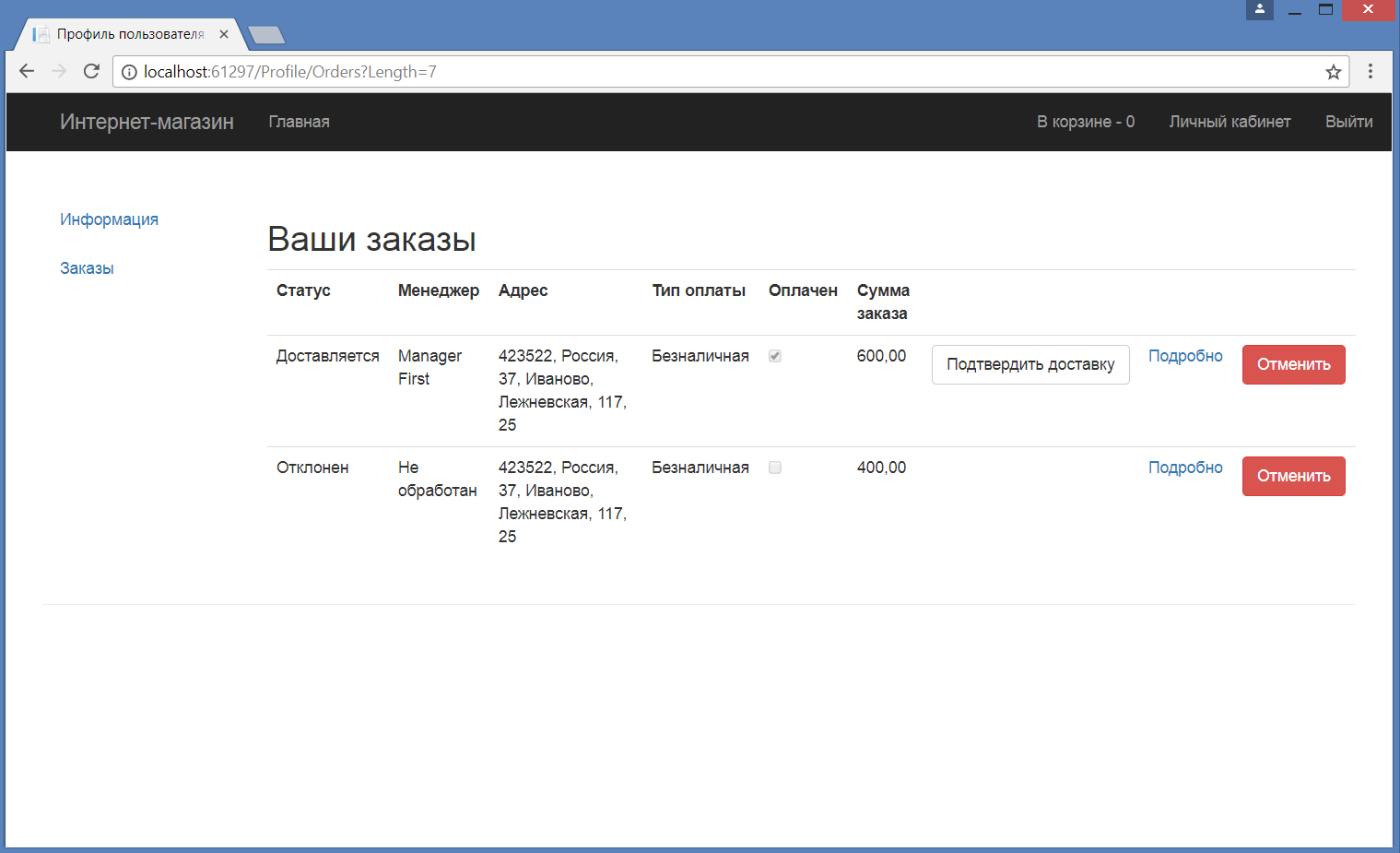


Рисунок 8. Страница с заказами пользователя.

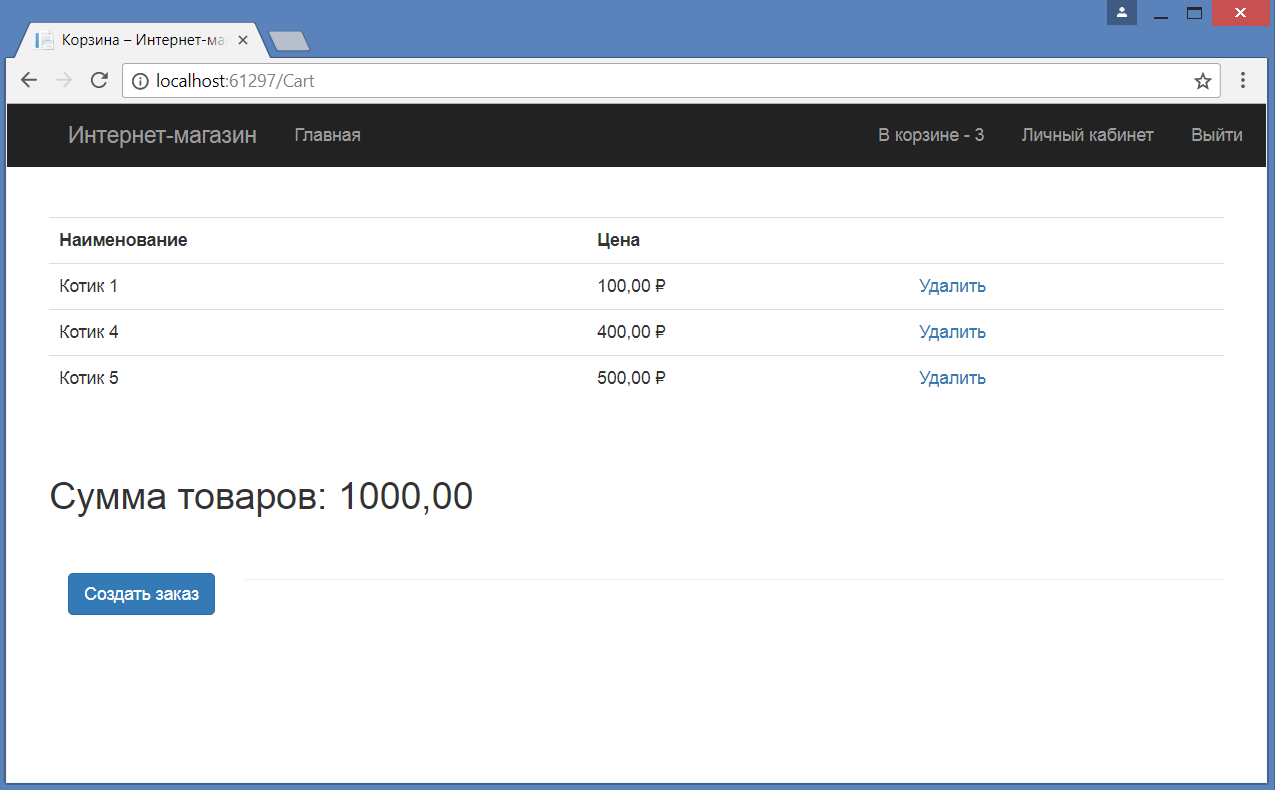


Рисунок 9. Страница корзины.

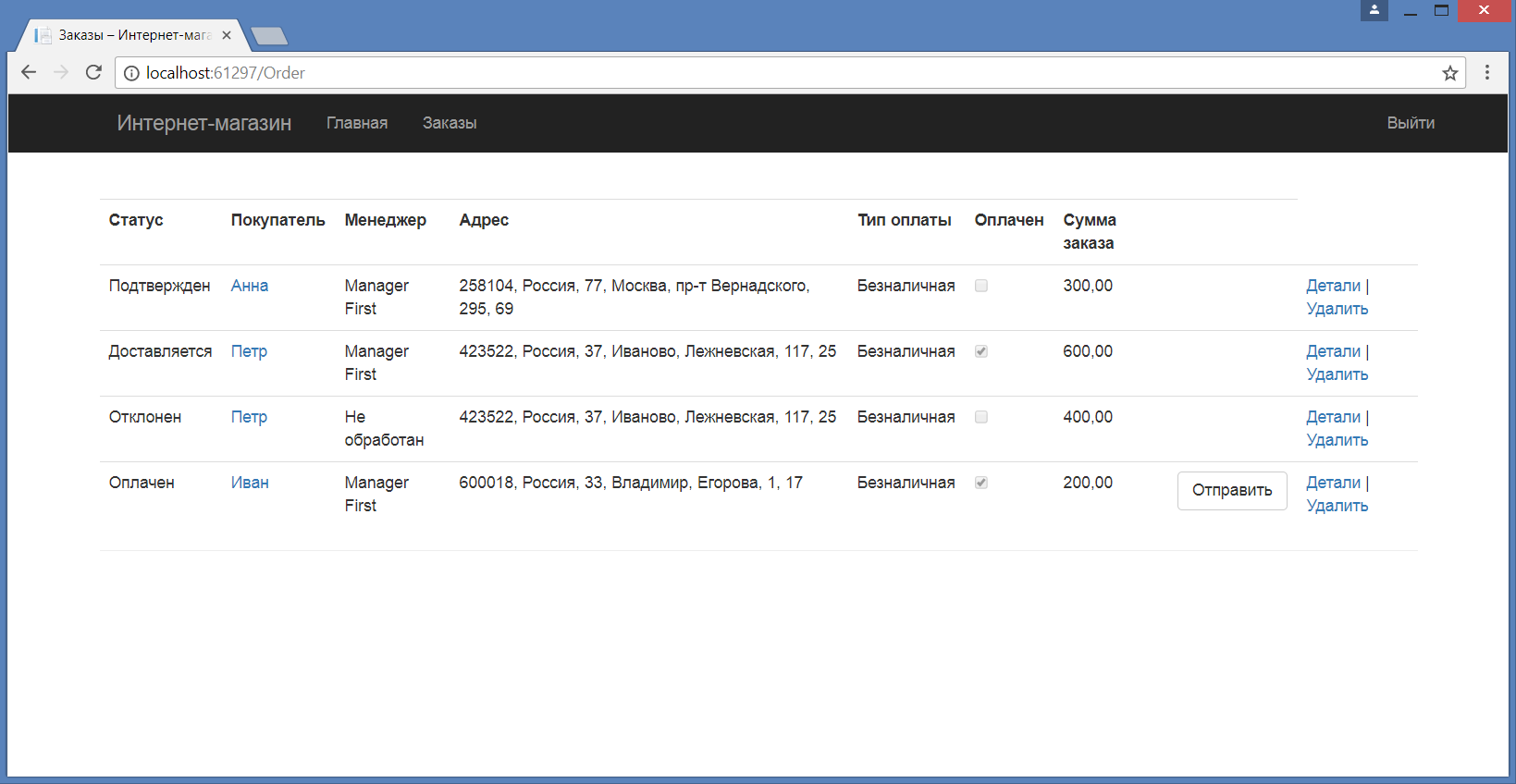


Рисунок 10. Страница с заказами для менеджера.

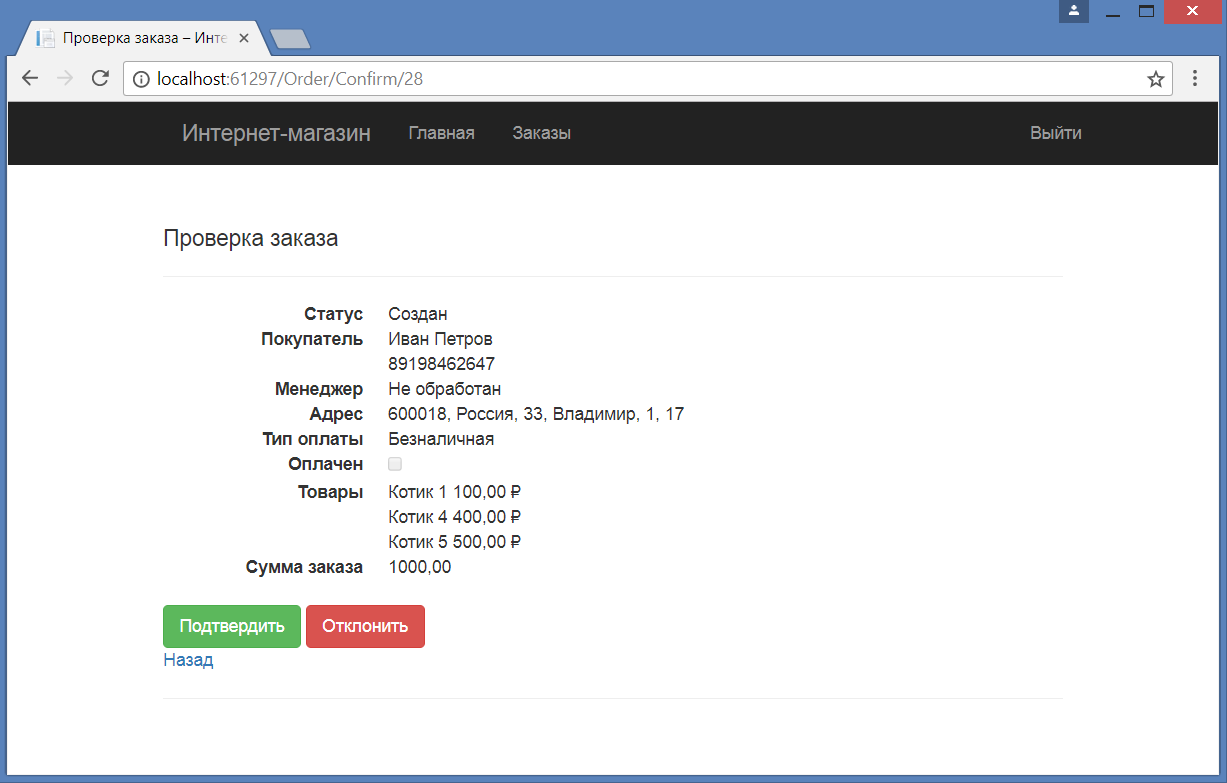


Рисунок 11. Страница подтверждения заказа.

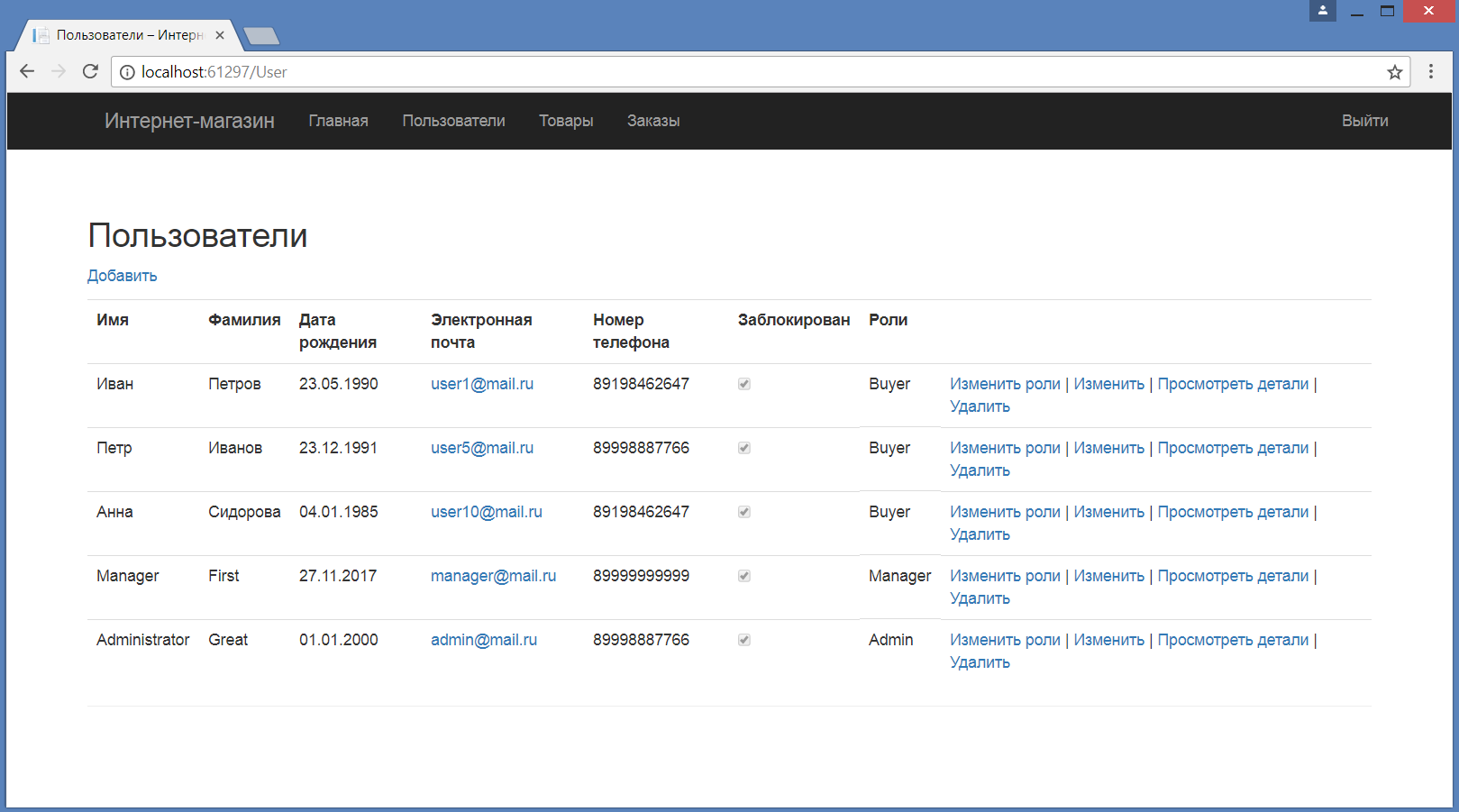


Рисунок 12. Страница с пользователями для администратора.

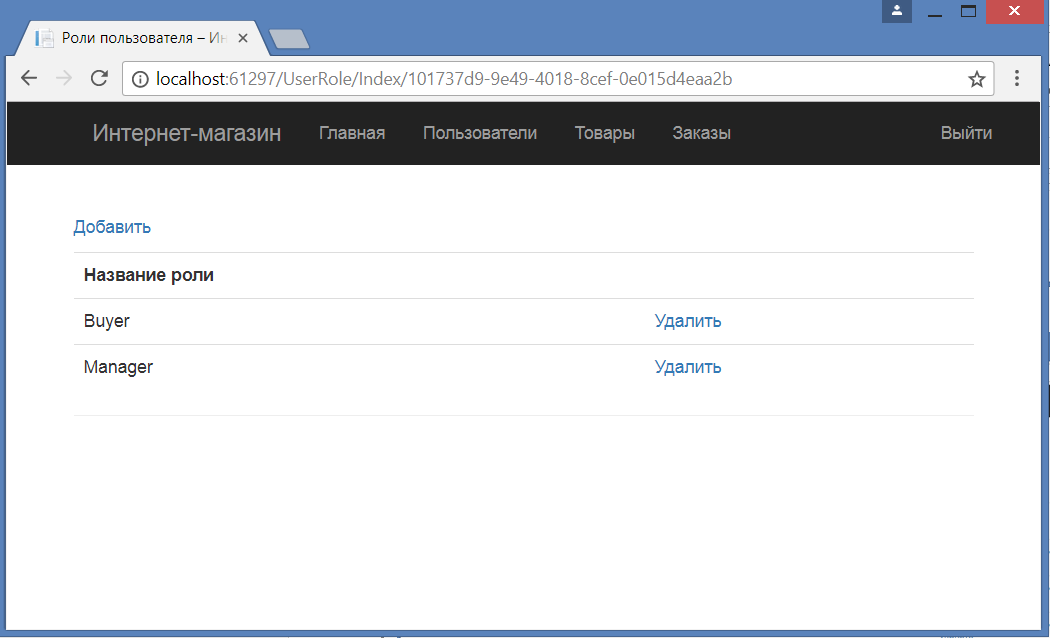


Рисунок 13. Страница изменения ролей пользователя.

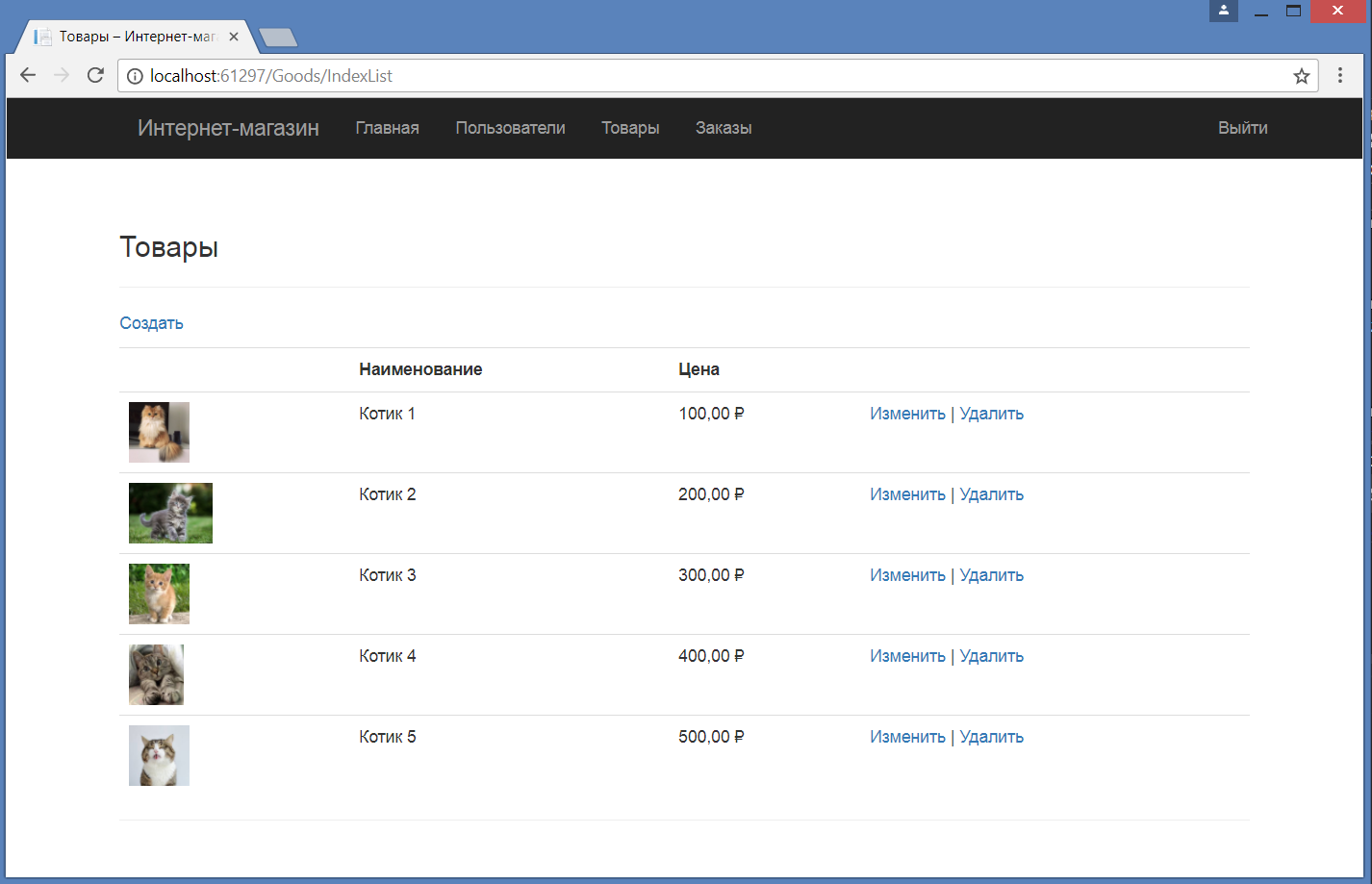


Рисунок 14. Страница с товарами для администратора.

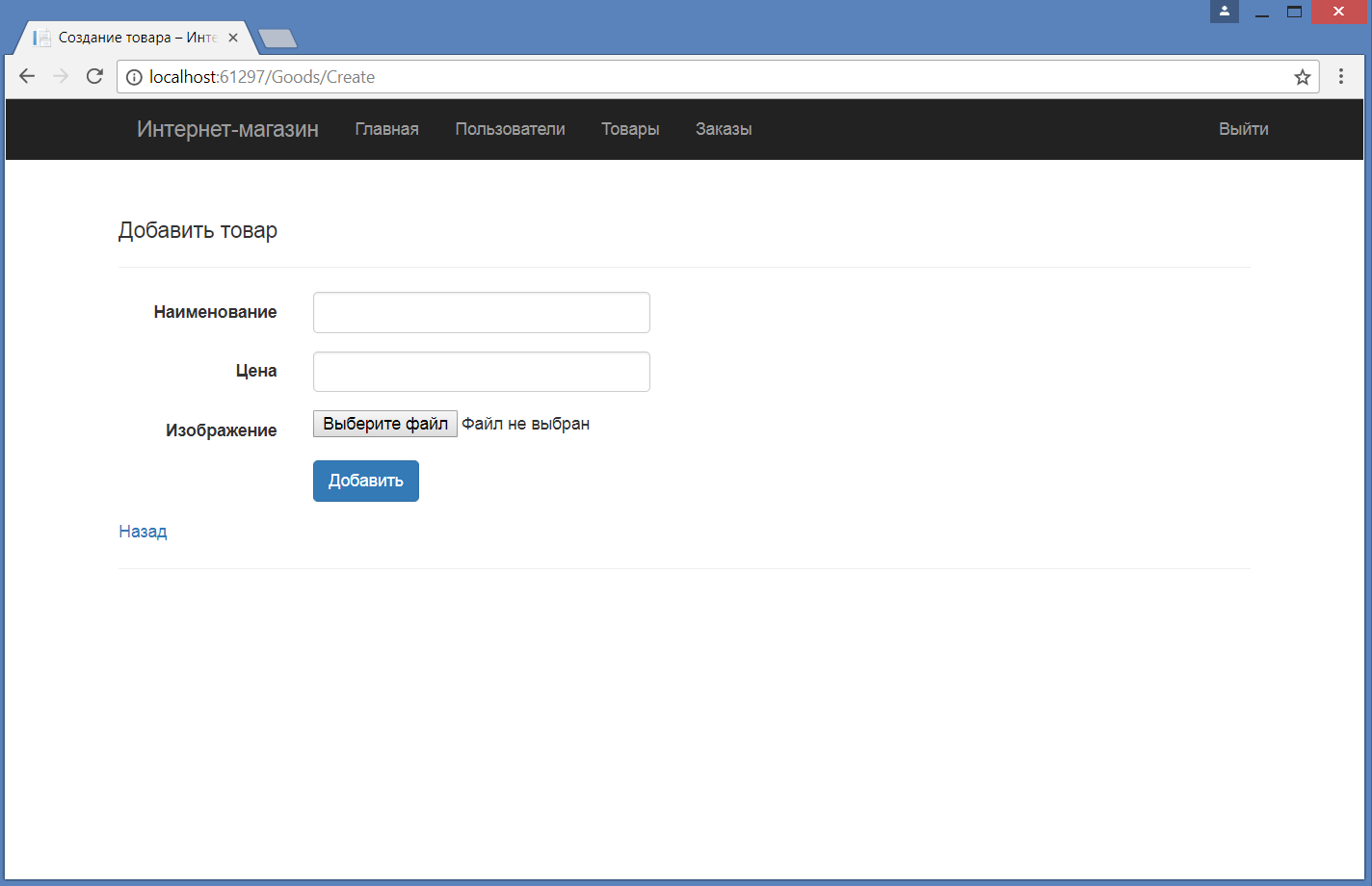


Рисунок 15. Страница добавления товара.

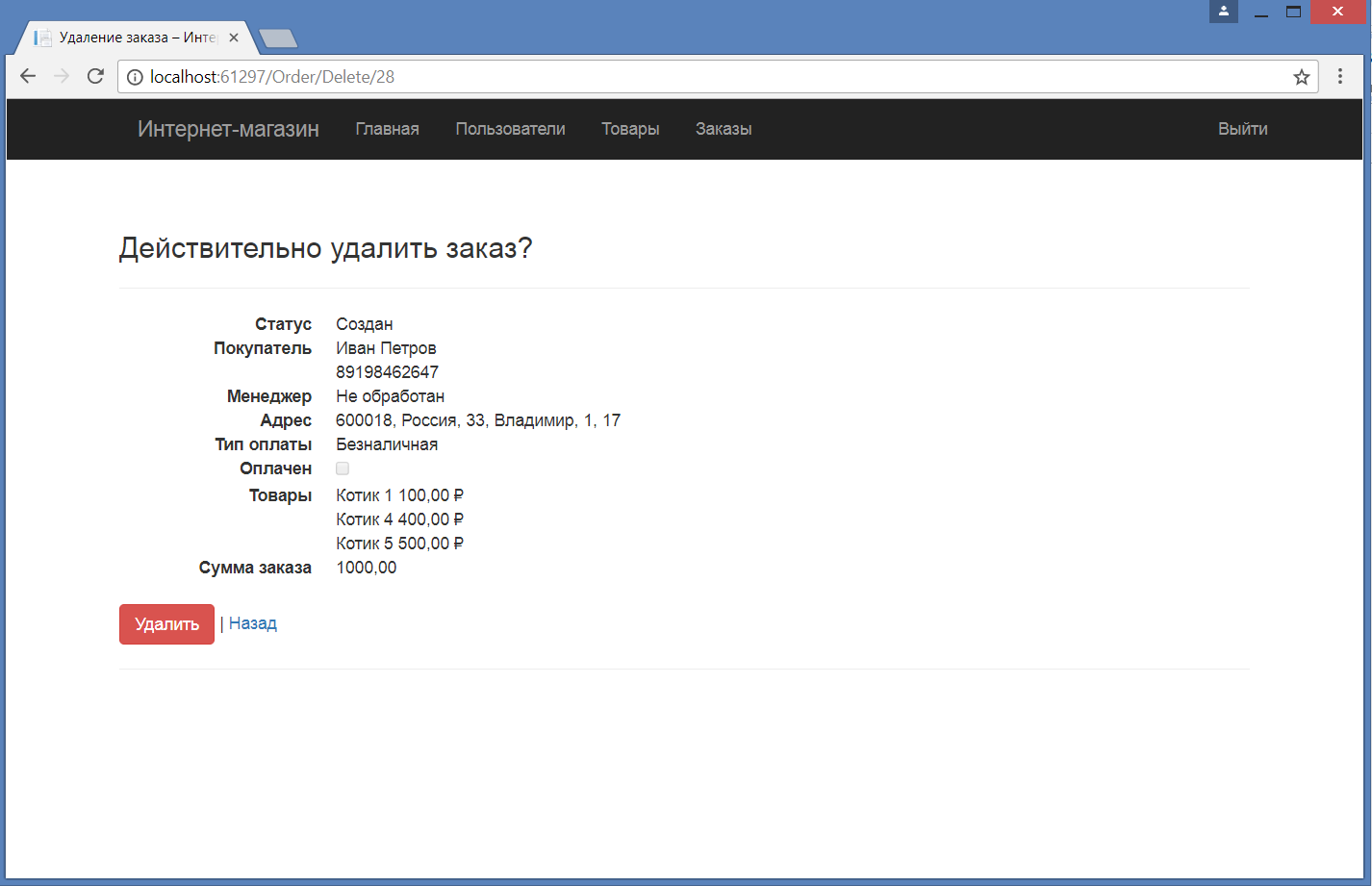


Рисунок 16. Страница удаления заказа.

# ТЕСТИРОВАНИЕ СИСТЕМЫ

Тестирование занимает важное место в жизненном цикле программного продукта. Ни один средний или, тем более, крупный проект не может обойтись без стадии тестирования. На этом этапе можно отловить много скрытых ошибок, которые могли бы проявиться во время работы приложения на стороне клиента.

В ходе курсового проектирования юнит-тестирование было применено при разработке веб-сервиса. Был протестирован единственный публичный метод сервиса. Исходные коды тестов представлены в приложении В. Результаты выполнения тестов представлены на рис. 17

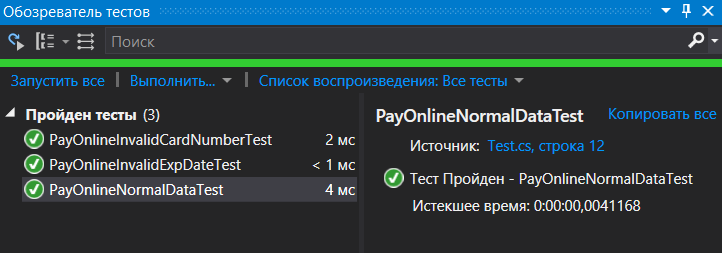


Рисунок 17. Результаты юнит-тестирования.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проектирования была спроектирована и реализована программная система автоматизации торговой деятельности предприятия в интернете.

Проектирование программного продукта было выполнено с использованием нотации UML Программное обеспечение было реализовано на языке C# в виде веб-приложения, построенного по архитектуре MVC на базе программной платформы Microsoft ASP .NET.

В ходе разработки были использованы различные технологии программирования: сторонние библиотеки, внешние веб-сервисы, системы контроля версий, и т.д. Также были выполнены юнит-тесты.

Данная программная система может быть использована торговыми предприятиями, с целью вывести и автоматизировать свою деятельность в новой сфере.

СПИСОК ИСПОЛЬЗВАННЫХ ИСТОЧНИКОВ

1. Фримен А. ASP.NET MVC 4 Framework с примерами на C# для профессионалов. 4-е издание. – Вильямс, 2014. 688 с.
2. NLog Wiki [Электронный ресурс]: GitHub. – Режим доступа: <https://github.com/nlog/nlog/wiki>.
3. Чакон С., Штрауб Б. Git для профессионального программиста. – СПб.:Питер, 2017. 496 с.

# ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММНОЙ СИСТЕМЫ

**Модели:**

public partial class AspNetUsers

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public AspNetUsers()

{

this.Message = new HashSet<Message>();

this.Message1 = new HashSet<Message>();

this.AspNetRoles = new HashSet<AspNetRoles>();

}

public string Id { get; set; }

[Required]

[Display(Name = "Имя")]

public string Name { get; set; }

[Required]

[Display(Name = "Фамилия")]

public string Surname { get; set; }

[Required]

[DataType(DataType.Date)]

[Display(Name = "Дата рождения")]

public System.DateTime Birthdate { get; set; }

[Required]

[DataType(DataType.EmailAddress)]

[Display(Name = "Электронная почта")]

public string Email { get; set; }

[Display(Name = "Подтверждение электронной почты")]

public bool EmailConfirmed { get; set; }

public string PasswordHash { get; set; }

public string SecurityStamp { get; set; }

[Required]

[DataType(DataType.PhoneNumber)]

[Display(Name = "Номер телефона")]

public string PhoneNumber { get; set; }

[Display(Name = "Подтверждение номера телефона")]

public bool PhoneNumberConfirmed { get; set; }

public bool TwoFactorEnabled { get; set; }

public Nullable<System.DateTime> LockoutEndDateUtc { get; set; }

[Display(Name = "Заблокирован")]

public bool LockoutEnabled { get; set; }

[Display(Name = "Осталось попыток входа")]

public int AccessFailedCount { get; set; }

[Display(Name = "Имя пользователя")]

public string UserName { get; set; }

public string Discriminator { get; set; }

[Display(Name = "Покупатель")]

public virtual Buyer Buyer { get; set; }

[Display(Name = "Сотрудник")]

public virtual Employee Employee { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Message> Message { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Message> Message1 { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

[Display(Name = "Роли")]

public virtual ICollection<AspNetRoles> AspNetRoles { get; set; }

}

public partial class AspNetRoles

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public AspNetRoles()

{

this.AspNetUsers = new HashSet<AspNetUsers>();

}

public string Id { get; set; }

[Required]

[Display(Name = "Название роли")]

public string Name { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<AspNetUsers> AspNetUsers { get; set; }

}

}

namespace OnlineShop.Models

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

public partial class Buyer

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Buyer()

{

this.Order = new HashSet<Order>();

}

public string UserId { get; set; }

[Display(Name = "Адрес")]

public virtual Address Address { get; set; }

[Display(Name = "Информация пользователя")]

public virtual AspNetUsers AspNetUsers { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

[Display(Name = "Заказ")]

public virtual ICollection<Order> Order { get; set; }

}

public partial class Address

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Address()

{

this.Order = new HashSet<Order>();

}

public string UserId { get; set; }

[Required]

[DataType(DataType.PostalCode)]

[Display(Name = "Индекс")]

public string Postcode { get; set; }

[Required]

[Display(Name = "Страна")]

public string Country { get; set; }

[Required]

[Display(Name = "Область")]

public string Region { get; set; }

[Required]

[Display(Name = "Город")]

public string City { get; set; }

[Required]

[Display(Name = "Улица")]

public string Street { get; set; }

[Required]

[Display(Name = "Дом")]

public string House { get; set; }

[Required]

[Display(Name = "Квартира")]

public string Flat { get; set; }

public virtual Buyer Buyer { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Order> Order { get; set; }

}

public partial class Employee

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Employee()

{

this.Order = new HashSet<Order>();

}

public string UserId { get; set; }

[Display(Name = "Заработная плата")]

public Nullable<decimal> MonthSalary { get; set; }

public virtual AspNetUsers AspNetUsers { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Order> Order { get; set; }

}

public partial class Goods

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Goods()

{

this.ShoppingCart = new HashSet<ShoppingCart>();

}

public int Id { get; set; }

[Required]

[Display(Name = "Наименование")]

public string Name { get; set; }

[Required]

[DataType(DataType.Currency)]

[Display(Name = "Цена")]

public decimal Price { get; set; }

[Display(Name = "Изображение")]

//[FileExtensions]

public byte[] Image { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<ShoppingCart> ShoppingCart { get; set; }

}

partial class Order

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Order()

{

this.ShoppingCart = new HashSet<ShoppingCart>();

}

public int Id { get; set; }

[Display(Name = "Статус")]

public short Status { get; set; }

[Display(Name = "Покупатель")]

public string BuyerId { get; set; }

[Display(Name = "Менеджер")]

public string ManagerId { get; set; }

[Display(Name = "Адрес")]

public string DeliveryAddressId { get; set; }

[Required]

[Display(Name = "Тип оплаты")]

public short PaymentType { get; set; }

[Display(Name = "Оплачен")]

public bool IsPayed { get; set; }

[Display(Name = "Сумма заказа")]

public decimal TotalCost { get; set; }

[Display(Name = "Адрес")]

public virtual Address Address { get; set; }

[Display(Name = "Покупатель")]

public virtual Buyer Buyer { get; set; }

[Display(Name = "Менеджер")]

public virtual Employee Employee { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

[Display(Name = "Товары")]

public virtual ICollection<ShoppingCart> ShoppingCart { get; set; }

}

public partial class ShoppingCart

{

public int Id { get; set; }

[Display(Name = "Товар")]

public int GoodsId { get; set; }

[Display(Name = "Заказ")]

public int OrderId { get; set; }

[Display(Name = "Товар")]

public virtual Goods Goods { get; set; }

[Display(Name = "Заказ")]

public virtual Order Order { get; set; }

}

public enum OrderStatus : short

{

[Display(Name = "Создан")]

Created,

[Display(Name = "Подтвержден")]

Confirmed,

[Display(Name = "Оплачен")]

Payed,

[Display(Name = "Доставляется")]

Delivering,

[Display(Name = "Завершен")]

Done,

[Display(Name = "Отклонен")]

Cancelled

}

public enum PaymentType : short

{

[Display(Name = "Наличная")]

Cash,

[Display(Name = "Безналичная")]

Cashless

}

public static class Extensions

{

public static TAttribute GetAttribute<TAttribute>(this Enum enumValue)

where TAttribute : Attribute

{

return enumValue.GetType()

.GetMember(enumValue.ToString())

.First()

.GetCustomAttribute<TAttribute>();

}

}

public class PurchaseInfo

{

[Required]

[DataType(DataType.CreditCard)]

[Display(Name = "Номер карты")]

public string CardNumber { get; set; }

[Required]

[Display(Name = "Год окончания действия карты")]

public int ExpYear { get; set; }

[Required]

[Display(Name = "Месяц окончания действия карты")]

public int ExpMonth { get; set; }

[Required]

[Display(Name = "Имя держателя карты")]

public string CardHolder { get; set; }

[Required]

[DataType(DataType.Password)]

[MinLength(3)]

[MaxLength(3)]

[Display(Name = "Код безопасности CVC")]

public string CVC { get; set; }

[Display(Name = "Сумма покупки")]

public decimal Sum { get; set; }

}

public class LocalShoppingCart

{

public LocalShoppingCart()

{

GoodsInCart = new List<Goods>();

}

public List<Goods> GoodsInCart { get; set; }

public void AddToCart(Goods goods)

{

GoodsInCart.Add(goods);

}

public void RemoveFromCart(int id)

{

Goods goodsToRemove = GoodsInCart.First(n => n.Id == id);

GoodsInCart.Remove(goodsToRemove);

}

}

**Контроллеры:**

public class AddressController : Controller

{

private AddressDAO addressDAO = new AddressDAO();

// GET: Address/Create

[Authorize(Roles = "Buyer")]

public ActionResult Create()

{

return View();

}

// POST: Address/Create

[HttpPost]

[Authorize(Roles = "Buyer")]

public ActionResult Create(Address address)

{

address.UserId = User.Identity.GetUserId();

if (addressDAO.Create(address))

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

else

return View();

}

// GET: Address/Edit/5

[Authorize(Roles = "Buyer")]

public ActionResult Edit(string id)

{

return View(addressDAO.GetById(id));

}

// POST: Address/Edit/5

[HttpPost]

[Authorize(Roles = "Buyer")]

public ActionResult Edit(Address address)

{

addressDAO.Update(address);

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

}

}

public class CartController : Controller

{

// GET: Cart

[Authorize(Roles = "Buyer")]

public ActionResult Index()

{

LocalShoppingCart cart = (LocalShoppingCart)Session["ShoppingCart"];

return View(cart.GoodsInCart);

}

[HttpPost]

[Authorize(Roles = "Buyer")]

public ActionResult Index(int id)

{

return Add(id);

}

[Authorize(Roles = "Buyer")]

public ActionResult Add(int id)

{

LocalShoppingCart cart = (LocalShoppingCart)Session["ShoppingCart"];

Goods goodsToAdd = new GoodsDAO().GetById(id);

cart.AddToCart(goodsToAdd);

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

}

// GET: Cart/Delete/5

[Authorize(Roles = "Buyer")]

public ActionResult Delete(int id)

{

Goods goodsToRemove = new GoodsDAO().GetById(id);

return View(goodsToRemove);

}

// POST: Cart/Delete/5

[HttpPost]

[Authorize(Roles = "Buyer")]

public ActionResult Delete(int id, Goods goods)

{

try

{

LocalShoppingCart cart = (LocalShoppingCart)Session["ShoppingCart"];

cart.RemoveFromCart(id);

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

public class GoodsController : Controller

{

GoodsDAO goodsDAO = new GoodsDAO();

// GET: Goods

public ActionResult Index()

{

return View(goodsDAO.GetAll());

}

// GET: Goods

[Authorize(Roles = "Admin")]

public ActionResult IndexList()

{

return View(goodsDAO.GetAll());

}

// GET: Goods/Create

[Authorize(Roles = "Admin")]

public ActionResult Create()

{

return View();

}

// POST: Goods/Create

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult Create(Goods goods, HttpPostedFileBase uploadImage)

{

try

{

if (uploadImage != null)

{

byte[] imageData = null;

using (var binaryReader = new BinaryReader(uploadImage.InputStream))

{

imageData = binaryReader.ReadBytes(uploadImage.ContentLength);

}

goods.Image = imageData;

}

goodsDAO.Create(goods);

return RedirectToAction("IndexList");

}

catch

{

return View();

}

}

public FileContentResult GetImage(int id)

{

Goods goods = goodsDAO.GetById(id);

if (goods != null)

{

return File(goods.Image, "image/jpeg");

}

else

{

return null;

}

}

// GET: Goods/Details/5

public ActionResult Details(int id)

{

return View(goodsDAO.GetById(id));

}

// GET: Goods/Edit/5

[Authorize(Roles = "Admin")]

public ActionResult Edit(int id)

{

return View(goodsDAO.GetById(id));

}

// POST: Goods/Edit/5

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult Edit(Goods goods, HttpPostedFileBase uploadImage)

{

try

{

if (uploadImage != null)

{

byte[] imageData = null;

using (var binaryReader = new BinaryReader(uploadImage.InputStream))

{

imageData = binaryReader.ReadBytes(uploadImage.ContentLength);

}

goods.Image = imageData;

}

goodsDAO.Update(goods);

return RedirectToAction("Index");

}

catch

{

return View();

}

}

// GET: Goods/Delete/5

[Authorize(Roles = "Admin")]

public ActionResult Delete(int id)

{

return View(goodsDAO.GetById(id));

}

// POST: Goods/Delete/5

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult Delete(int id, Goods goods)

{

try

{

goodsDAO.Delete(goods);

return RedirectToAction("IndexList");

}

catch

{

return View();

}

}

}

public class HomeController : Controller

{

public ActionResult Index()

{

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

}

}

public class OrderController : Controller

{

private OrderDAO orderDAO = new OrderDAO();

// GET: Order

[Authorize(Roles = "Admin, Manager")]

public ActionResult Index()

{

return View(orderDAO.GetAll());

}

// GET: Order/Details/5

[Authorize(Roles = "Admin, Manager")]

public ActionResult Details(int id)

{

return View(orderDAO.GetByIdWithDetails(id));

}

// GET: Order/Confirm/5

[Authorize(Roles = "Manager")]

public ActionResult Confirm(int id)

{

return View(orderDAO.GetById(id));

}

[HttpPost]

[Authorize(Roles = "Manager")]

public ActionResult Confirm(int id, string submitButton)

{

if (submitButton == "Confirm")

{

orderDAO.UpdateManager(id, User.Identity.GetUserId());

orderDAO.UpdateStatus(id, OrderStatus.Confirmed);

}

else

{

orderDAO.UpdateStatus(id, OrderStatus.Cancelled);

}

return RedirectToAction("Index");

}

// GET: Order/Send

[Authorize(Roles = "Manager")]

public ActionResult Send(int id)

{

return View(orderDAO.GetById(id));

}

// POST: Order/Send

[HttpPost]

[Authorize(Roles = "Manager")]

public ActionResult Send(Order order)

{

orderDAO.UpdateStatus(order.Id, OrderStatus.Delivering);

return RedirectToAction("Index");

}

// GET: Order/ConfirmDelivering

[Authorize(Roles = "Buyer")]

public ActionResult ConfirmDelivering(int id)

{

return View(orderDAO.GetById(id));

}

// POST: Order/ConfirmDelivering

[HttpPost]

[Authorize(Roles = "Buyer")]

public ActionResult ConfirmDelivering(Order order)

{

orderDAO.UpdateStatus(order.Id, OrderStatus.Done);

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

}

// GET: Order/ConfirmDelivering

[Authorize(Roles = "Buyer")]

public ActionResult Cancel(int id)

{

return View(orderDAO.GetById(id));

}

// POST: Order/ConfirmDelivering

[HttpPost]

[Authorize(Roles = "Buyer")]

public ActionResult Cancel(Order order)

{

orderDAO.UpdateStatus(order.Id, OrderStatus.Cancelled);

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

}

// GET: Order/Create

[Authorize(Roles = "Buyer")]

public ActionResult Create()

{

return View();

}

// POST: Order/Create

[HttpPost]

[Authorize(Roles = "Buyer")]

public ActionResult Create(Order order)

{

try

{

order.Status = (short)OrderStatus.Created;

string userId = User.Identity.GetUserId();

order.BuyerId = userId;

order.ManagerId = "default";

order.DeliveryAddressId = userId;

order.IsPayed = false;

decimal totalCost = 0;

List<Goods> goodsInOrder = (Session["ShoppingCart"] as LocalShoppingCart).GoodsInCart;

foreach(Goods goods in goodsInOrder)

totalCost += goods.Price;

order.TotalCost = totalCost;

Order addedOrder = orderDAO.Create(order);

if (orderDAO.AddGoodsToOrder(addedOrder, goodsInOrder))

Session["ShoppingCart"] = new LocalShoppingCart();

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

}

catch

{

return View();

}

}

// GET: Order/Delete/5

[Authorize(Roles = "Admin")]

public ActionResult Delete(int id)

{

return View(orderDAO.GetById(id));

}

// POST: Order/Delete/5

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult Delete(int id, Order order)

{

try

{

Order deletedOrder = orderDAO.GetByIdWithDetails(id);

orderDAO.DeleteGoodsFromOrder(deletedOrder);

orderDAO.Delete(deletedOrder);

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

public class PayController : Controller

{

// GET: Pay

[Authorize(Roles = "Buyer")]

public ActionResult Online(int id, decimal sum)

{

ViewData["orderId"] = id;

PurchaseInfo info = new PurchaseInfo { Sum = sum };

return View(info);

}

// GET: Pay

[HttpPost]

[Authorize(Roles = "Buyer")]

public ActionResult Online(int orderId, PurchaseInfo purchase)

{

PayServiceSoapClient payService = new PayServiceSoapClient();

bool payResult = payService.PayOnline(

purchase.CardNumber,

purchase.ExpYear,

purchase.ExpMonth,

purchase.CardHolder,

purchase.CVC,

purchase.Sum

);

if(payResult)

{

OrderDAO orderDAO = new OrderDAO();

orderDAO.UpdateStatus(orderId, OrderStatus.Payed);

orderDAO.UpdateIsPayed(orderId, true);

}

return RedirectToAction("Orders", "Profile");

}

}

public class ProfileController : Controller

{

private BuyerDAO buyerDAO = new BuyerDAO();

// GET: Profile

[Authorize(Roles = "Buyer")]

public ActionResult Index()

{

string userId = User.Identity.GetUserId();

return View(buyerDAO.GetById(userId));

}

// GET: Profile/Orders

[Authorize(Roles = "Buyer")]

public ActionResult Orders()

{

string userId = User.Identity.GetUserId();

return View(new OrderDAO().GetAllForUserWithDetails(userId));

}

}

public class UserController : Controller

{

private UserDAO userDAO = new UserDAO();

// GET: User

[Authorize(Roles = "Admin")]

public ActionResult Index()

{

return View(userDAO.GetAll());

}

// GET: User/Details/5

[Authorize(Roles = "Admin")]

public ActionResult Details(string id)

{

return View(userDAO.GetById(id));

}

// GET: User/Create

[Authorize(Roles = "Admin")]

public ActionResult Create()

{

return RedirectToAction("RegisterByAdmin", "Account");

}

// GET: User/Edit/5

[Authorize(Roles = "Admin")]

public ActionResult Edit(string id)

{

return View(userDAO.GetById(id));

}

// POST: User/Edit/5

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult Edit(AspNetUsers user)

{

try

{

userDAO.Update(user);

return RedirectToAction("Index");

}

catch

{

return View();

}

}

// GET: User/EditByUser/5

[Authorize(Roles = "Buyer")]

public ActionResult EditByUser(string id)

{

return View(userDAO.GetById(id));

}

// POST: User/EditByUser/5

[HttpPost]

[Authorize(Roles = "Buyer")]

public ActionResult EditByUser(string id, AspNetUsers user)

{

userDAO.Update(user);

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

}

// GET: User/Delete/5

[Authorize(Roles = "Admin")]

public ActionResult Delete(string id)

{

return View(userDAO.GetById(id));

}

// POST: User/Delete/5

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult Delete(AspNetUsers user)

{

try

{

userDAO.Delete(user);

return RedirectToAction("Index");

}

catch

{

return View(user);

}

}

}

public class UserRoleController : Controller

{

private UserRoleDAO userRoleDAO = new UserRoleDAO();

// GET: UserRole

[Authorize(Roles = "Admin")]

public ActionResult Index(string id)

{

ViewData["userId"] = id;

return View(userRoleDAO.GetAllForUser(id));

}

// GET: UserRole/Add

[Authorize(Roles = "Admin")]

public ActionResult Add(string id)

{

ViewData["userId"] = id;

ViewData["roleList"] = new SelectList(userRoleDAO.GetAll(), "Name", "Name");

return View();

}

// POST: UserRole/Add

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult Add(string id, AspNetRoles role)

{

try

{

AspNetRoles addRole = userRoleDAO.GetByName(role.Name);

userRoleDAO.AddRole(id, addRole);

if (role.Name.Equals("Manager") || role.Name.Equals("Admin"))

new EmployeeDAO().Create(id);

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

}

catch

{

return View();

}

}

// GET: UserRole/Delete/5

[Authorize(Roles = "Admin")]

public ActionResult Delete(string id, string name)

{

return View(userRoleDAO.GetByName(name));

}

// POST: UserRole/Delete/5

[HttpPost]

[Authorize(Roles = "Admin")]

public ActionResult Delete(string id, AspNetRoles role)

{

try

{

userRoleDAO.RemoveRole(id, role);

if (role.Name.Equals("Manager") || role.Name.Equals("Admin"))

new EmployeeDAO().Delete(id);

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

}

catch

{

return View();

}

}

}

**DAO**

public class AddressDAO

{

private Entities entities = new Entities();

private static Logger logger = LogManager.GetCurrentClassLogger();

public bool Create(Address input)

{

bool result;

try

{

entities.Address.Add(input);

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка добавления адреса");

result = false;

}

logger.Debug("Добавление адреса выполнено успешно");

return result;

}

public bool Delete(Address input)

{

bool result;

try

{

entities.Address.Remove(input);

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка удаления адреса");

result = false;

}

logger.Debug("Удаление адреса выполнено успешно");

return result;

}

public IEnumerable<Address> GetAll()

{

IEnumerable<Address> result;

try

{

result = entities.Address;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения всех адресов");

result = null;

}

logger.Debug("Получение всех адресов выполнено успешно");

return result;

}

public Address GetById(string id)

{

Address result;

try

{

result = entities.Address.FirstOrDefault(n => n.UserId == id);

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения адреса по id");

result = null;

}

logger.Debug("Получение адреса по id выполнено успешно");

return result;

}

public bool Update(Address input)

{

bool result;

try

{

Address current = entities.Address.FirstOrDefault(n => n.UserId == input.UserId);

current.Postcode = input.Postcode;

current.Country = input.Country;

current.Region = input.Region;

current.City = input.City;

current.Street = input.Street;

current.House = input.House;

current.Flat = input.Flat;

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка обновления адреса");

result = false;

}

logger.Debug("Обновление адреса выполнено успешно");

return result;

}

}

public class BuyerDAO

{

private Entities entities = new Entities();

private static Logger logger = LogManager.GetCurrentClassLogger();

public bool Create(string id)

{

bool result;

try

{

Buyer buyer = new Buyer { UserId = id };

entities.Buyer.Add(buyer);

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка добавления товара");

result = false;

}

logger.Debug("Добавление товара выполнено успешно");

return result;

}

public bool Delete(Buyer input)

{

bool result;

try

{

entities.Buyer.Remove(input);

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка удаления покупателя");

result = false;

}

logger.Debug("Удаление покупателя выполнено успешно");

return result;

}

public Buyer GetById(string id)

{

Buyer result;

try

{

result = entities.Buyer.FirstOrDefault(n => n.UserId == id);

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения покупателя по id");

result = null;

}

logger.Debug("Получение покупателя по id выполнено успешно");

return result;

}

public Buyer GetByIdWithDetails(string id)

{

Buyer result;

try

{

result = entities.

Buyer.

Where(n => n.UserId == id).

Include("Address").

Include("AspNetUsers").

Include("Orders").

FirstOrDefault();

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения расширенной информации покупателя по id");

result = null;

}

logger.Debug("Получение расширенной информации покупателя по id выполнено успешно");

return result;

}

}

public class EmployeeDAO

{

private Entities entities = new Entities();

private static Logger logger = LogManager.GetCurrentClassLogger();

public bool Create(string id)

{

bool result;

try

{

Employee employee = new Employee { UserId = id, MonthSalary = 10000 };

entities.Employee.Add(employee);

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка добавления сотрудника");

result = false;

}

logger.Debug("Добавление сотрудника выполнено успешно");

return result;

}

public bool Delete(string id)

{

bool result;

try

{

Employee employee = entities.Employee.FirstOrDefault(n => n.UserId == id);

entities.Employee.Remove(employee);

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка удаления сотрудника");

result = false;

}

logger.Debug("Удаление сотрудника выполнено успешно");

return result;

}

}

public class GoodsDAO

{

private Entities entities = new Entities();

private static Logger logger = LogManager.GetCurrentClassLogger();

public bool Create(Goods input)

{

bool result;

try

{

entities.Goods.Add(input);

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка добавления товара");

result = false;

}

logger.Debug("Добавление товара выполнено успешно");

return result;

}

public bool Delete(Goods input)

{

bool result;

try

{

Goods rmGoods = entities.Goods.FirstOrDefault(n => n.Id == input.Id);

entities.Goods.Remove(rmGoods);

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка удаления товара");

result = false;

}

logger.Debug("Удаление товара выполнено успешно");

return result;

}

public IEnumerable<Goods> GetAll()

{

IEnumerable<Goods> result;

try

{

result = entities.Goods;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения всех товаров");

result = null;

}

logger.Debug("Получение всех товаров выполнено успешно");

return result;

}

public Goods GetById(int id)

{

Goods result;

try

{

result = entities.Goods.FirstOrDefault(n => n.Id == id);

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения товара по id");

result = null;

}

logger.Debug("Получение товара по id выполнено успешно");

return result;

}

public bool Update(Goods input)

{

bool result;

try

{

Goods current = entities.Goods.FirstOrDefault(n => n.Id == input.Id);

current.Name = input.Name;

current.Price = input.Price;

current.Image = input.Image;

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка обновления товара");

result = false;

}

logger.Debug("Обновление товара выполнено успешно");

return result;

}

}

public class OrderDAO

{

private Entities entities = new Entities();

private static Logger logger = LogManager.GetCurrentClassLogger();

public Order Create(Order order)

{

Order result;

try

{

result = entities.Order.Add(order);

entities.SaveChanges();

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка добавления заказа");

result = null;

}

logger.Debug("Добавление заказа выполнено успешно");

return result;

}

public bool AddGoodsToOrder(Order order, List<Goods> goodsInOrder)

{

bool result;

try

{

foreach (Goods goods in goodsInOrder)

{

ShoppingCart item = new ShoppingCart { OrderId = order.Id, GoodsId = goods.Id };

entities.ShoppingCart.Add(item);

}

result = entities.SaveChanges() == goodsInOrder.Count;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка добавления товара в заказ");

result = false;

}

logger.Debug("Добавление товара в заказ выполнено успешно");

return result;

}

public bool Delete(Order input)

{

bool result;

try

{

Order current = GetById(input.Id);

entities.Order.Remove(current);

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка удаления заказа");

result = false;

}

logger.Debug("Удаление заказа выполнено успешно");

return result;

}

public bool DeleteGoodsFromOrder(Order order)

{

bool result;

try

{

List<ShoppingCart> cart = entities.ShoppingCart.Where(n => n.OrderId == order.Id).ToList();

foreach (var item in cart)

{

entities.ShoppingCart.Remove(item);

}

result = entities.SaveChanges() == cart.Count;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка удаления товара из заказа");

result = false;

}

logger.Debug("Удаление товара из заказа выполнено успешно");

return result;

}

public IEnumerable<Order> GetAll()

{

IEnumerable<Order> result;

try

{

result = entities.Order;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения всех заказов");

result = null;

}

logger.Debug("Получение всех заказов выполнено успешно");

return result;

}

public IEnumerable<Order> GetAllForUserWithDetails(string id)

{

IEnumerable<Order> result;

try

{

result = entities.

Order.

Where(n => n.BuyerId == id).

Include("Employee.AspNetUsers").

Include("Address").

Include("ShoppingCart");

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения всех заказов с расширенной информацией для пользователя по id");

result = null;

}

logger.Debug("Получение всех заказов с расширенной информацией для пользователя по id выполнено успешно");

return result;

}

public Order GetById(int id)

{

Order result;

try

{

result = entities.Order.FirstOrDefault(n => n.Id == id);

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения заказа по id");

result = null;

}

logger.Debug("Получение заказа по id выполнено успешно");

return result;

}

public Order GetByIdWithDetails(int id)

{

Order result;

try

{

result = entities.

Order.

Where(n => n.Id == id).

Include("Buyer.AspNetUsers").

Include("Employee.AspNetUsers").

Include("Address").

Include("ShoppingCart").

FirstOrDefault();

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения заказа с расширенной информацией по id");

result = null;

}

logger.Debug("Получение заказа с расширенной информацией по id выполнено успешно");

return result;

}

public bool UpdateManager(int id, string managerId)

{

bool result;

try

{

Order current = entities.Order.FirstOrDefault(n => n.Id == id);

current.ManagerId = managerId;

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка обновления менеджера у заказа");

result = false;

}

logger.Debug("Обновление менеджера у заказа выполнено успешно");

return result;

}

public bool UpdateIsPayed(int id, bool value)

{

bool result;

try

{

Order current = entities.Order.FirstOrDefault(n => n.Id == id);

current.IsPayed = value;

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка обновления поля IsPayed у заказа");

result = false;

}

logger.Debug("Обновление поля IsPayed у заказа выполнено успешно");

return result;

}

public bool UpdateStatus(int id, OrderStatus status)

{

bool result;

try

{

Order current = entities.Order.FirstOrDefault(n => n.Id == id);

current.Status = (short)status;

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка обновления статуса заказа");

result = false;

}

logger.Debug("Обновление статуса заказа выполнено успешно");

return result;

}

}

public class UserDAO

{

private Entities entities = new Entities();

private static Logger logger = LogManager.GetCurrentClassLogger();

public bool Delete(AspNetUsers input)

{

bool result = true;

try

{

UserRoleDAO roleDAO = new UserRoleDAO();

IEnumerable<AspNetRoles> roles = roleDAO.GetAllForUser(input.Id);

AspNetUsers user = entities.AspNetUsers.FirstOrDefault(n => n.Id == input.Id);

foreach (AspNetRoles item in roles)

{

if(item.Name == "Buyer")

{

BuyerDAO buyerDAO = new BuyerDAO();

result &= buyerDAO.Delete(buyerDAO.GetById(user.Id));

AddressDAO addressDAO = new AddressDAO();

result &= addressDAO.Delete(addressDAO.GetById(user.Id));

}

else

{

EmployeeDAO employeeDAO = new EmployeeDAO();

result &= employeeDAO.Delete(user.Id);

}

result &= roleDAO.RemoveRole(input.Id, item);

}

entities.AspNetUsers.Remove(user);

result &= entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка удаления пользователя");

result = false;

}

logger.Debug("Удаление пользователя выполнено успешно");

return result;

}

public IEnumerable<AspNetUsers> GetAll()

{

IEnumerable<AspNetUsers> result;

try

{

result = entities.AspNetUsers;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения всех пользователей");

result = null;

}

logger.Debug("Получение всех пользователей выполнено успешно");

return result;

}

public AspNetUsers GetById(string id)

{

AspNetUsers result;

try

{

result = entities.AspNetUsers.FirstOrDefault(n => n.Id == id);

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения пользователя по id");

result = null;

}

logger.Debug("Получение пользователя по id выполнено успешно");

return result;

}

public AspNetUsers GetByIdWithRoles(string id)

{

AspNetUsers result;

try

{

result = entities.

AspNetUsers.

Include("AspNetRoles").

FirstOrDefault(n => n.Id == id);

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения пользователя с ролями по id");

result = null;

}

logger.Debug("Получение пользователя с ролями по id выполнено успешно");

return result;

}

public bool Update(AspNetUsers input)

{

bool result;

try

{

AspNetUsers current = entities.AspNetUsers.FirstOrDefault(n => n.Id == input.Id);

current.Name = input.Name;

current.Surname = input.Surname;

current.Email = input.Email;

current.PhoneNumber = input.PhoneNumber;

current.PhoneNumberConfirmed = input.PhoneNumberConfirmed;

current.LockoutEnabled = input.LockoutEnabled;

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка обновления пользователя");

result = false;

}

logger.Debug("Обновление пользователя выполнено успешно");

return result;

}

}

public class UserRoleDAO

{

private Entities entities = new Entities();

private static Logger logger = LogManager.GetCurrentClassLogger();

public IEnumerable<AspNetRoles> GetAllForUser(string id)

{

IEnumerable<AspNetRoles> result;

try

{

result = entities.

AspNetUsers.

Include("AspNetRoles").

Where(n => n.Id == id).

Select(n => n.AspNetRoles).

FirstOrDefault();

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения всех ролей для пользователя по id");

result = null;

}

logger.Debug("Получение всех ролей для пользователя по id выполнено успешно");

return result;

}

public IEnumerable<AspNetRoles> GetAll()

{

IEnumerable<AspNetRoles> result;

try

{

result = entities.AspNetRoles;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения всех ролей");

result = null;

}

logger.Debug("Получение всех ролей выполнено успешно");

return result;

}

public AspNetRoles GetByName(string name)

{

AspNetRoles result;

try

{

result = entities.AspNetRoles.FirstOrDefault(n => n.Name == name);

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения роли по имени");

result = null;

}

logger.Debug("Получение роли по имени выполнено успешно");

return result;

}

public AspNetRoles GetById(string id)

{

AspNetRoles result;

try

{

result = entities.AspNetRoles.FirstOrDefault(n => n.Id == id);

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка получения роли по id");

result = null;

}

logger.Debug("Получение роли по id выполнено успешно");

return result;

}

public bool AddRole(string userId, AspNetRoles role)

{

bool result;

try

{

AspNetUsers changedUser = entities.AspNetUsers.FirstOrDefault(n => n.Id == userId);

changedUser.AspNetRoles.Add(role);

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка добавления роли пользователю");

result = false;

}

logger.Debug("Добавление роли пользователю выполнено успешно");

return result;

}

public bool RemoveRole(string userId, AspNetRoles role)

{

bool result;

try

{

AspNetUsers changedUser = entities.AspNetUsers.FirstOrDefault(n => n.Id == userId);

AspNetRoles rmRole = entities.AspNetRoles.FirstOrDefault(n => n.Name == role.Name);

changedUser.AspNetRoles.Remove(rmRole);

result = entities.SaveChanges() == 1;

}

catch (Exception ex)

{

logger.Error(ex, "Ошибка удаления роли у пользователя");

result = false;

}

logger.Debug("Удаление роли у пользователя выполнено успешно");

return result;

}

}

**Файлы конфигурации приложения:**

Global.asax.cs

public class MvcApplication : HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

protected void Session\_Start(object sender, EventArgs e)

{

Session["ShoppingCart"] = new LocalShoppingCart();

}

}

Mvc.sitemap

<?xml version="1.0" encoding="utf-8" ?>

<mvcSiteMap xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://mvcsitemap.codeplex.com/schemas/MvcSiteMap-File-4.0"

xsi:schemaLocation="http://mvcsitemap.codeplex.com/schemas/MvcSiteMap-File-4.0 MvcSiteMapSchema.xsd">

<mvcSiteMapNode title="Главная" controller="Home" action="Index">

<mvcSiteMapNode title="Пользователи" controller="User" action="Index" roles="Admin"/>

<mvcSiteMapNode title="Товары" controller="Goods" action="IndexList" roles="Admin"/>

<mvcSiteMapNode title="Заказы" controller="Order" action="Index" roles="Admin, Manager" />

</mvcSiteMapNode>

</mvcSiteMap>

NLog.config

<?xml version="1.0" encoding="utf-8" ?>

<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.nlog-project.org/schemas/NLog.xsd NLog.xsd"

autoReload="true"

throwExceptions="false"

internalLogLevel="Off" internalLogFile="c:\temp\nlog-internal.log">

<variable name="myvar" value="myvalue"/>

<targets>

<target name="logfile" xsi:type="File" layout="${longdate} ${logger} ${message}" encoding="utf-8" fileName="logs/log-${shortdate}.txt" />

</targets>

<rules>

<logger name="\*" minlevel="Trace" writeTo="logfile" />

</rules>

</nlog>

Web.config

<?xml version="1.0" encoding="utf-8"?>

<!--

For more information on how to configure your ASP.NET application, please visit

https://go.microsoft.com/fwlink/?LinkId=301880

-->

<configuration>

<configSections>

<section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />

</configSections>

<connectionStrings>

<add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet-OnlineShop-20171118091036.mdf;Initial Catalog=aspnet-OnlineShop-20171118091036;Integrated Security=True" providerName="System.Data.SqlClient" />

<add name="Entities" connectionString="metadata=res://\*/Models.Model.csdl|res://\*/Models.Model.ssdl|res://\*/Models.Model.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=(LocalDb)\MSSQLLocalDB;attachdbfilename=|DataDirectory|\aspnet-OnlineShop-20171118091036.mdf;initial catalog=aspnet-OnlineShop-20171118091036;integrated security=True;multipleactiveresultsets=True;application name=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

<appSettings>

<add key="webpages:Version" value="3.0.0.0" />

<add key="webpages:Enabled" value="false" />

<add key="ClientValidationEnabled" value="true" />

<add key="UnobtrusiveJavaScriptEnabled" value="true" />

<add key="MvcSiteMapProvider\_IncludeAssembliesForScan" value="OnlineShop" />

<add key="MvcSiteMapProvider\_UseExternalDIContainer" value="false" />

<add key="MvcSiteMapProvider\_ScanAssembliesForSiteMapNodes" value="true" />

<add key="MvcSiteMapProvider\_SecurityTrimmingEnabled" value="true" />

</appSettings>

<system.web>

<authentication mode="None" />

<compilation debug="true" targetFramework="4.6.1" />

<httpRuntime targetFramework="4.6.1" />

<httpModules>

<add name="ApplicationInsightsWebTracking" type="Microsoft.ApplicationInsights.Web.ApplicationInsightsHttpModule, Microsoft.AI.Web" />

</httpModules>

<pages>

<namespaces>

<add namespace="MvcSiteMapProvider.Web.Html" />

<add namespace="MvcSiteMapProvider.Web.Html.Models" />

</namespaces>

</pages>

</system.web>

<system.webServer>

<modules>

<remove name="FormsAuthentication" />

<remove name="ApplicationInsightsWebTracking" />

<add name="ApplicationInsightsWebTracking" type="Microsoft.ApplicationInsights.Web.ApplicationInsightsHttpModule, Microsoft.AI.Web" preCondition="managedHandler" />

<remove name="UrlRoutingModule-4.0" />

<add name="UrlRoutingModule-4.0" type="System.Web.Routing.UrlRoutingModule" />

</modules>

<validation validateIntegratedModeConfiguration="false" />

</system.webServer>

<runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity name="Microsoft.Owin.Security" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="0.0.0.0-3.0.1.0" newVersion="3.0.1.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="Microsoft.Owin.Security.OAuth" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="0.0.0.0-3.0.1.0" newVersion="3.0.1.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="Microsoft.Owin.Security.Cookies" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="0.0.0.0-3.0.1.0" newVersion="3.0.1.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="Microsoft.Owin" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="0.0.0.0-3.0.1.0" newVersion="3.0.1.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="Newtonsoft.Json" culture="neutral" publicKeyToken="30ad4fe6b2a6aeed" />

<bindingRedirect oldVersion="0.0.0.0-6.0.0.0" newVersion="6.0.0.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.Optimization" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="1.0.0.0-1.1.0.0" newVersion="1.1.0.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="WebGrease" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="0.0.0.0-1.5.2.14234" newVersion="1.5.2.14234" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.Helpers" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.Mvc" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="0.0.0.0-5.2.3.0" newVersion="5.2.3.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.WebPages" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0" />

</dependentAssembly>

</assemblyBinding>

</runtime>

<entityFramework>

<defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework" />

<providers>

<provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />

</providers>

</entityFramework>

<system.codedom>

<compilers>

<compiler language="c#;cs;csharp" extension=".cs" type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.CSharpCodeProvider, Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=1.0.7.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" warningLevel="4" compilerOptions="/langversion:default /nowarn:1659;1699;1701" />

<compiler language="vb;vbs;visualbasic;vbscript" extension=".vb" type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.VBCodeProvider, Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=1.0.7.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" warningLevel="4" compilerOptions="/langversion:default /nowarn:41008 /define:\_MYTYPE=\&quot;Web\&quot; /optionInfer+" />

</compilers>

</system.codedom>

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding name="PayServiceSoap" />

</basicHttpBinding>

</bindings>

<client>

<endpoint address="http://localhost:49781/PayService.asmx" binding="basicHttpBinding" bindingConfiguration="PayServiceSoap" contract="PayService.PayServiceSoap" name="PayServiceSoap" />

</client>

</system.serviceModel>

</configuration>

# ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ВЕБ-СЕРВИСА

namespace PayService

{

[WebService(Namespace = "http://tempuri.org/")]

[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]

[System.ComponentModel.ToolboxItem(false)]

public class PayService : WebService

{

[WebMethod]

public bool PayOnline(string cardNumber, int expYear, int expMonth, string cardHolderName, string cvcCode, decimal purchaseSum)

{

ClientAccount account = CheckInfo(cardNumber, expYear, expMonth, cardHolderName, cvcCode);

if (account == null) return false;

ClientAccountDAO clientAccountDAO = new ClientAccountDAO();

return clientAccountDAO.WriteOff(account.CardNumber, purchaseSum);

}

private ClientAccount CheckInfo(string cardNumber, int expYear, int expMonth, string cardHolderName, string cvcCode)

{

ClientAccount currentAccount = new ClientAccount(cardNumber, expYear, expMonth, cardHolderName.ToUpper(), cvcCode);

DateTime expDate = new DateTime(expYear, expMonth, 1);

if (expDate < DateTime.Now || cardNumber.Length != 16)

return null;

ClientAccountDAO clientAccountDAO = new ClientAccountDAO();

ClientAccount expectedAccount = clientAccountDAO.GetByCardNumber(cardNumber);

if (currentAccount.Equals(expectedAccount))

return expectedAccount;

else

return null;

}

}

}

namespace PayService.Model

{

using System;

using System.Collections.Generic;

public partial class ClientAccount

{

public ClientAccount()

{

}

public ClientAccount(string cardNumber, int expYear, int expMonth, string cardHolder, string cvcCode)

{

CardNumber = cardNumber;

CardHolder = cardHolder;

CVC = cvcCode;

ExpYear = expYear;

ExpMonth = expMonth;

}

public string CardNumber { get; set; }

public string CardHolder { get; set; }

public string CVC { get; set; }

public decimal Balance { get; set; }

public int ExpYear { get; set; }

public int ExpMonth { get; set; }

public override bool Equals(object obj)

{

var account = obj as ClientAccount;

return account != null &&

CardNumber == account.CardNumber &&

CardHolder == account.CardHolder &&

CVC == account.CVC &&

ExpYear == account.ExpYear &&

ExpMonth == account.ExpMonth;

}

}

}

namespace PayService.DAO

{

public class ClientAccountDAO

{

private PayDataEntities entities = new PayDataEntities();

public ClientAccount GetByCardNumber(string cardNumber) => entities.ClientAccount.FirstOrDefault(n => n.CardNumber == cardNumber);

public bool WriteOff(string cardNumber, decimal writeOffSum)

{

ClientAccount account = GetByCardNumber(cardNumber);

if (account.Balance - writeOffSum >= 0)

{

account.Balance -= writeOffSum;

return entities.SaveChanges() == 1 ? true : false;

}

else

return false;

}

}

}

# ПРИЛОЖЕНИЕ В. ИСХОДНЫЙ КОД ЮНИТ-ТЕСТОВ

[TestClass]

public class Test

{

[TestMethod]

public void PayOnlineNormalDataTest()

{

PayService.PayService ps = new PayService.PayService();

bool actual = ps.PayOnline("1234123412340001", 2018, 5, "IVAN PETROV", "123", 0);

Assert.IsTrue(actual);

}

[TestMethod]

public void PayOnlineInvalidCardNumberTest()

{

PayService.PayService ps = new PayService.PayService();

bool actual = ps.PayOnline("00000000", 2018, 5, "IVAN PETROV", "123", 0);

Assert.IsFalse(actual);

}

[TestMethod]

public void PayOnlineInvalidExpDateTest()

{

PayService.PayService ps = new PayService.PayService();

bool actual = ps.PayOnline("1234123412340001", 2007, 9, "IVAN PETROV", "123", 0);

Assert.IsFalse(actual);

}

}