МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

«Основы веб-разработки-»

Направление подготовки 09.03.02 «Информационные системы и технологии»
Направленность (профиль) «Прикладное программирование и интернет-технологии»

Ставрополь
2026

ВВЕДЕНИЕ

1. Цель и задачи освоения дисциплины (модуля)

Целью изучения дисциплины «Основы веб-разработки» является формирование у студентов знаний, умений и навыков применения технологий реализации электронного бизнеса, использования методов построения электронного бизнеса, его инструментария при работе на различных сегментах рынка; базовая подготовка по технологиям электронного бизнеса и навыки по применению данных технологий, достаточные для последующей самостоятельной работы со специальной литературой и изучения специальных дисциплин.

Задачами дисциплины «Основы веб-разработки» являются:

1) формирование у студентов необходимых знаний по дисциплине;

2) ознакомление с техническими, алгоритмическими, программными и технологическими решениями, используемыми в данной области;

3) создание и развитие у студентов умений методического и прикладного характера, необходимых в электронном бизнесе;

4) выработка практических навыков аналитического и экспериментального исследования основных методов и средств, используемых в области, изучаемой в рамках данной дисциплины;

5) формирование набора компетенций.

Лабораторная работа № 1. Структура веб-проекта.

Цель работы: разработка структуры веб-проекта.

Теоретическое обоснование

Структура сайта – это четкая схема, по которой будет разрабатываться ресурс. Наглядная структура покажет вид будущего сайта.

Существуют классические составляющие структуры любого сайта, но есть и дополнительные опции, которые преследуют свою цель. Например, в интернет-магазине обязательно должна быть страница для оплаты, а в лендинге – место под призыв к действию.

Любой ресурс создается согласно определенному плану. Именно он отображает структуру сайта. В плане обязательно указывается, как должны располагаться страницы ресурса относительно друг друга. Чаще всего это делается в виде графической схемы с отдельными блоками и связывающими их стрелками (рисунок 1.1).

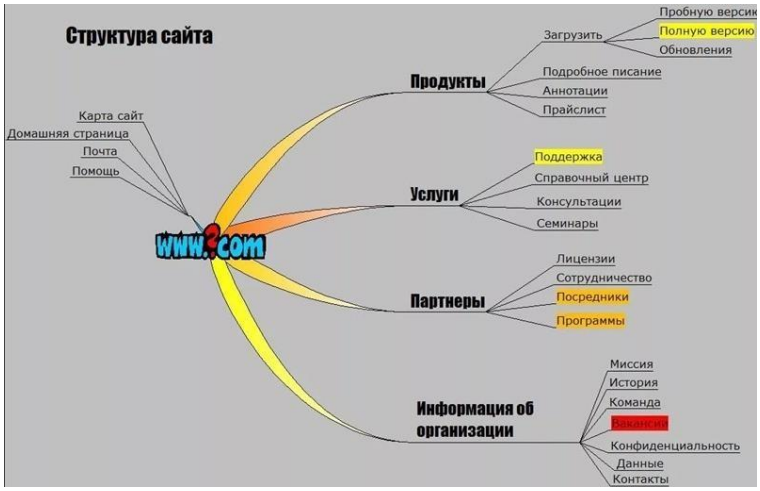


Рисунок 1.1 – Графическая схема структуры сайта

Структура может быть внешней и внутренней. Внешняя представляет собой макет страницы, на котором блоками обозначены отдельные ее элементы. Внутренняя структура включает в себя категории и разделы сайта и отношение к ним отдельных страниц. Ее сложнее всего организовать правильно.

Схема ресурса в первую очередь зависит от его специфики и направленности, того, какую бизнес-задачу он решает. Если речь идет об одностраничном сайте-визитке, то составить его план довольно просто: на одной странице будет размещаться вся основная информация. Другое дело – информационный портал или интернет-магазин.

Требования к структуре сайта могут быть разными, однако независимо от них информация должна подаваться таким образом, чтобы пользователи:

- получали исчерпывающий ответ на свои вопросы;
- понимали логику сайта;
- увлекались опубликованным материалом и стремились найти и другие статьи.

Помимо перечисленного, размещенный контент должен улучшать положение сайта в поисковых выдачах.

Формирование четкой структуры ресурса дает следующие преимущества:

- позволяет разработать план развития проекта, на основе которого будут создаваться новые страницы и контент;
- делает возможным планирование расходов на открытие площадки.

Виды логики структуры сайта.

Линейная

Логика такой структуры – ознакомить пользователей сайта со всеми его страницами, расположенными в определенной последовательности. Линейная схема применяется в сайтах-презентациях и портфолио. Из главной страницы как бы вытекают все остальные и расставляются цепочкой, звенья которой взаимосвязаны. Подобная структура не удобна для продвижения. Это обусловлено тем, что рекламировать можно только главную страницу (рисунок 1.2).

Рисунок 1.2 – Линейная структура сайта

Линейная с ответвлениями

Пример – сайт-портфолио фотомодели, работающей в нескольких разных стилях. Благодаря ответвлениям на одном сайте можно показывать сразу несколько продуктов. Переходя на ветку, пользователь будет видеть постраничную презентацию товара. Линейная структура с ответвлениями подразумевает, что у сайта будет одна главная страница, но несколько последних. Для продвижения схема также не удобна.

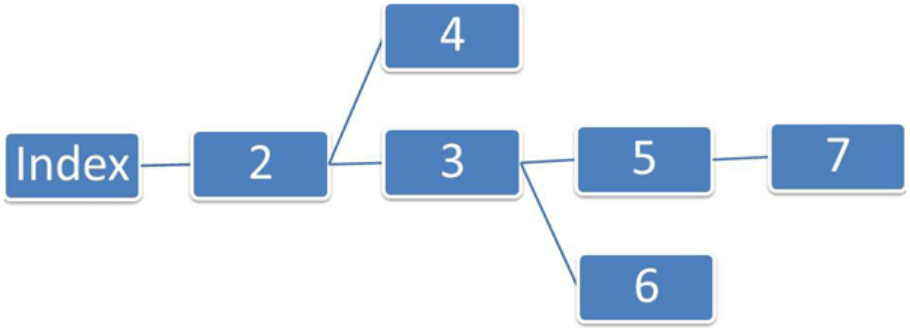


Рисунок 1.3 – Линейная структура с ответвлениями

Блочная

Подразумевает, что есть несколько равнозначных страниц, на которые ссылаются все остальные. Блочная структура сайта подходит дляпрезентации продукта: на страницах можно разместить описания отдельных свойств или характеристик товара (рисунок 1.4). Все страницы будут перелинкованы и связаны с главной, благодаря чему сайт будет легче продвигать. Однако блочная верстка достаточно специфична и подходит не всем видам ресурсов.

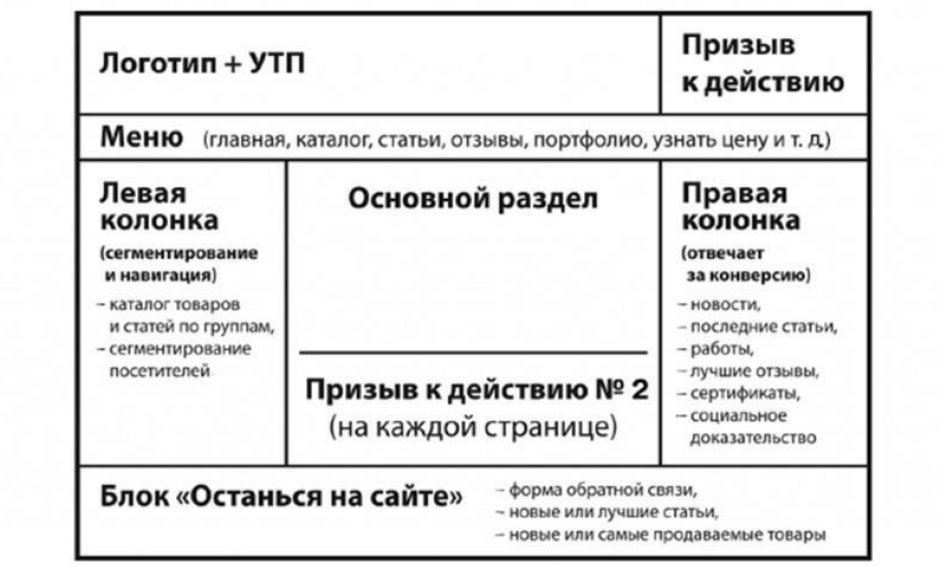


Рисунок 1.4 – Блочная структура

Древовидная

Наиболее универсальный вариант. В нем каждому товару или услуге отводится отдельная ветка: раздел или подраздел. Привычнее всего пользователям общаться именно с такими ресурсами. При древовидной структуре смысловая нагрузка делится между главной страницей и отдельными разделами, так как с ними линкуются отдельные страницы. Для продвижения это наиболее оптимальный вариант, позволяющий рекламировать сразу несколько разделов сайта.

Рисунок 1.5 – Древовидная структура сайта

Пример древовидной верстки в URL:

- name.ru/kresla/
- name.ru/kresla/tkani.html
- name.ru/kresla/kozha.html
- name.ru/krovati/

- name.ru/stylya/
- name.ru/stoly/derevo.html
- name.ru/stoly/plastic.html
- name.ru/stoly/rotang.html
- name.ru/stoly/metall.html

Из-за сильной разветвлённости древовидная структура сайта в виде схемы воспринимается проще и нагляднее.

Требования к структуре сайта от Яндекса и Гугла

Все онлайн-ресурсы затачиваются под поисковые системы, так как иначе пользователи их не увидят. Поэтому, говоря о требованиях к верстке, нельзя не вспомнить о Яндексе и Google.

Поисковые системы анализируют верстку ресурса по-своему. Их оценка сильно отличается от пользовательской, так как основывается на других принципах. Поисковые системы исследуют URL-структуру сайта. И уних есть определенные требования.

Требования Яндекс

- **Необходимо иметь четкую ссылочную структуру**. Каждая страница или документ должны относиться к своему разделу. На каждую страницу должна вести хотя бы одна ссылка с другой страницы.
- **Для ускорения индексации сайта нужна его xml-карта**.
- **С помощью файла robots.txt необходимо ограничивать индексирование служебной информации**.
- **У каждой страницы должен быть уникальный URL-адрес**. Разные страницы должны размещаться под разными адресами, а одна и та же страница должна иметь только один URL.
- **Ссылки на другие разделы необходимо делать текстовыми**, так Яндексу проще анализировать информацию.
- **Нужно проверять корректность symlink-ов**: когда пользователь переходит со страницы на страницу, адреса URL не должны суммироваться (пример от Яндекса, как быть не должно:

example.com/name/name/name/name/).

Рекомендации от Google

Этот поисковик лаконичен. В отличие от Яндекса его требования кверстке просты, понятны и занимают всего несколько строк:

- простая структура;
- понятная логика URL-адреса;
- слова, а не идентификаторы;
- присутствие знаков пунктуации в URL (особенно рекомендуется дефис «-»);
- короткие и простые URL.

4 варианта создания структуры сайта

После выбора структуры сайта следующий шаг – определиться, как он будет реализован на практике, то есть запланировать страницы, разделы,подразделы и т. п. Визуальное представление (шрифт, цвет кнопок, расположение меню и пр.) пока не обсуждаются.

Страницы, фигурирующие в структуре сайта

Структура во многом зависит от того, какие задачи должен решать ресурс. Выделяют четыре основных вида сайтов: визитка, коммерческий, информационный или блог, интернет-магазин. Остановимся на каждом подробнее.

1. **Сайт-визитка**

Обладает самой простой структурой. Как правило, состоит всего из двух уровней: главной страницы и остальных.

2. **Коммерческий сайт**

Такой ресурс должен решать более сложную бизнес-задачу, чем знакомство с компанией. Поэтому понадобится и более разветвлённая структура с дополнительными уровнями страниц. Например, основные страницы можно посвятить главным направлениям работы предприятия, а подразделы – более узким сферам деятельности. Такое решение оптимально

для компаний, оказывающих услуги, или любых других коммерческих организаций. Однако функция интернет-торговли здесь не предусмотрена.

3. Информационный портал и блог

Здесь многоуровневость создается с помощью разделов, которые состоят из отдельных страниц. На эти страницы невозможно попасть из главного меню. Это основное отличие информационного сайта от коммерческого, на все подстраницы которого можно перейти из меню. Исключение – страницы со статьями. Иногда доступ к ним делают не только из меню, но даже с главной страницы.

Автоматизация сайта

Улучшаем сервис для пользователей, сэкономив на административных расходах

Подробнее

С количеством разделов и подразделов нужно определиться до утверждения структуры сайта, то есть на этапе проработки. Если речь идет о личном блоге, нужно заранее продумать, каким темам будут посвящены публикуемые статьи и как читатели смогут найти их на сайте.

Если планируется зарабатывать на рекламе, то верстка ресурса должна основываться на семантическом ядре.

4. Интернет-магазин

Самый сложный по своей структуре вид сайтов. Чтобы учесть все поисковые запросы пользователей, потребуется создать не только многоуровневую систему разделов и подразделов, но и внедрить фильтры. И здесь важно не ошибиться и правильно решить, какие свойства реализуемых товаров пойдут в фильтры. Например, если магазин занимается продажей мягкой мебели, логично разделить ассортимент по качеству обивки (ткань, натуральная кожа, экокожа и т. п.), а не по цвету.

Хотя можно установить несколько фильтров, и тогда пользователи смогут группировать товары сразу по двум-трем и более свойствам. Ведь если магазин специализируется на продаже именно разноцветной мебели, то логично сделать фильтр по цветам. Разветвленная система фильтрации необходима, когда у товаров есть несколько основных характеристик, по которым люди их ищут в поисковых системах. Например, это может быть сочетание цвета и конструкции (диван синий угловой) или размера, цвета и материала (кровать двуспальная белая из дерева).

Фильтры необходимы, когда товары имеют несколько основных характеристик и варианты их сочетаний. Если же магазин специализируется на реализации конкретного продукта, например, одеял из верблюжьей шерсти, у которых меняется только размер, то система фильтрации не нужна.

У разделов каталога сайта могут быть как одинаковые фильтры, так и разные. Здесь все зависит от специфики реализуемой продукции.

Страницы, фигурирующие в структуре сайта

На сегодняшний день особой популярностью пользуются одностраничные сайты. Их структура проста, а создание несложное. Однако такой вид сайтов подходит не под все бизнес-задачи. Поэтому чаще всего встречаются многостраничные ресурсы.

Существует перечень базовых страниц. Не все они обязательно должны присутствовать в структуре сайта, однако поисковые системы обращают внимание на их наличие. Таким образом, включение базовых страниц в схему ресурса может облегчить продвижение по ключевым запросам.

К основным страницам относятся следующие:

К основным страницам относятся следующие:

- **Главная**

Название отражает суть этой страницы. С нее пользователи должны иметь возможность попасть в любой раздел и подраздел ресурса. Структура страницы сайта может быть очень сложной. В первую очередь это касается

главной страницы. Часто ее верстают в самый последний момент, когда известны основные разделы площадки и ее общая схема.

Следует иметь в виду, что сейчас присутствие слова «главная» не популярно. Оно занимает много места и часто не вписывается в дизайн.Вместо «главная» используется иконка в виде домика, название фирмы илиее логотип.

Контакты

Это также одна из самых важных страниц структуры сайта. От того, легко ли ее найти и как она оформлена, зависит эффективность коммуникации с пользователями. Если у компании несколько каналов связи, то, как правило, приоритетные размещаются в шапке сайта, а остальные в «Контактах».

Поисковые системы в первую очередь оценивают наличие этой страницы, поэтому важно, чтобы на ней была размещена максимально подробная информация о способах связи с компанией: городские телефоны, факс, электронную почту, фактический адрес с картой или схемой проезда, социальные сети, мессенджеры. Можно добавить реквизиты фирмы. Обязательно наличие микроразметки. Пример приведен на рисунке 1.6

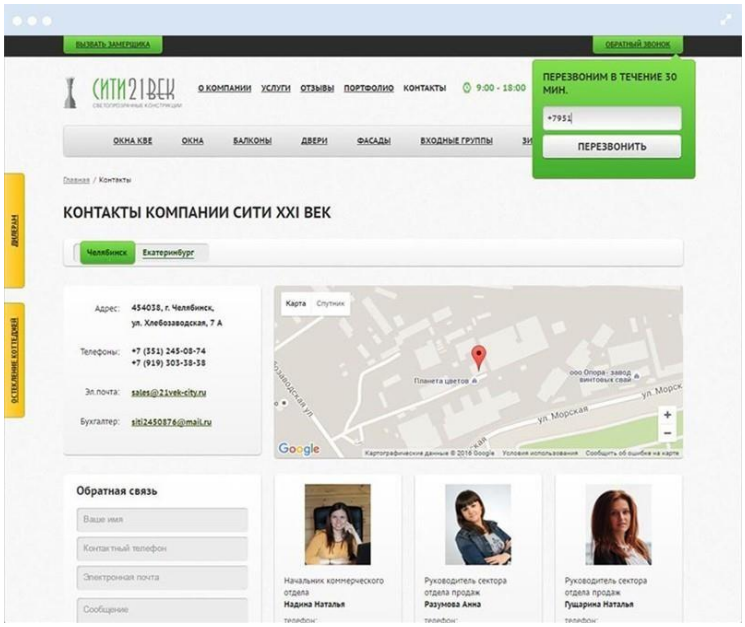


Рисунок 1.6 – Пример страницы Контакты

Если у компании несколько городских номеров или почтовых ящиков, необходимо обозначить, какой за что отвечает. Также на странице «Контакты» должен присутствовать распорядок работы фирмы и период, в который принимаются звонки.

Типичными ошибками являются скудная контактная информация, непривлекательный дизайн, отсутствие микроразметки и как следствие, стимулов связаться с компанией.

Доставка

Если площадка занимается реализацией товаров, она так или иначе связана с доставкой. Даже если компания сама не оказывает транспортные услуги, на сайте обязательно должна быть страница с подробным описанием того, как можно получить заказ. Например, можно разместить список рекомендованных транспортных компаний и номера их телефонов. Главное, чтобы клиент понимал, как он получит свой заказ. Это существенно увеличит вероятность совершения покупки.

Когда речь идет о доставке, обязательно нужно указывать, платная она или бесплатная, сроки, условия оказания услуги, время работы, способы транспортировки. Ссылки на пункты выдачи товаров или офисы транспортных компаний помогут пользователям сориентироваться в выборе способа доставки.

Оплата

Если компания занимается реализацией товаров, значит, на сайте предусмотрена система оплаты заказов. Ее детальное описание следует сделать доступным для клиентов. Еще до совершения покупки пользователи должны понимать, какими способами смогут оплатить заказ.

Вопрос ответ (FAQ)

Почему-то многие компании исключают этот раздел из структуры сайта, а ведь он очень важен. Как правило, люди задают примерно одинаковые вопросы. Если ответы на них разместить на сайте, это снимет часть нагрузки с телефонных операторов.

О компании

Многие фирмы на этой странице рассказывают о периоде становления и своих достижениях. Это ошибка. Эффективнее говорить о том, что компания может дать своим клиентам и подкреплять обещания дипломами, кейсами и примерами решенных проблем.

Отзывы (Обратная связь)

Если компания постоянно контактирует с клиентами, этот раздел необходим. Он должен быть легкодоступным и простым для заполнения. Не стоит делать раздел доступным только для зарегистрированных пользователей. Отзывы должны быть открыты для всех желающих их почитать.

- Многие фирмы стимулируют своих клиентов делиться комментариями, предлагая подарки и дополнительные скидки. Хорошо зарекомендовали себя видеоотзывы. Если есть возможность включить их в структуру сайта, это нужно сделать.

Типичные ошибки – размещение отзывов с одного IP, заказные комментарии с завышенной оценкой компании, анонимные отзывы, небольшое количество публикаций.

Для партнёров (Оптовым клиентам)

В структуре сайтов онлайн-магазинов часто есть раздел «Для оптовых покупателей». Он помогает разграничить аудитории и наладить общение с партнерами. Как правило, страница содержит описание условий оптовых закупок или размещения рекламы.

Новости (Блог, события, статьи)

Этот раздел помогает привлекать пользователей из поисковиков. Публикуемый контент должен не только содержать ключевые запросы, но и быть интересным и полезным для посетителей сайта. Размещать материал следует регулярно и постоянно. Если у компании нет новостей, то лучше исключить этот раздел из структуры ресурса.

Типичные ошибки – публикации ненужного материала (например, поздравлений с государственными праздниками), редкое размещение статей (раз в месяц), некачественный контент.

Услуги

Бывает так, что компания оказывает несколько услуг, но на сайте они объединены на одной странице. Такая подача сведений о деятельности фирмы неэффективна. Она не дает притока пользователей из поисковиков. Рекомендуется под каждый продукт делать отдельную страницу, на которой будет опубликована максимально подробная информация об оказываемой услуге, желательно с примерами работ, фото и видео.

Страница ошибки 404

Многие компании не уделяют этой странице внимания, из-за чего теряют часть посетителей. Страница с ошибкой 404 должна быть оформленав том же стиле, что и основной сайт. На ней следует разместить информацию для пользователей о том, почему они попали сюда и что делать дальше.

Посетителям можно предложить перейти на главную страницу сайта, на основные разделы ресурса или в карточку похожего товара. Обязательно должны быть ссылки на перечисленные страницы.

Типичные ошибки – отсутствие страницы 404, другой дизайн,отличимый от основного сайта, отсутствие информации для пользователей о том, что делать дальше.

Важность призыва к действию в структуре сайта

Какой бы ни была структура сайта, схема обязательно должна предусматривать призыв к действию. Как правило, он размещается в тексте несколько раз. Первый – сразу после специального предложения. Второй раз – в середине текста. Третий – в конце. Если текст длинный, призывов к действию может быть и больше. Они могут быть сформулированы одинаково или по-разному в зависимости от общего контекста статьи.

Иногда призывы делают двойными, например, «Купите товар и оформите подписку». Или добавляют предложение о выгоде: «Запишитесь на

консультацию. Первая бесплатно!». Или ограничивают срок действия предложения: «Купите новые шины. Только сегодня скидка 20 %».

Рисунок 1.7 – Пример оформления призыва

Если товар недорогой и его покупка не требует долгого обдумывания, призыв формулируется с помощью глаголов, обозначающих немедленное действие: купи, закажи, подпишись и т. п.

Когда товар дорогой или сложный или продажи осуществляются в B2B-сегменте, процесс принятия решения о покупке может занять долгое время. Призывы к немедленному действию здесь не сработают. Поэтому продавцы делают акцент на общение с клиентами и установление с ними крепкой эмоциональной связи. Потенциальных покупателей приглашают на бесплатные консультации, клиентские мероприятия, предоставляют пробную версию продукта и т. п.

Также в продажах дорогого или сложного товара для мотивации клиентов часто используются буклеты, электронные рассылки или ссылки. Они нужны тогда, когда потенциальный покупатель не готов общаться с менеджером по телефону или при личной встрече, но ему необходима информация о товаре, которая поможет принять окончательное решение.

Как правило, в конце посадочной страницы, помимо последнего призыва к действию, размещаются контакты компании, ссылка на форму оформления заказа, схема проезда, если у фирмы есть офлайн-офис.

Дополнительные опции в структуре сайта

В процессе управления структурой сайта можно добавлять различные элементы, которые способны повысить эффективность ресурса. К таким фишкам относятся:

- **Сценарии использования товара**

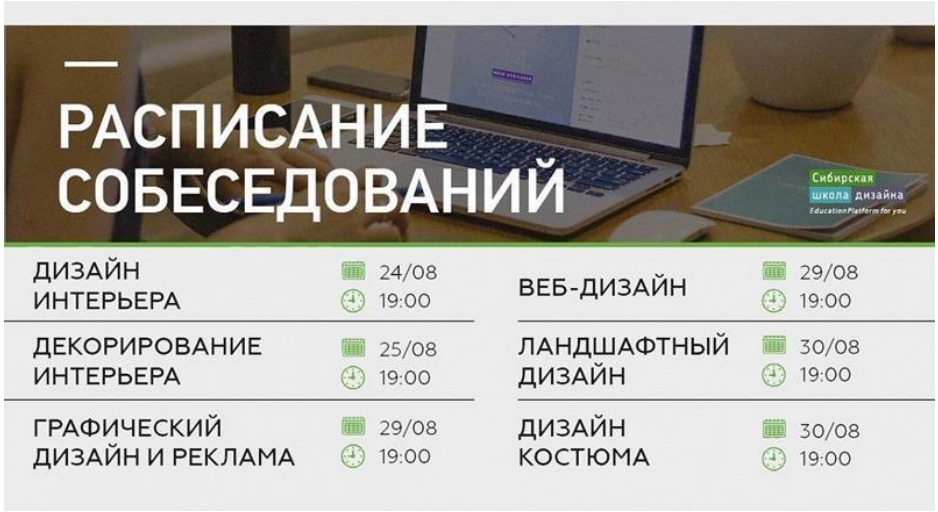
Примеры того, как можно использовать товар, могут положительно повлиять на его продажи. Форма подачи информации стандартная: заголовок, описание и фото или видеоролик.

- **Форма подписки**

Для многих сайтов одно из целевых действий – оформление подписки. Ведь если человек пока не готов купить, но согласен участвовать в рассылке, со временем он может стать постоянным клиентом. Как правило, форма имеет заголовок и содержит минимальное количество полей.

- **Расписание события**

Используется, если сайт посвящен мероприятию, протяжённому по времени и состоящему из отдельных этапов. Как правило, расписание оформляется как таблица или таймлайн. Если в событии принимает участие несколько спикеров, желательно разместить их ФИО и фото.



- **Счетчик обратного отсчета**

Его основное предназначение – показать, сколько времени осталось до окончания срока действия, например, специального предложения. Счетчики

мотивируют потенциальных потребителей быстрее принимать решение о покупке. Как правило, расположены в категории «Обложка».

Другой вариант использования счетчика – показать, сколько времени осталось до выхода товара в свет или старта продаж. Такой рекламный ход способен собрать целевую аудиторию еще на этапе изготовления продукта. Счетчик можно разместить на главной странице или на форме сбора подписчиков.

- **Видеоконтент**

С помощью видеоматериалов можно наглядно объяснить миссию проекта, показать, как использовать продукт, продемонстрировать новые достижения, истории успеха и собрать отзывы пользователей. Ролики следует размещать в галерее, в поп-апах, а также внутри статей и текстов.

- **Тезисный список преимуществ**

Как правило, блок «Наши преимущества» имеет три-четыре позиции с описанием и фото. Здесь же речь идет именно о тезисной подаче, то есть пользователям предлагается список из пяти и более пунктов. Примеры заголовков: «7 причин выбрать наш ресторан», «Почему модницы выбирают наши сумки» и т. п.

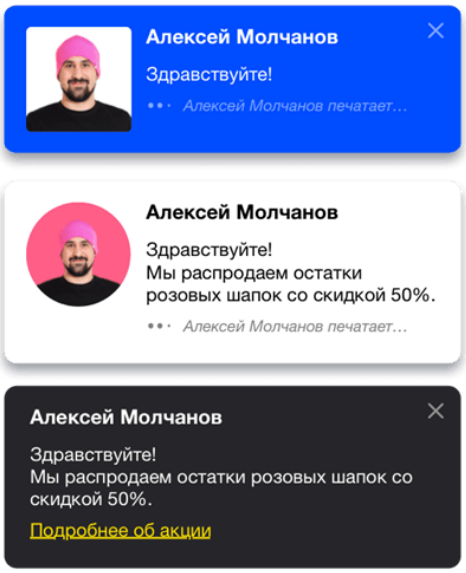
- **Истории успеха клиентов**

Ничто так не мотивирует к покупке, как истории успеха других людей. В публикациях должно рассказываться о том, как люди добились поставленной цели благодаря продаваемому на сайте продукту. Хорошо, если герои историй будут совершенно не похожи друг на друга, как и их достижения. Это наглядно продемонстрирует диапазон использования продукта. Для некоторых сегментов бизнеса актуальны будут фотографии из разряда «до и после».

- **Всплывающие окна**

Чаще всего в поп-апах размещается информация об акциях или предложение оформить подписку. Всплывающие окна могут появляться как

самостоятельно, так и в результате определенного действия посетителя, например, нажатия кнопки.

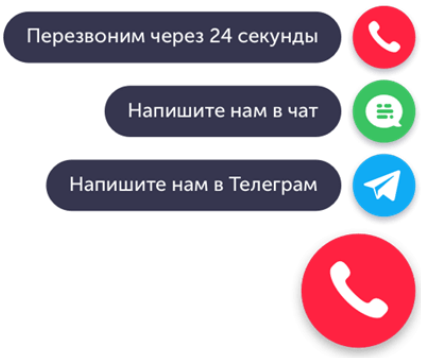


- **Факты роста в цифрах**

Цифры наглядны, легко воспринимаются и запоминаются, привлекают внимание. Оптимально разместить 3-4 факта, выраженных числами, с кратким описанием каждого.

- **Кнопки обратной связи**

От этих элементов напрямую зависит успешность коммуникации с аудиторией. Ведь чем проще человеку связаться с компанией, тем большим доверием он к ней проникнется. А от уровня лояльности целевой аудитории, как известно, напрямую зависит конверсия.



- **Меню**

Здесь есть два варианта. Первый – залипающее меню со ссылками к основным разделам сайта. Второй – боковая навигация с точками. Она не отвлекает внимание пользователей, но в то же время позволяет быстро попасть в нужное место.

На многих сайтах главная проблема с меню – его перегруженность. Этот раздел начинает больше отвлекать внимание и раздражать, чем помогать. Поэтому главное – не делать слишком большое количество полей и использовать только короткие слова.

- **Иконки социальных сетей**

Бывают двух видов: со ссылкой на профиль компании и на шаринг. Иконки могут располагаться в шапке главной страницы, в контактах, во всплывающих окнах.

Социальные сети – еще один канал установления эффективной коммуникации с целевой аудиторией. В первую очередь он ориентирован на тех пользователей, которые проводят много времени в онлайн-пространстве и для которых важно, насколько у компании много расшариваний, лайков и комментариев.

- **Мотивирующая фраза**

Эмоциональный, вдохновляющий призыв, размещенный на фоне атмосферного фото или видео, способен пробудить в человеке светлые чувства, натолкнуть на определенную мысль и просто надолго запомниться. Мотивирующие фразы делают сайт менее строгим и формальным. Также они помогают установить прочную эмоциональную связь с аудиторией, основанную на общих ценностях.

11 советов по разработке структуры сайта

Представление структуры сайта должно основываться не только на потребностях компании, но и на требованиях поисковых систем. Ниже описаны основные правила индексирования, по которым работают Яндекс и Google:

1. **Слеш в конце адреса** (пример: name.ru/gde-to-tut/) **говорит поисковому роботу о том, что необходима индексация глубже**. Однако если на этой странице нет внутренних ссылок, система может ее не проиндексировать. Поэтому слеш следует использовать аккуратно.

2. Не нужно страницам с дополнительными статьями давать название «Статьи». Поисковые роботы скорее всего их не проиндексируют. Лучше структурировать информацию и **каждой странице присваивать уникальное имя**.

3. **Закрывайте служебные страницы от поисковых систем**.

4. **В структуре сайта должна быть предусмотрена строка навигации**. Она подскажет пользователю, где он находится, и поможет вернуться на предыдущие уровни.

5. **С каждой страницы должен быть переход на главную**.

6. **URL должен быть понятным человеку и логичным**. Пример плохого адреса: name.ru/rt4566543tr/s45456/665665/345663/rtsrt/1t23. При создании ЧПУ (человеко понятных урлов) следует применять транслитерацию.

7. Если есть необходимость в четвертом и последующих уровнях, нужно **делать карту сайта и больше ссылок с «верхних» страниц**.

8. Правильный путь к страницам третьего уровня должен быть таким: **Главная → Документ 2 уровня → Документ 3 уровня**.

9. **Уровень вложенности не глубже трех**. Поисковые системы могут не проиндексировать более глубокие страницы.

10. **Внутренняя перелинковка должна быть грамотной**, так как она влияет на поведение пользователей и, соответственно, отношение поисковых роботов к страницам сайта.

11. **Такие разделы, как «Каталог», «Услуги», следует размещать в одном клике от главной**, желательно в ее верхней части. Они должны быть хорошо заметны и доступны для пользователей.

Методические указания к выполнению

1. Ознакомиться с теоретическим содержанием лабораторной работы.
2. Выбрать тип структуры сайта (обосновать свой выбор)
3. Определить, какие задачи будет решать сайт.
4. Определить страницы, фигурирующие в структуре сайта
5. Разработать графическую схему сайта
6. Разработать внешнюю и внутреннюю структуру сайта
7. Разработайте адреса страниц в соответствии со структурой сайта

Контрольные вопросы:

1. Что такое структура сайта?
2. Для чего необходима графическая схема структуры сатйа?
3. Что из себя представляет внешняя структура сайта?
4. Что из себя представляет внутренняя структура сайта?
5. Каковы требования к структуре сайта?
6. Какие преимущества дает формирование четкой структуры ресурса?
7. Дайте сравнительную характеристику различным структурам сайта
8. Какие требования к структуре сайта предъявляет Яндекс?
9. Какие требования к структуре сайта предъявляет Гугл? 10.Каков базовый перечень страниц сайта? Дайте им краткую

характеристику

10. Современные требования к странице Главная
11. Типичные ошибки страницы Контакты

Лабораторная работа № 2. Разработка архитектуры и файловой структуры веб-приложения

Цель занятия: освоить теоретические аспекты архитектуры веб-приложения. Разработать архитектуру веб-приложения

Теоретическое обоснование

1. Описание понятия архитектуры веб-приложения

Архитектура веб-приложения в основном представляет отношения и взаимодействия между такими компонентами, как пользовательские интерфейсы, мониторы обработки транзакций, базы данных и другие.

Основная цель разработки архитектуры веб-приложения – это слаженная и правильная работа всех элементов веб-приложения.

Логика довольно проста: когда пользователь вводит URL-адрес в браузере и нажимает «ввод», браузер делает запрос к серверу. Сервер отвечает, а затем показывает требуемую веб-страницу. Все эти компоненты создают архитектуру веб-приложения.

Все приложения состоят из двух частей - клиентской (front-end) и серверной (back-end).

Интерфейс - это визуальная часть приложения. Пользователи могут видеть интерфейс и взаимодействовать с ним. Клиентский код реагирует на действия пользователей. Серверная часть не визуальна для пользователей, но заставляет их запросы работать. Он обрабатывает бизнес-логику и отвечает на HTTP-запросы.

Например, когда пользователь вводит свои учетные данные в регистрационную форму, он имеет дело с внешним интерфейсом, но как только пользователь нажимает «ввод» и регистрируетесь - это серверная часть заставляет его работать.

При правильной работе клиентская и серверная стороны составляют архитектуру программного обеспечения веб-приложения.

Веб-приложения разделяет свои основные функции на уровни. Это позволяет заменять или обновлять каждый слой независимо.

Базовые компоненты архитектуры веб-приложений делятся на компоненты пользовательского интерфейса и структурные компоненты. Структурные компоненты делятся на клиентские и серверные.

Компоненты пользовательского интерфейса обозначают все элементы интерфейса, такие как журналы активности, информационные панели, уведомления, настройки и многое другое. Они являются частью макета интерфейса веб-приложения.

Структурные компоненты состоят из клиентской и серверной сторон: Клиентский компонент разработан с HTML, CSS или JavaScript. Веб- браузеры запускают код и преобразуют его в интерфейс, поэтому нет необходимости в настройке операционной системы.

Что касается серверного компонента, он построен на Java, .Net, Node.JS, Python и других языках программирования. Сервер состоит из двух частей - логики приложения и базы данных. Логика приложения - это центр управления веб-приложением. База данных отвечает за хранение информации (например, учетных данных пользователей).

2. Уровни архитектуры веб-приложений

Существует четыре общих уровня веб-приложений:

- Уровень представления (PL)
- Уровень обслуживания данных (DSL)
- Уровень бизнес-логики (BLL)
- Уровень доступа к данным (DAL)

Уровень представления PL отображает пользовательский интерфейс и упрощает взаимодействие с пользователем. Уровень представления имеет компоненты пользовательского интерфейса, которые визуализируют и показывают данные для пользователей. Также существуют компоненты пользовательского процесса, которые задают взаимодействие с пользователем. PL предоставляет всю необходимую информацию клиентской стороне. Основная цель уровня представления - получить входные данные,

обработать запросы пользователей, отправить их в службу данных и показать результаты.

Слой бизнес-логики BLL несет ответственность за надлежащий обмен данными. Этот уровень определяет логику бизнес-операций и правил. Вход на сайт - это пример уровня бизнес-логики.

Уровень службы данных DSL передает данные, обработанные уровнем бизнес-логики, на уровень представления. Этот уровень гарантирует безопасность данных, изолируя бизнес-логику со стороны клиента.

Уровень доступа к данным DAL предлагает упрощенный доступ к данным, хранящимся в постоянных хранилищах, таких как двоичные файлы и файлы XML. Уровень доступа к данным также управляет операциями CRUD - создание, чтение, обновление, удаление.

3. Типы архитектуры веб-приложений

Можно выделить несколько типов архитектуры веб-приложений, в зависимости от того, как логика приложения распределяется между клиентской и серверной сторонами. Наиболее распространенные архитектуры веб-приложений:

1. Одностраничные веб-приложения
2. Многостраничные веб-приложения
3. Архитектура микросервисов
4. Бессерверная архитектура
5. Прогрессивные веб-приложения

Разберемся в деталях каждого вида.

1. Одностраничное приложение или SPA - это веб-сайт или веб-приложение, которое загружает всю необходимую информацию при входе на страницу. Одностраничные приложения имеют одно существенное преимущество - они обеспечивают потрясающий пользовательский интерфейс, поскольку пользователи не испытывают перезагрузки веб-страниц. Одностраничные веб-приложения часто разрабатываются с

использованием фреймворков JavaScript, таких как Angular, React и других. Известные СПА : Gmail, Facebook, Twitter, Slack.

2. Многостраничное приложение или MPA. Многостраничные приложения более популярны в Интернете, так как в прошлом все веб-сайты были MPA. В наши дни компании выбирают MPA, если их веб-сайт довольно большой (например, eBay). Такие решения перезагружают веб-страницу для загрузки или отправки информации с / на сервер через браузеры пользователей. Известные MPA: eBay, Amazon.

3. Архитектура микросервисов. Чтобы понять архитектуру микросервисов, лучше сравнить ее с монолитной моделью. Традиционная монолитная архитектура веб-приложения состоит из трех частей - базы данных, клиентской и серверной сторон. Это означает, что внутренняя и внешняя логика, как и другие фоновые задачи, генерируются в одной кодовой базе. Чтобы изменить или обновить компонент приложения, разработчики программного обеспечения должны переписать все приложение. Что касается микросервисов, этот подход позволяет разработчикам создавать веб-приложение из набора небольших сервисов. Разработчики создают и развертывают каждый компонент отдельно. Архитектура микросервисов выгодна для больших и сложных проектов, поскольку каждый сервис может быть изменен без ущерба для других блоков. Поэтому, если необходимо обновить логику какого-либо бизнес-процесса, то микросервисная архитектура позволит на время не останавливать работу сайта. Известные проекты: Netflix, Uber, Spotify, PayPal.

Типы архитектуры веб-приложений

- Монолитные и микросервисы
- Бессерверная архитектура

Этот тип архитектуры веб-приложений заставляет разработчиков использовать облачную инфраструктуру сторонних поставщиков услуг,таких как Amazon и Microsoft.

Чтобы сохранить веб-приложение в Интернете, разработчики должны управлять серверной инфраструктурой (виртуальной или физической), операционной системой и другими процессами хостинга, связанными с сервером. Поставщики облачных услуг, такие как Amazon или Microsoft, предлагают виртуальные серверы, которые динамически управляют распределением машинных ресурсов. Другими словами, если вашеприложение испытывает огромный всплеск трафика, к которому ваши серверы не готовы, приложение не будет отключено.

Прогрессивные веб-приложения или PWA

Одна из основных тенденций в разработке веб-приложений последних лет - это прогрессивные веб-приложения. Это веб-решения, которые работают как собственные приложения на мобильных устройствах. PWA предлагают push-уведомления, автономный доступ и возможность установить приложение на домашний экран.

Для создания PWA разработчики используют «языки веб-программирования», такие как HTML, CSS и JavaScript. Если приложению требуется доступ к функциям устройств, разработчики используют дополнительные API - NFC API, API геолокации, Bluetooth API и другие.

Известные PWA: Uber, Starbucks, Pinterest.

4. Разработка архитектуры для веб-приложения

Качественная архитектура веб-приложения делает процесс разработки более эффективным и простым. Веб-приложение с продуманной архитектурой легче масштабировать, изменять, тестировать и отлаживать.

Есть несколько общих критериев для хорошо построенной архитектуры веб-приложения:

1. Эффективность
2. Гибкость
3. Расширяемость
4. Соблюдение принципа открыто-закрыто

5. Масштабируемость процесса разработки
6. Легко проверить
7. Возможность повторного использования
8. Хорошо структурированный и читаемый код

5. Архитектурные шаблоны

Архитектурный шаблон - это общее и повторяющееся решение часто возникающей проблемы архитектуры приложений в пределах заданного контекста. Архитектурные шаблоны схожи с шаблонами программного дизайна, однако имеют более широкий охват.

Рассмотрим наиболее популярные архитектурные шаблоны.

1. Многоуровневый шаблон

Данный шаблон используется для структурирования программ, которые можно разложить на группы неких подзадач, находящихся на определенных уровнях абстракции. Каждый слой предоставляет службы для следующего, более высокого слоя.

Чаще всего в общих информационных системах встречаются следующие 4 слоя:

1. Слой представления (также известен как слой пользовательского интерфейса)
2. Слой приложения (также известен как слой сервиса)
3. Слой бизнес-логики (также известен как уровень предметной области)
4. Слой доступа к данным (также известен как уровень хранения данных)

Использование:

- Общие десктопные приложения.
- Веб-приложения электронной коммерции (e-commerce).

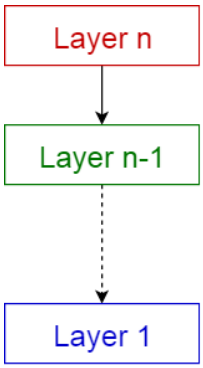


Рисунок 2.1 - Многоуровневый шаблон

2. Клиент-серверный шаблон

Данный шаблон состоит из двух частей: **сервера** и множества **клиентов**. Серверный компонент предоставляет службы клиентским компонентам. Клиенты запрашивают услуги у сервера, а он, в свою очередь, оказывает эти самые услуги клиентам. Более того, сервер продолжает «подслушивать» клиентские запросы.

Используется в основном в онлайн приложения (электронная почта, совместный доступ к документам, банковские услуги).

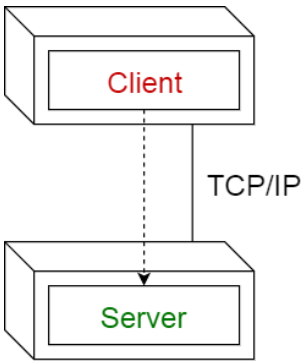


Рисунок 2.2 - Шаблон «Клиент-сервер»

3. Ведущий-ведомый

В этом шаблоне также задействованы два участника — **ведущий** и **ведомые**. Ведущий компонент распределяет задачи среди идентичных ведомых компонентов и вычисляет итоговый результат на основании результатов, полученных от своих «подчиненных».

Использование:

- В репликации баз данных. Главная БД считается авторитетным источником, а подчиненные базы с ней синхронизируются.
- Периферийные устройства, подключенные к шине в компьютере (ведущие и ведомые устройства).

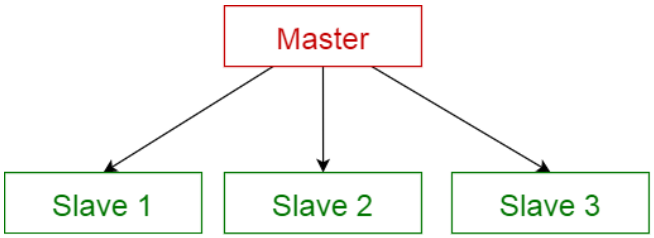


Рисунок 2.3 - Шаблон «Ведущий-ведомый»

4. Каналы и фильтры

Этот шаблон подходит для систем, которые производят и обрабатывают потоки данных. Каждый этап обработки происходит внутри некоего компонента **фильтра**. Данные для обработки передаются через **каналы**. Эти каналы можно использовать для буферизации или синхронизации данных.

Использование:

- Компиляторы. Последовательные фильтры выполняют лексический, синтаксический, семантический анализ и создание кода.
- Рабочие процессы в биоинформатике.

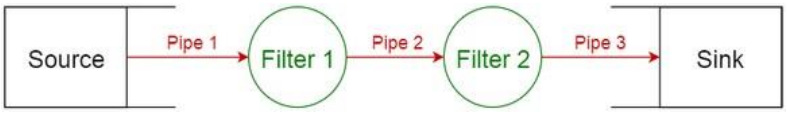


Рисунок 2.4 - Шаблон «Каналы и фильтры»

5. Шаблон посредника

Данный шаблон нужен для структуризации распределенных систем с несвязными компонентами. Эти компоненты могут взаимодействовать друг с другом через удаленный вызов службы. Компонент **посредник** отвечает за координацию взаимодействия **компонентов**.

Сервер размещает свои возможности (службы и характеристики) у посредника (брокера). Клиент запрашивает услугу у брокера. Затем брокер перенаправляет клиента к подходящей службе из своего реестра.

Использование:

- Брокеры сообщений по типу Apache ActiveMQ, Apache Kafka, RabbitMQ и JBoss Messaging.

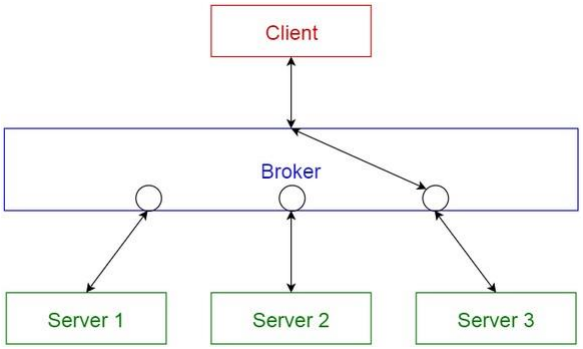


Рисунок 2.5 - Шаблон «Посредник»

6. Одноранговый шаблон

В данном шаблоне существуют отдельные компоненты, так называемые **пиры**. Пиры могут выступать в роли как **клиента**, запрашивающего услуги от других равноправных участников (пиров), так и **сервера,** предоставляющего услуги другим пирам. Пир может быть клиентомили сервером, или всем сразу, а также способен со временем динамически изменять свою роль.

Использование:

- Файлообменные сети (Gnutella и G2)
- Мультимедийные протоколы (P2PTV и PDTP)
- Проприетарные мультимедийные приложения (как тот же Spotify).

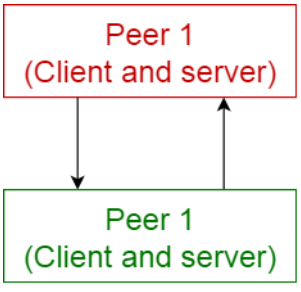


Рисунок 2.6 - Одноранговый шаблон

7. Шина событий

Этот шаблон, в основном, взаимодействует с событиями и состоит из 4 главных компонентов: **источник события, прослушиватель события, канал** и **шина событий.** Источники размещают сообщения для определенных каналов на шине событий. Прослушиватели подписываются на определенные каналы. Прослушиватели получают уведомления о появлении сообщений, размещенных на каналах из их подписки.

Использование:

- Разработки на Android
- Сервисы уведомлений

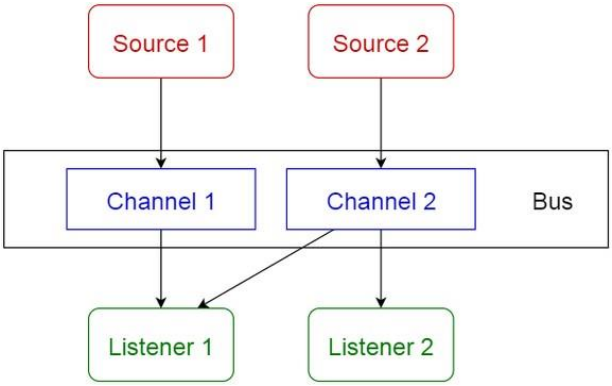


Рисунок 2.7 - Шаблон «Шина событий»

8. Модель-представление-контроллер

Этот шаблон также известен как MVC-шаблон. Он разделяет интерактивные прикладные программы на 3 части:

1. **Модель** - содержит ключевые данные и функционал;
2. **Представление** - показывает информацию пользователю (можно задавать более одного представления);
3. **Контроллер** - занимается обработкой данных от пользователя.

Это делается с целью разграничения внутреннего представления информации от способов ее представления и принятия от пользователя. Данная схема изолирует компоненты и позволяет эффективно реализовать повторное использование кода.

Использование:

- Архитектура WWW-приложений, написанных на основных языках программирования.
- Веб-фреймворки (например, Django и Rails)

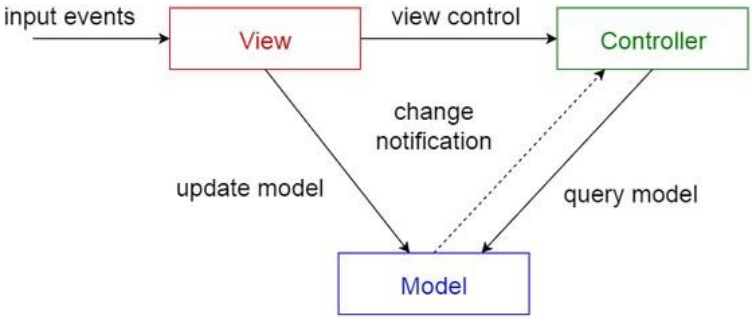


Рисунок 2.8 - Шаблон «Модель-представление-контроллер»

9. Доска

Такой шаблон подходит для проблем, для которых отсутствуют четкие детерминированные решения. Шаблон «Доска» состоит из 3 главных компонентов:

1. **Доска** - это структурированная глобальная память, содержащая объекты из пространства возможных решений;
2. **Источник знания** - специализированные модули со своим собственным представлением;
3. **Компоненты управления** - выбирает, настраивает и исполняет модули.

Все компоненты имеют доступ к доске. Компоненты могут производить новые объекты данных, которые добавляются к доске. Компоненты ищут на доске конкретные виды данных. Одним из способов поиска является сопоставление шаблонов с существующим источником знаний.

Использование:

- распознавание речи;
- идентификация и отслеживание транспортных средств;
- определение структур белка;
- интерпретация сигналов Sonar.

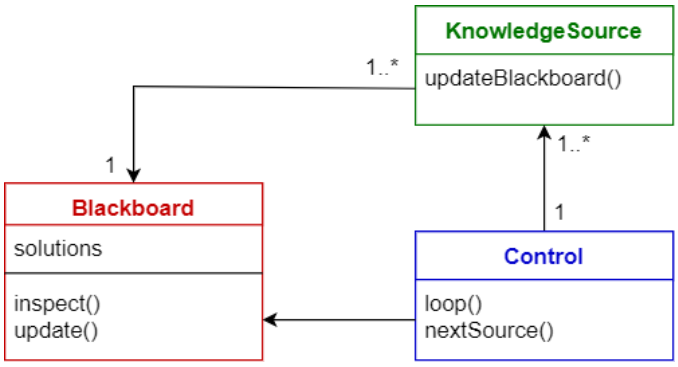


Рисунок 2.9 - Шаблон «Доска»

10. Интерпретатор

Он подходит для разработки компонента, который должен интерпретировать программы, написанные на специальном языке программирования. В основном, там расписано, как вычислять строки (иначе говоря: «предложения» или «выражения»), написанные на каком-то определенном языке программирования. Суть в том, чтобы присвоить класс каждому символу языка.

Использование:

- языки запросов к базе данных (SQL);
- языки, которые используются для описания протоколов передачи данных.

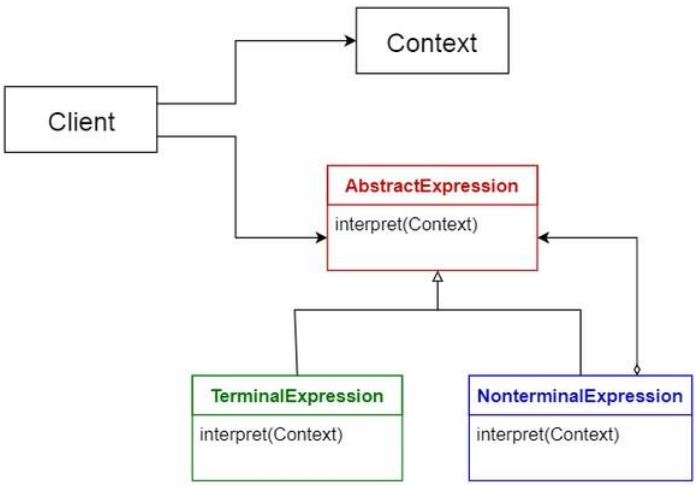


Рисунок 2.10 - Шаблон «Интерпретатор»

Сравнение архитектурных шаблонов

Ниже приводятся плюсы и минусы каждого из архитектурных шаблонов.

Наименование шаблона	Плюсы	Минусы
Многоуровневый шаблон	Одним низким слоем могут пользоваться разные слои более высокого ранга; Слои упрощают стандартизацию, т.к. мы четко определяем уровни; Изменения вносятся внутри какого-то одного слоя, при этом остальные слои остаются неизменными.	Не универсален; В ряде ситуаций возможен пропуск некоторых слоев.
Клиент-серверный шаблон	Подходит для моделирования набор служб, которые смогут запрашивать клиенты.	Запросы обычно выполняются в отдельных потоках на сервере. Взаимодействие между процессами повышает ресурсозатратность, т.к. разные клиенты имеют разное представление.
Шаблон «Ведущий-ведомый»	Точность, т.к. выполнение службы делегируется разным ведомым с разной реализацией.	Все ведомые изолированы, у них отсутствует общее состояние. Период ожидания в коммуникации «ведущий-ведомый» — это значительный минус. Например, в системах реального времени. Подходит только для тех проблем, решение которых можно разложить на части.

Шаблон «Каналы и фильтры»	Могут реализовывать параллельные процессы, когда вход и выход состоит из потоков,а фильтры начинают вычисления после получения данных. Простое добавление фильтров. Систему можно легко расширить. Фильтры подходят для повторного использования. Могут выстраивать различные конвейеры, создавая всевозможные комбинации существующего набора фильтров.	Эффективность снижается из-за самых медленных процессов фильтрации. При переходе от одного фильтра к другому выполняется трансформация данных, которая ведет к повышенному потреблению ресурсов
Шаблон «Посредник»	Возможно динамическое изменение, добавление, удаление и перемещение объектов. Этот шаблон делает процесс распределения прозрачным для разработчика.	Необходима стандартизация описаний служб.
Одноранговый шаблон	Поддерживает децентрализованные вычисления. Крайне устойчив к сбоям в любом узле. Высокая масштабируемость по части ресурсной и вычислительной мощности.	Отсутствует гарантия качества служб, т.к. узлы кооперируются стихийно. Трудно гарантировать защищенность. Производительность зависит от количества узлов.
Шаблон «Шина событий»	Простое добавление новых подписчиков, издателей и связей. Хорошо зарекомендовал себя для сильно распределенных приложений.	Проблема с масштабируемостью, т.к. все сообщения проходят через одну шину событий.
Шаблон «Модель-представление-контроллер»	Упрощает создание различных представлений одной и той же модели; их можно включить или	Возрастает сложность алгоритма. Может привести ко многим ненужным

	отключить на этапе выполнения	корректировкам действий пользователей.
Шаблон «Доска»	Легкое добавление новых приложений. Можно без труда расширять структуры пространства данных.	Редактировать структуры данных действительно трудно, т.к. такие изменения затрагивают все приложения. Могут потребоваться синхронизация и управление доступом.
Шаблон «Интерпретатор»	Возможно высокодинамичное поведение. Отличное решение для конечных пользователей с точки зрения удобства программирования.	Проблемы с производительностью, т.к. интерпретированный язык медленнее скомпилированного.

Методические указания к выполнению

Задание 1. Познакомиться с теоретической частью лабораторнойработой.

Задание 2. Разработать архитектуру веб-проекта индивидуального варианта. В отчете обосновать выбор архитектуры своего проекта.

Задание 3. Представить архитектуру с функциональными связями между компонентами в виде схемы с использованием любых графических средств. Пример на рисунке 2.11

Задание 4. Описать и дать характеристику каждого компонента архитектуры

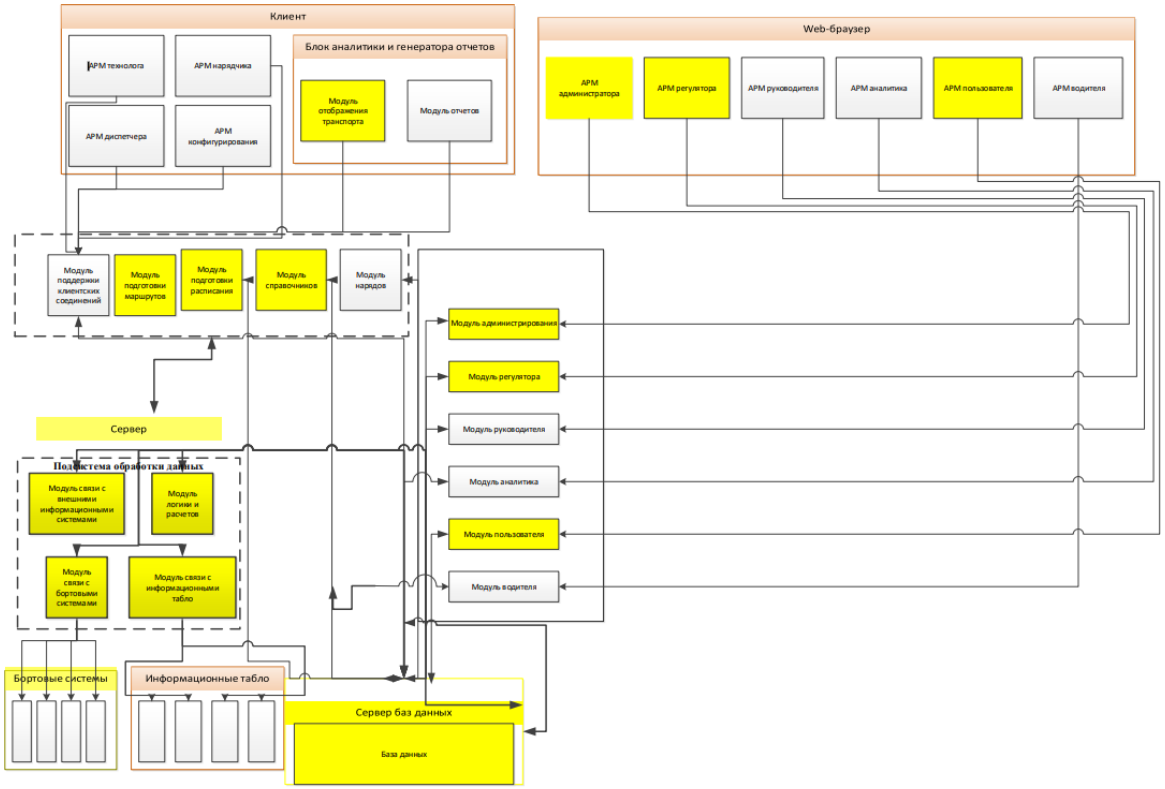


Рисунок 2.11 – Пример архитектуры информационной системы

Лабораторная работа № 3. Создание первого приложения. Разработка макета веб-приложения электронной коммерции

Цель и содержание работы: знакомство с ASP.NET MVC5, разработка макетов страниц веб-приложения.

Реализуемые компетенции: ПК-2, ПК-3

Теоретическое обоснование

Паттерн (англ. design pattern) (шаблон проектирования) в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Model-View-Controller (с, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») – схема разделения 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 (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

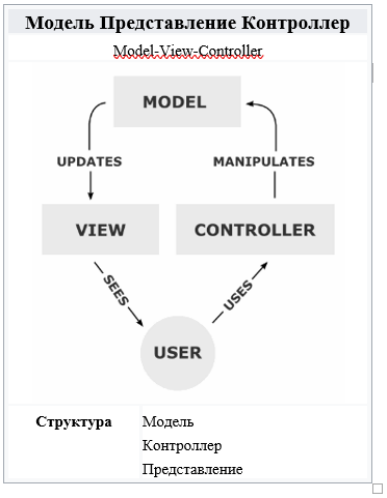


Рисунок 1 – Архитектурный паттерн MVC

Структура проекта MVC 5

App_Data – содержит файлы, ресурсы и базы данных, используемые приложением

App_Start – хранит ряд статических файлов, которые содержат логику инициализации приложения при запуске

Content – содержит вспомогательные файлы, которые не включают код на C# или JavaScript, и которые развертываются вместе с приложением, например, файлы стилей css

Controllers – содежит файлы классов контроллеров. По умолчанию в эту папку добавляются два контроллера - HomeController и AccountController

Fonts – хранит дополнительные файлы шрифтов, используемых приложением

Models – содержит файлы моделей. По умолчанию Visual Studio добавляет пару моделей, описывающих учетную запись и служащих для аутентификации пользователя

Scripts – каталог со скриптами и библиотеками на языке javascript

Views – здесь хранятся представления. Все представления группируются по папкам, каждая из которых соответствует одному контроллеру. После обработки запроса контроллер отправляет одно из этих представлений клиенту. Также здесь имеется каталог Shared, который содержит общие для всех представления

Global.asax – файл, запускающийся при старте приложения и выполняющий начальную инициализацию. Как правило, здесь срабатывают методы классов, определенных в папке App_Start

Startup.cs – поскольку в приложении MVC 5 используются библиотеки, применяющие спецификацию OWIN, то данный файл организует связь между OWIN и приложением. (OWIN или Open Web Interface for .NET представляет собой спецификацию, определяющую взаимодействие между веб-приложением и веб-сервером.)

Web.config – файл конфигурации приложения. **Web.config –** это конфигурационный файл, в котором хранятся все настройки и отдельные параметры веб-приложения.

Рассмотрим подробнее, из чего состоит этот файл **Web.config**:

configSection. Это секция отвечает за то, какие классы будут обрабатывать далее объявленные секции. Состоит из атрибута name — это тег, далее объявленной секции, и type – к какому классу относится.

connectionStrings. Это секция отвечает за работу с указанием строк инициализаций соединений с базами данных.

appSettings. Секция параметров типа key/value.

system.web, system.webServer. Секции параметров для работы веб-приложения.

runtime. Секция по настройке в режиме выполнения. Определение зависимостей между dll.

Остальные секции. Другие секции с параметрами, объявленными в **configSection**.

Конкретная структура каждого отдельного приложения, естественно, будет отличаться, а гибкость MVC позволяет изменять структуру, приспосабливая, ее к своим потребностям. Но описанные выше моменты будут общими для большинства проектов.

Технологии, используемые в лабораторной работе

Entity Framework (EF) — объектно-ориентированная технология доступа к данным, является object-relational mapping (ORM) решением для .NET Framework от Microsoft.

NuGet — система управления пакетами для платформ разработки Microsoft, в первую очередь библиотек .NET Framework. Управляется .NET Foundation.

Методические указания

1. Запускаем среду разработки Visual Studio исоздаем новый проект.

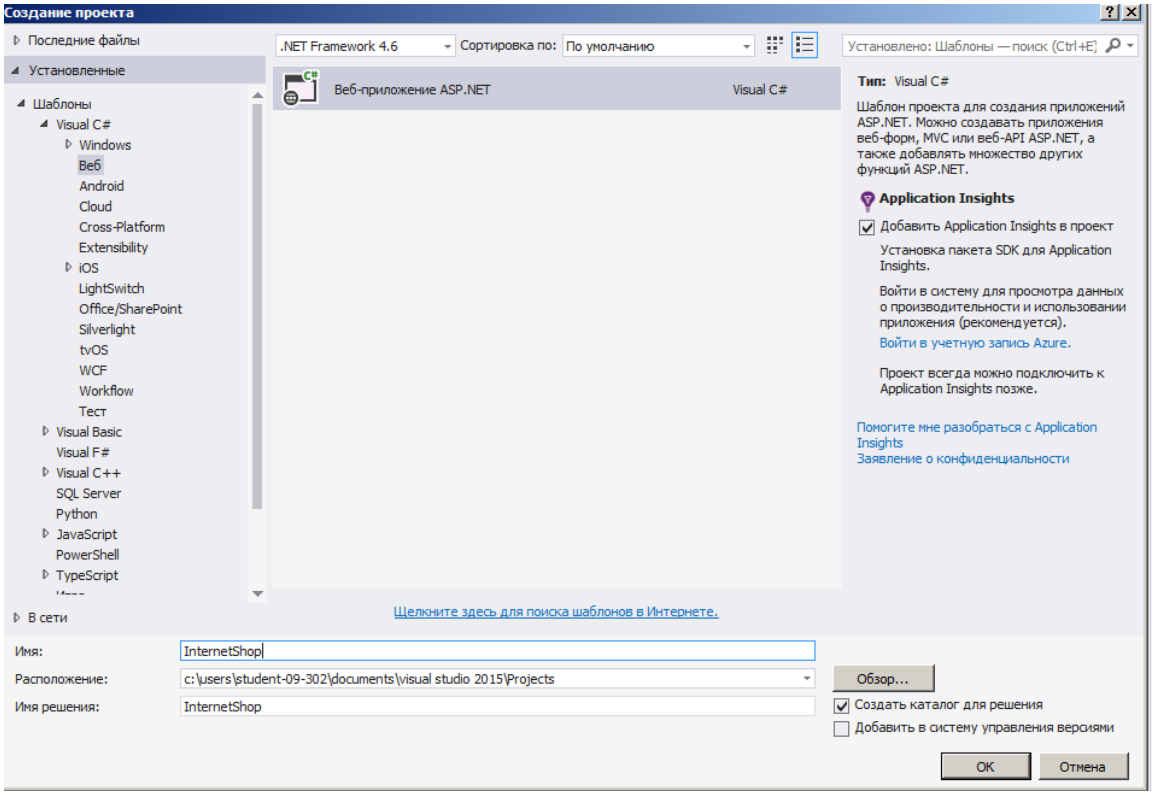


Рисунок 1 – Выбор ASP.NET

2. В левой части окна «Создание проекта» выбираем в списке язык C# и во вложенном списке выбираем «Веб» (рисунок 1).

3. В средней части окна выбираем шаблон «Веб-приложение ASP.NET в поле Имя вводим название будущего проекта, например, InternetShop и путь его расположения как показано на рисунке 1. ОК.

4. Выбираем шаблон MVC, как показано на рисунке 2. В разделе «Добавить папки и основные ссылки для:» окна Выбора шаблона установить флажок MVC. После чего создастся проект нового вашего приложения (рисунок 3).

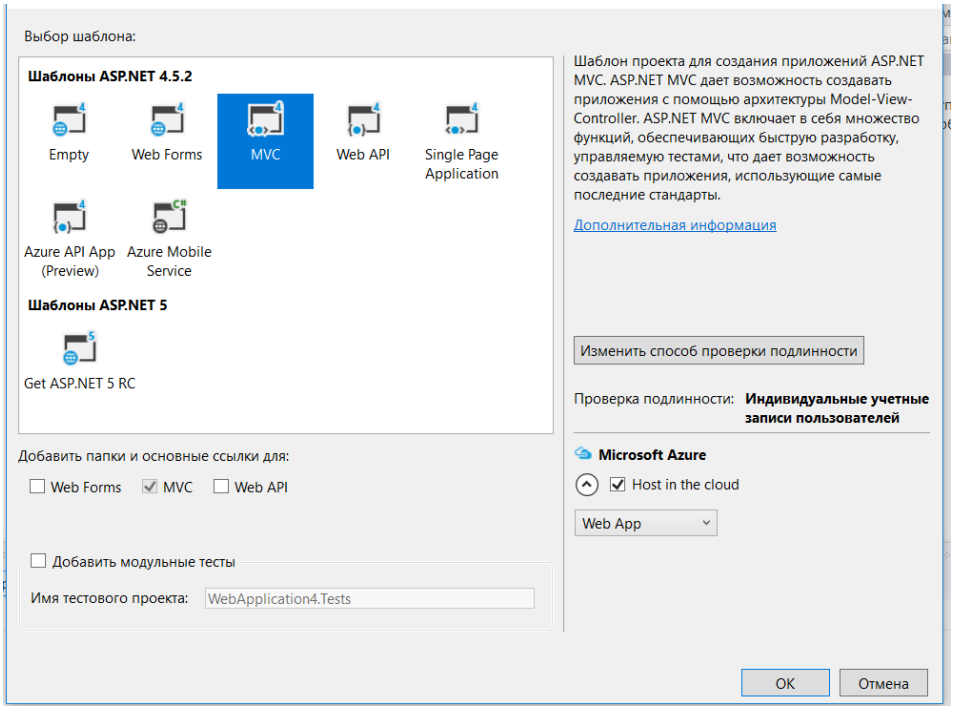


Рисунок 2 – Окно выбора шаблона

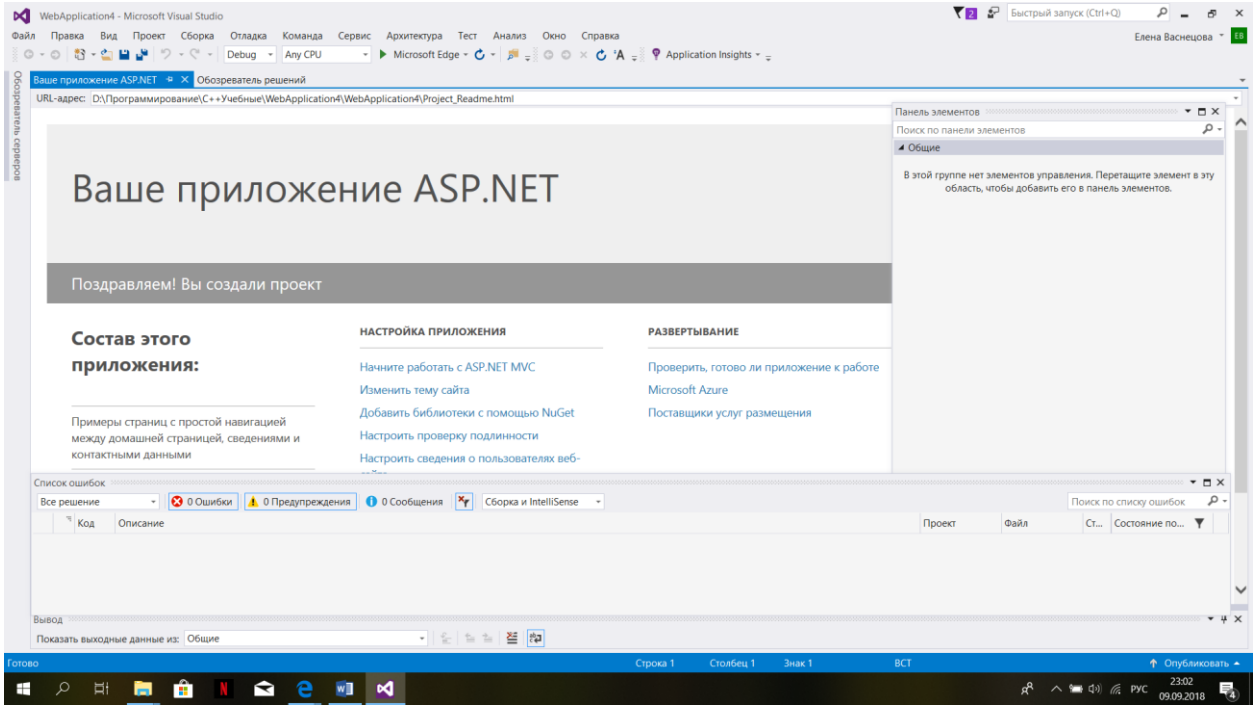


Рисунок 3 – Первая страница нового приложения

Таким образом, создан проект, который практически не обладает никакой функциональностью, хотя уже имеет базовую структуру (рисунок 4), которую вы можете просмотреть через Обозреватель решений (рисунок 5). Назначение папок структуры проекта описаны в теоретической части данной лабораторной работы.

После создания проекта необходимо определить модели данных, разрабатываемого приложения. Поскольку речь идет об интернет магазине компьютерной техники, то такими моделями могут быть модель компьютеры и модель покупки компьютерной техники.

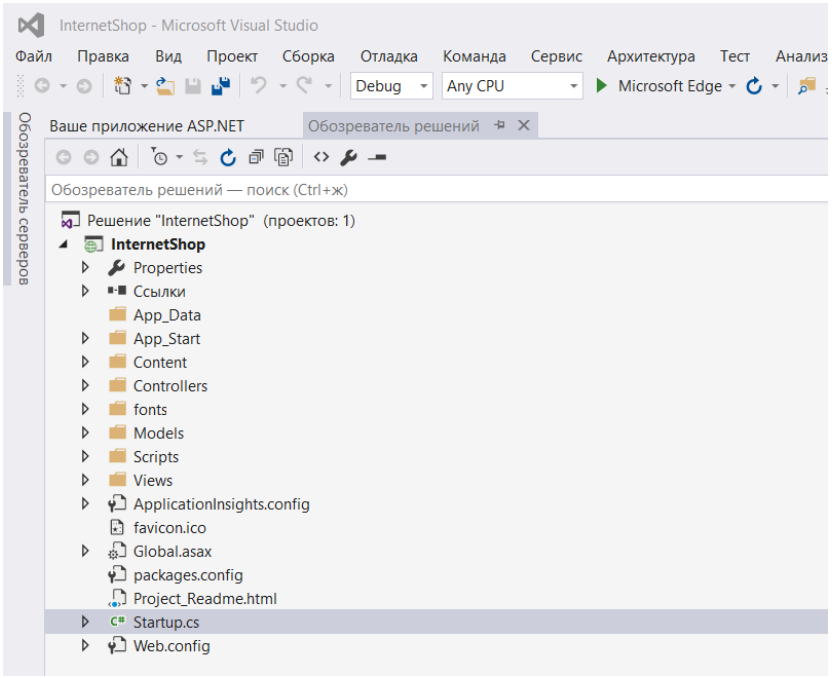


Рисунок 5 – Обозреватель решений (структура проекта)

5. В проекте уже по умолчанию определена папка Models. В ней будут находиться все модели. Нажмем на эту папку правой кнопкой мыши и в появившемся меню выберем Добавить (Add) /Класс (Class).... (рисунок 6 и 7). Назовем первый новый класс Computer и добавим в него код, описывающий модель Computers

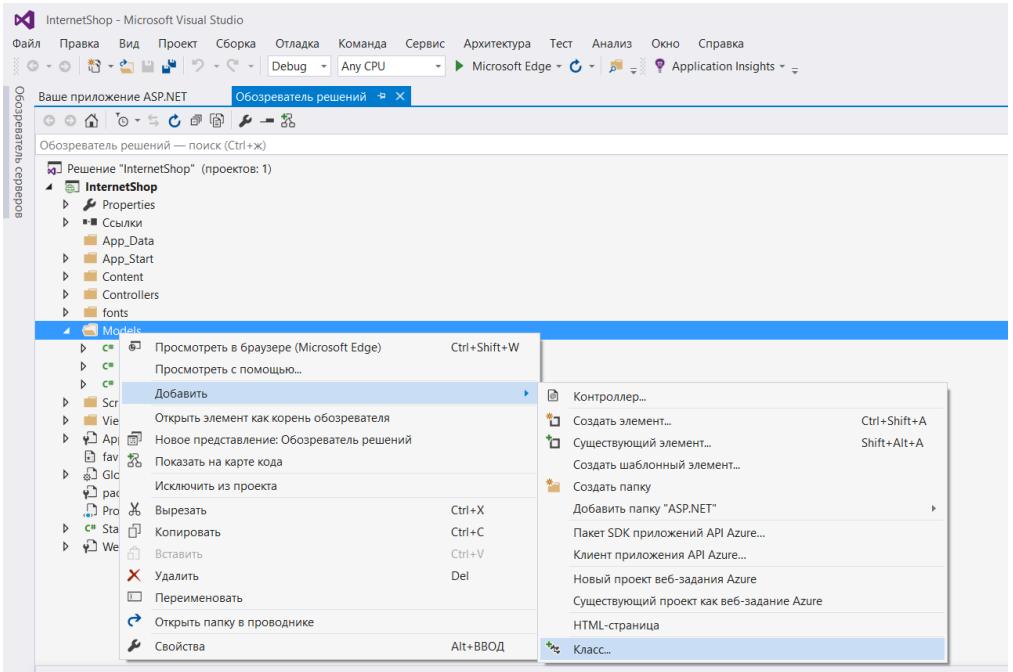


Рисунок 6 – Добавление класса

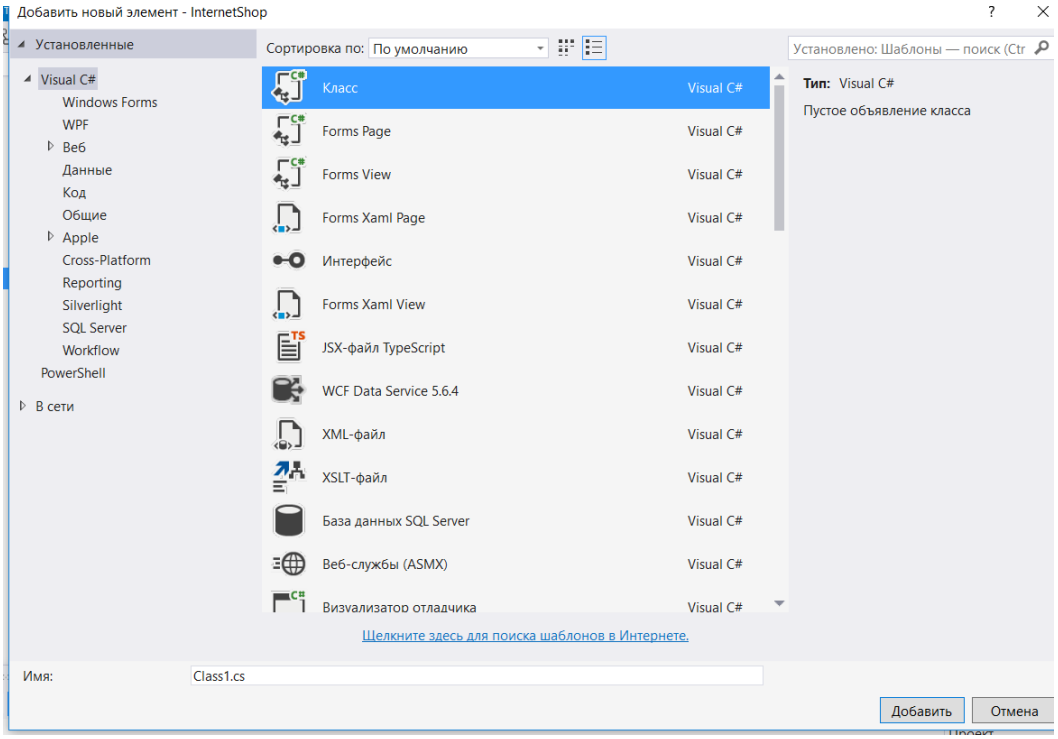


Рисунок 7 – Выбор нового элемента Класс

6. Запишем следующий программный код на C#, определяющий модель Computer.

```
namespace InternetShop.Models
{
    public class Computer
    {// ID компьютера (первичный ключ)
        public int Id { get; set; }
        // название компьютера
        public string Name { get; set; }
        // страна
        public string Country { get; set; }
        // цена
        public int Price { get; set; }
    }
}
```

Конечно, в реальном приложении подобная модель должна включать изображение компьютера, количество штук на складе, данные процессора и т.д. Но для учебной задачи ограничимся небольшим набором полей.

7. Подобным образом второй класс - модель Purchase, которая будет отвечать за отдельную совершаемую покупку компьютера.

```
public class Purchase

{
        // ID покупки
```

```
        public int PurchaseId { get; set; }
        // имя и фамилия покупателя
        public string Person { get; set; }
        // адрес покупателя
        public string Address { get; set; }
        // ID компьютера
        public int ComputerId { get; set; }
        // дата покупки
        public DateTime Date { get; set; }
}
```

8. Модель представляет обычный класс на языке C#, поэтому все модели имеют набор свойств, описывающие реальные свойства объекта. Поскольку для хранения моделей будет использоваться база данных, например, SQL Server, то для манипуляции над объектами в базе данных нам надо определить для них первичный ключ (Primary Key), который выполняет роль универсального идентификатора объекта. Поэтому первым свойством в каждой модели должно быть свойство Id, предназначенное для хранения первичного ключа.

При этом необходимо соблюдать следующие требования:

- свойство идентификатора модели должна иметь имя либо Имя_моделиId, либо просто Id. Так, у нас в модели Computer определено свойство Id, то есть данное свойство является первичным ключом. А в случае с моделью Purchase свойство носит название PurchaseId.

9. Для работы с данными в ASP.NET MVC рекомендуется использовать фреймворк Entity Framework, хотя его использование необязательно и всецело зависит от предпочтений разработчика. Преимущество этого фреймворка состоит в том, что он позволяет абстрагироваться от структуры конкретной базы данных и вести все операции с данными через модель.

На данном этапе наш проект не содержит библиотек EntityFramework. И чтобы их добавить в проект используется пакетный менеджер NuGet. Итак, в окне Обозреватель решений нажмем правой кнопкой мыши в структуре проекта на узел References и в появившемся меню выберем Управление пакетами NuGet...(рисунок 8).

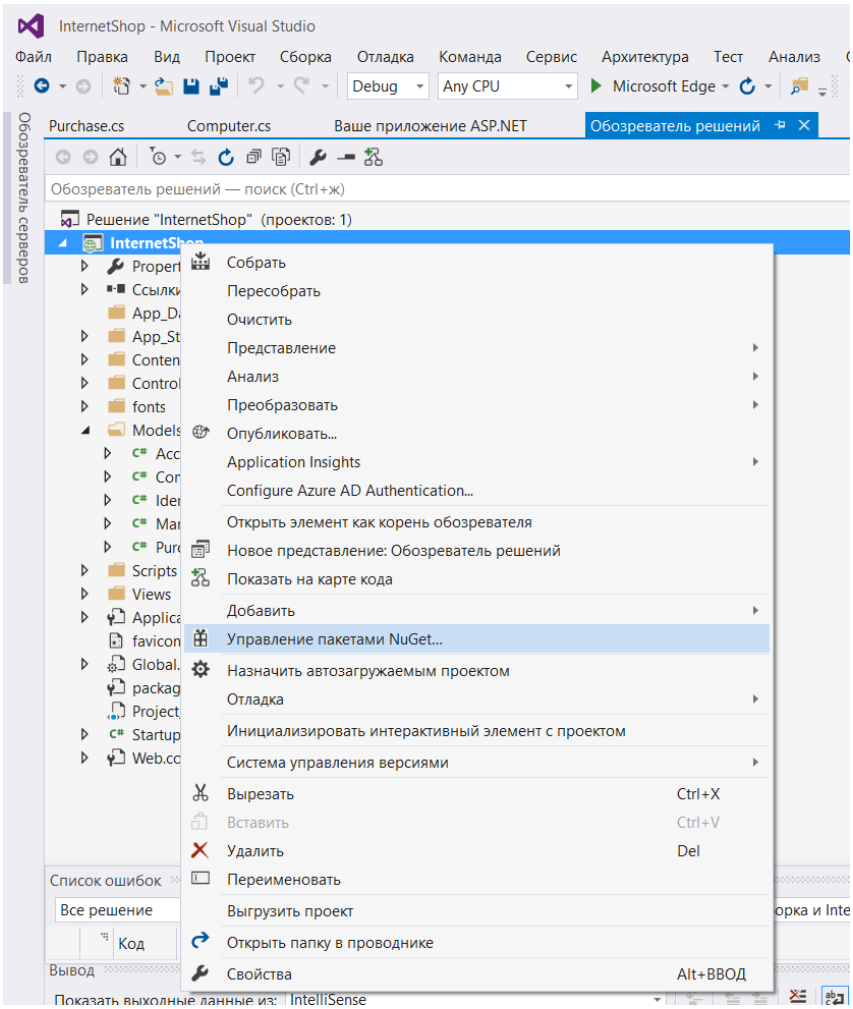


Рисунок 8

10. В окне управления пакетами NuGet в правом верхнем углу введите в поле поиска EntityFramework и нажмите Enter. После этого в среднем столбце будут отображены все найденные пакеты, которые имеют отношение к запросу, а самым первым будет пакет самого фреймворка EntityFramework, который нам и надо установить (рисунок 9):

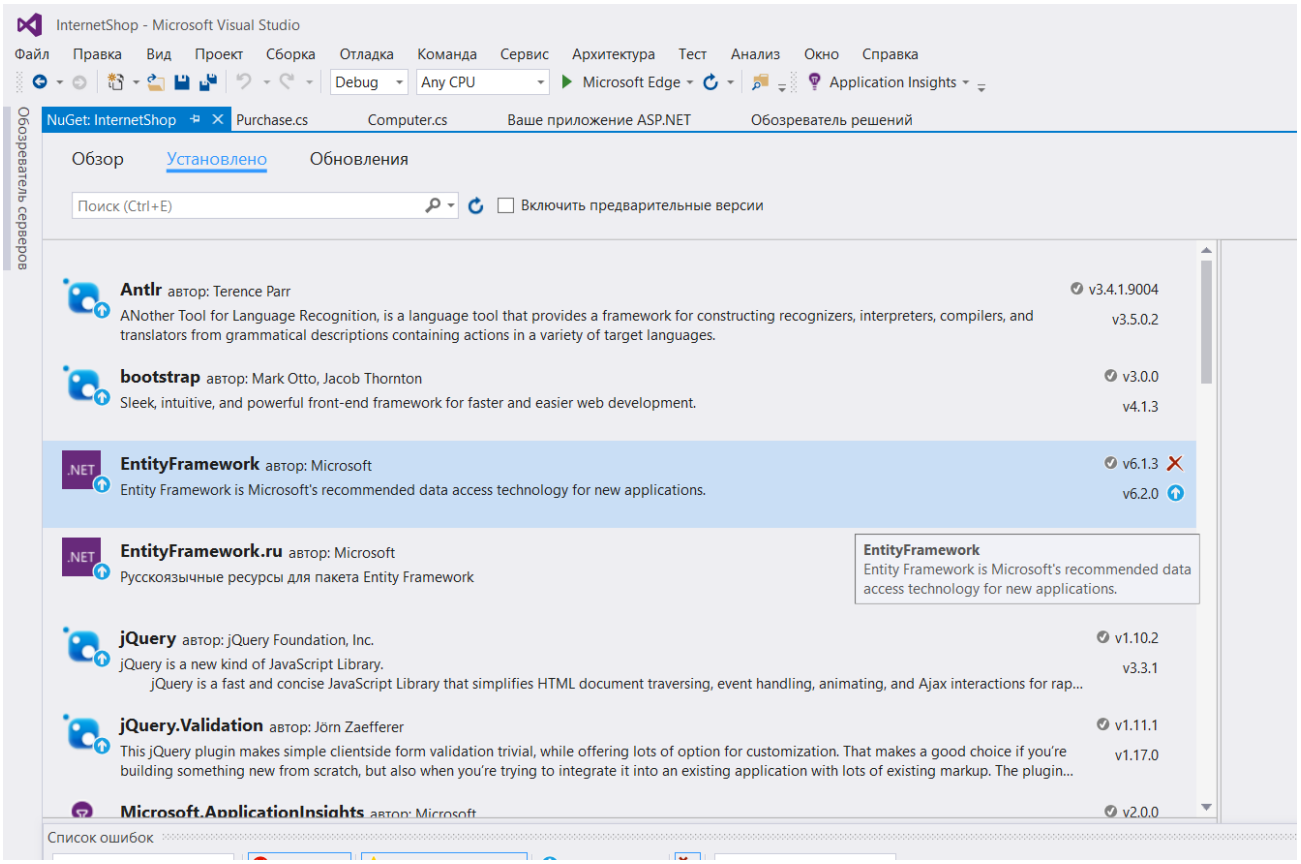


Рисунок 9

11. Следующий этап – это создание контекста данных. После завершения установки создадим контекст данных. Контекст данных использует EntityFramework для доступа к БД на основе некоторой модели. Итак, добавим в папку Models новый класс ComputerContext:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;

namespace InternetShop1.Models
{
    public class ComputerContext : DbContext
    {
        public DbSet<Computer> Computer { get; set; }
        public DbSet<Purchase> Purchases { get; set; }
    }

   }
```

12. Чтобы создать контекст, нам надо унаследовать новый класс от класса DbContext. Свойства наподобие public DbSet<Computer> Computer { get; set; } помогают получать из БД набор данных определенного типа (например, набор объектов Computer).

Хотя мы будем использовать базу данных, но создавать явным образом мы пока ее не будем. За нас все сделает EntityFramework. Это так называемый

подход Code First - у нас есть модели, и по ним фреймворк будет создавать таблицы в базе данных.

13. Далее над модельной частью установим строку подключения. Для этого откроем файл web.config, найдем секцию configSections и сразу после нее вставим секцию connectionStrings.

14. Файл написан в формате XML. То есть, мы вставляем в корневой элемент configuration дочерний элемент connectionStrings, если он отсутствует, но он там может уже быть, поэтому, новую строку мы добавляем уже внутрь connectionStrings. Итоговая структура Web.config, в общем, такая:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  Дополнительные сведения о настройке приложения ASP.NET см. по адресу:
  http://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" />
  <!-- For more information on Entity Framework configuration, visit
http://go.microsoft.com/fwlink/?LinkID=237468 --></configSections>

  <connectionStrings>
    <add name="CompuerContext"
         connectionString="Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename='|DataDirectory|\InternetShop.mdf';Initial
Catalog=InternetShop;Integrated Security=True"
         providerName="System.Data.SqlClient" />
</connectionStrings>
```

…….

В этой секции задается путь к базе данных, которая затем будет создаваться. Выражение |DataDirectory| представляет заместитель, который указывает, что база данных будет создаваться в проекте в папке App_Data. Позднее мы подробнее разберем настройки подключения к БД.

15. Следующий этап работы – это работа с компонентом приложения контроллером. Для контроллеров предназначена папка Controllers. По умолчанию при создании проекта в нее добавляется контроллер HomeController, который практически не имеет никакой функциональности, и сейчас его код выглядит следующим образом:

```
using InternetShop.Models;
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace InternetShop.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            // создаем контекст данных
            using (ComputerContext db = new ComputerContext())
            {
                // получаем из бд все объекты Computer
                List<Computer> Computers = db.Computers.ToList();
                // передаем все объекты в динамическое свойство Computer в ViewBag
                ViewBag.Computers = Computers;
                // возвращаем представление
                return View();
            }
            // после выхода из using объект ComputerContext будет удален из памяти и
подключение к БД будет закрыто
        }

        public ActionResult About()
        {
            ViewBag.Message = "Your application description page.";

            return View();
        }

        public ActionResult Contact()
        {
            ViewBag.Message = "Your contact page.";

            return View();
        }
    }
   }
```

16. Автоматически создаются методы контроллера Index(), About(), Contact().

17. Прежде всего, мы подключаем пространство имен моделей, даже не смотря на то, что он находятся в одном проекте, но в разных пространствах. Затем создается объект контекста данных, через который мы будем взаимодействовать с БД: ComputerContext db = new ComputerContext(); , т.е. создадим в этом контроллере метод .

```
using System;
    using System.Collections.Generic;
using System.Linq;
    using System.Web;
    using System.Web.Mvc;
    using InternetShop1.Models;


namespace InternetShop1.Controllers
{
  public class HomeController : Controller
    {
```

```
        // создаем контекст данных
       ComputerContext db = new ComputerContext();

        public ActionResult Index()
        {
            // получаем из бд все объекты Computer
           IEnumerable<Computer> computers = db.Computer;
            // передаем все полученный объекты в динамическое свойство Books в ViewBag
             ViewBag.Computers = computers;
            // возвращаем представление
                return View();
         }
     }
    }
```

И подключим пространство имен проекта в разделе пространства имен HomeController:

```
using InternetShop.Models;
```

18. Далее используя свойство db.Computer, получаем из базы данных набор объектов Computer. Теперь надо передать этот набор в представление.

19. Для передачи списка объектов Computer в представление используем объект ViewBag. ViewBag представляет такой объект, который позволяет определить любую переменную и передать ей некоторое значение, а затем в представлении извлечь это значение. Так, мы определяем переменную ViewBag.Computer, которая и будет хранить набор компьютеров.

20. Теперь создадим само представление для вывода списка компьютеров. Для представлений в проекте предназначена папка Views. По умолчанию в этой папке уже есть подкаталог для представлений контроллера Home, в котором три представления: About.cshtml, Contact.cshtml и Index.chtml.

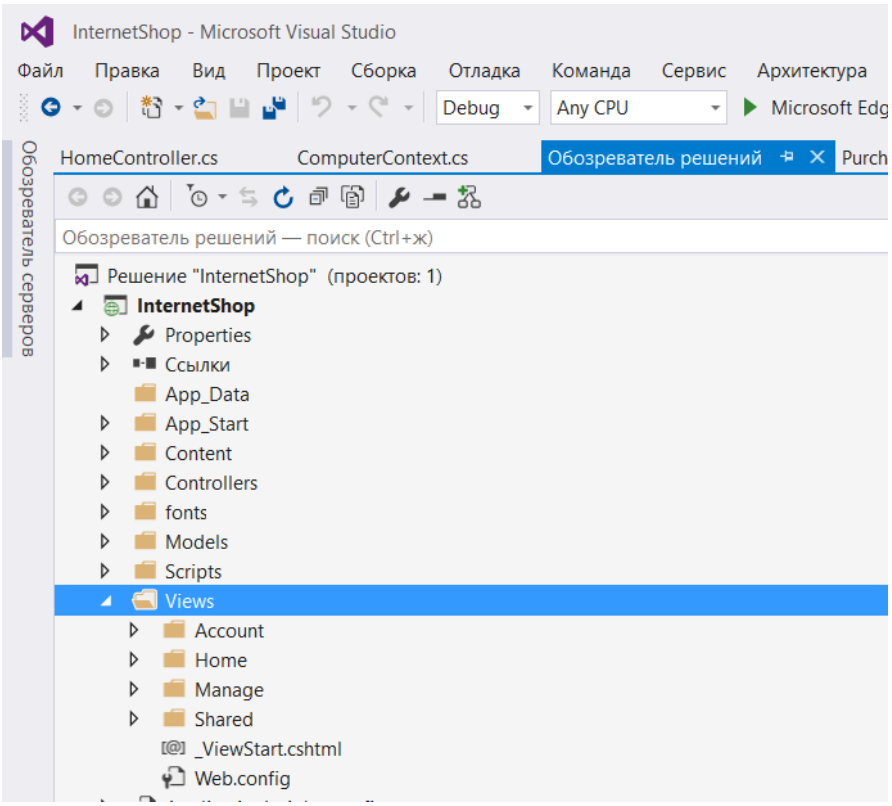


Рисунок 10

21. Перейдем к представлению **Index.cshtml** откроем и изменим следующим образом:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Интернет магазин</title>
</head>
<body>
 <div>
     <h3>Распродажа компьютеров</h3>
        <table>
         <tr>
             <td><p>Модель компьютера</p></td>
             <td><p>Страна</p></td>
               <td><p>Цена</p></td>
                <td></td>
         </tr>
            @foreach (var b in ViewBag.Computers)
            {
                <tr>
                    <td><p>@b.Name</p></td>
                    <td><p>@b.Country</p></td>
                    <td><p>@b.Price</p></td>
                    <td><p><a href="/Home/Buy/@b.Id">Купить</a></p></td>
                </tr>
            }
        </table>
    </div>
</body>
</html>
```

Самым первым выражением Layout = null; мы указываем, что мастер-страница не будет применяться к этому представлению. Далее мы добавим к нему мастер-страницу и узнаем, зачем она нужна, а пока обойдемся без нее.

22. Практически весь остальной код представляет собой стандартный код на языке html: создание обычной таблицы, которая выводит информацию о продаваемых компьютерах. Здесь также используется интересная конструкция @foreach (var b in ViewBag.Computer). Эта конструкция применяет синтаксис Razor. Подробнее о движке Razor и его синтаксисе мы рассмторим позже, а пока вам надо знать, что после символа @ согласно синтаксису мы можем использовать выражения кода на языке C#/VB.NET.

23. То есть тут мы создаем цикл. В нем мы пробегаемся по всем элементам в объекте ViewBag.Computer, который был ранее создан в методе контроллера. И затем получаем значение свойства каждого элемента с помощью синтаксиса Razor: @b.Name и помещаем его в ячейку таблицы.

24. В последнюю колонку таблицы для каждого элемента добавляется ссылка Купить. При нажатии на эту ссылку методу Buy контроллера HomeController будет отправляться запрос, в котором вместо @b.Id будет указан id книги. Пока у нас, правда, отсутствует метод Buy, но скоро мы его создадим. Если вы проведете компиляцию проекта, то у вас сформируется страница с отображением заголовков таблицы (рисунок 11)..

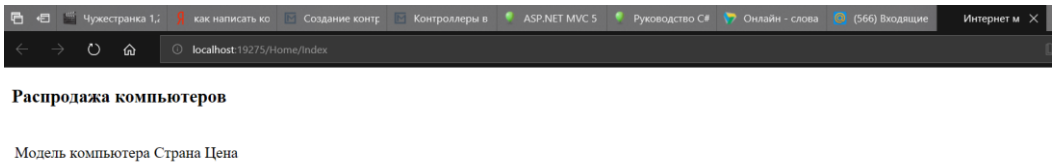


Рисунок 11

25. Необходимо провести настройку обращения к контроллеру. Чтобы обратиться к контроллеру HomeController или отправить ему запрос, нам надо указать в строке запроса его имя - Home. Кроме того, после имени контроллера нам надо через слеш указать действие или метод контроллера, к которому отправляется запрос. По умолчанию при запуске проекта или при обращении к сайту система mvc будет вызывать действие Index контроллера HomeController, если мы не укажем иной маршрут по умолчанию в параметрах маршрутизации.

Путь /Home/Buy означает, что мы будем обращаться к методу Buy контроллера HomeController. А добавление в запрос параметра /Home/Buy/@b.Id, означает, что такой метод может принимать параметр. Но перед тем как создать этот метод, наполним приложение данными.

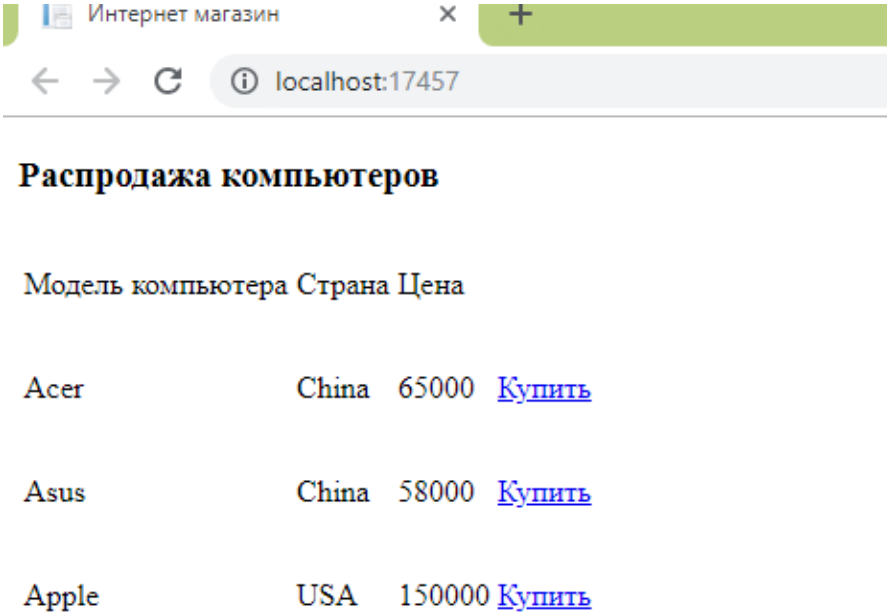
26. **Данные для моделей по умолчанию**. Так как мы будем использовать подход Code First, то нам не надо вручную создавать базу данных и наполнять ее данными. Мы можем воспользоваться специальным классом, который за нас добавит начальные данные в БД. Для этого в папку Models добавим новый класс ComputerDbInitializer и изменим его код следующим образом:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;

namespace InternetShop.Models
{
    public class ComputerDbInitializer : DropCreateDatabaseAlways<ComputerContext>
    {
        protected override void Seed(ComputerContext db)
        {
            db.Computer.Add(new Computer { Name = "Acer", Country = "China", Price =
       65000 });
            db.Computer.Add(new Computer { Name = "Asus", Country = "China", Price =
       58000 });
               db.Computer.Add(new Computer { Name = "Apple", Country = "USA", Price =
   150000 });

            base.Seed(db);
        }
    }
}
```

27. Класс DropCreateDatabaseAlways позволяет при каждом новом запуске заполнять базу данных заново некоторыми начальными данными. В качестве таких начальных значений здесь создаются три объекта Computer. Используя метод db.Computer.Add мы добавляем каждый такой объект в базу данных. Однако чтобы этот класс действительно сработал, и заполнение базы данных произошло, нам надо запустить его при запуске приложения. Все начальные настройки приложения и конфигурации находятся в файле Global.asax. Откроем его и добавим в метод Application_Start, который отрабатывает при старте приложения, следующую строку Database.SetInitializer(new ComputerDbInitializer());:

```
using System.Web.Mvc;
using System.Web.Optimization;
using InternetShop1.Models;
using System.Web.Routing;
using System.Data.Entity;

namespace InternetShop
{
        public class MvcApplication : System.Web.HttpApplication
        {
            protected void Application_Start()
         {
             Database.SetInitializer(new ComputerDbInitializer());
             AreaRegistration.RegisterAllAreas();
             FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
              RouteConfig.RegisterRoutes(RouteTable.Routes);
             BundleConfig.RegisterBundles(BundleTable.Bundles);
         }
     }
    }
```

Запустите теперь на компиляцию проект. Результат компиляции на рисунке



Лабораторная работа № 4. Технология MVC. Контроллер

Цель и содержание работы: создание стилизации приложения и разработка методов контроллера

Теоретическое обоснование

Контроллер управляет запросами пользователя (получаемые в виде запросов HTTP GET или POST, когда пользователь нажимает на элементы интерфейса для выполнения различных действий). Его основная функция — вызывать и координировать действие необходимых ресурсов и объектов,

нужных для выполнения действий, задаваемых пользователем. Обычно контроллер вызывает соответствующую модель для задачи и выбирает подходящий вид.

Контроллеры часто нуждаются в доступе к данным из входящего запроса, таким как значения строки запроса, значения формы и параметры, извлеченные из URL системой маршрутизации. Существуют три основных способа доступа к таким данным:

- извлечение данных из набора объектов контекста;
- передача данных в качестве параметров методу действия;
- явное обращение к средству привязки моделей инфраструктуры.

Получение данных из объектов контекста.

Создание контроллера путем его наследования от базового класса Controller, то получают в свое распоряжение набор удобных свойств для доступа к информации, касающейся запроса. К таким свойствам относятся Request, Response, RouteData, HttpContext и Server. Каждое перечисленное свойство отвечает за конкретный аспект запроса. Называют их удобными свойствами, поскольку каждое из них извлекает определенный тип данных из экземпляра ControllerContext для запроса (который доступен через свойство Controller.ControllerContext).

Наиболее часто используемые объекты контекста и свойства описаны в таблице ниже:

Свойство	Тип	Описание
Request.QueryString	NameValueCollection	Переменные GET, отправленные с этим запросом
Request.Form	NameValueCollection	Переменные POST, отправленные с этим запросом

Request.Cookies	HttpCookieCollection	Cookie-наборы, отправленные браузером с этим запросом
Request.HttpMethod	string	Метод HTTP (команда наподобие GET или POST), используемый для этого запроса
Request.Headers	NameValueCollection	Полный набор заголовков HTTP, отправленных с этим запросом
Request.Url	Uri	Элемент RouteTable.Routes, выбранный для этого запроса
Request.UserHostAddress	string	IP-адрес пользователя, сделавшего этот запрос
RouteData.Route	RouteBase	Элемент Routetable.Routes, выбранный для этого запроса
RouteData.Values	RouteValueDictionary	Активные параметры маршрута (либо извлеченные из URL, либо стандартные

		значения)
HttpContext.Application	HttpApplicationStateBase	Хранилище состояния приложения
HttpContext.Cache	Cache	Хранилище кеша приложения
HttpContext.Items	IDictionary	Хранилище состояния для текущего запроса
HttpContext.Session	HttpSessionStateBase	Хранилище состояния для сеанса посетителя
User	IPrincipal	Аутентификационная информация о вошедшем пользователе
TempData	TempDataDictionary	Временные элементы данных, сохраненные для текущего пользовател

Создание объектов параметров

Базовый класс Controller получает значения для параметров методов действий с использованием компонентов MVC Framework, называемых **поставщиками значений** и **связывателями моделей**. Поставщики значений представляют набор элементов данных, доступных контроллеру. Существуют встроенные поставщики значений, которые производят выборку элементов из Request.Form, Request.QueryString, Request.Files и RouteData.Values. Эти значения затем передаются связывателям моделей, которые пытаются отобразить их на типы параметров, принимаемых методами действий.

Стандартные связыватели моделей могут создавать и заполнять объекты любого типа .NET, включая коллекции и специальные типы, специфичные для проектов. Соответствующий пример приводился в статье Админ панель: редактирование товаров, в котором отправляемая администратором форма была представлена методу действия как одиночный объект Game, несмотря на то, что индивидуальные значения были разбросаны по элементам HTML-формы.

Контроллер является центральным компонентом в архитектуре MVC. Контроллер получает ввод пользователя, обрабатывает его и посылает обратно результат обработки, например, в виде представления.

При использовании контроллеров существуют некоторые условности. Так, по соглашениям об именовании названия контроллеров должны оканчиваться на суффикс "Controller", остальная же часть до этого суффикса считается именем контроллера.

Методика выполнения работы

В предыдущей работе была получена таблица товаров, но она выглядит немного неказисто. Поэтому необходимо ей придать стиль.

Но для начала создадим метод Buy, который будет отвечать за обработку ввода пользователя при оформлении покупки. Добавим в контроллер HomeController следующие два метода:

```
[HttpGet]
        public ActionResult Buy(int id)
        {
            ViewBag.ComputerId = id;
            return View();
        }
        [HttpPost]
        public string Buy(Purchase purchase)
        {
            purchase.Date = DateTime.Now;
            // добавляем информацию о покупке в базу данных
            db.Purchases.Add(purchase);
            // сохраняем в бд все изменения
            db.SaveChanges();
            return "Спасибо," + purchase.Person + ", за покупку!";
               }
```

В этих методе public ActionResult Buy(int id) было определено действие Buy, которое будет выполняться при получении запроса GET.

В public string Buy(Purchase purchase) будет выполняться действие при получении запроса POST. Каждый запрос определен с помощью атрибутов [HttpGet] и [HttpPost]. Метод public ActionResult Buy(int id) очень прост - он просто принимает id выбранного компьютера и возвращает для нее соответствующее представление.

Метод public string Buy(Purchase purchase) выглядит несколько сложнее. Он принимает переданную ему в запросе POST модель purchase и добавляет ее в базу данных. В конце возвращаем пользователю строку сообщения.

Если проведете компиляцию и в полученной странице кликните на ссылку Купить, то получите окно с сообщением (рисунок 12)

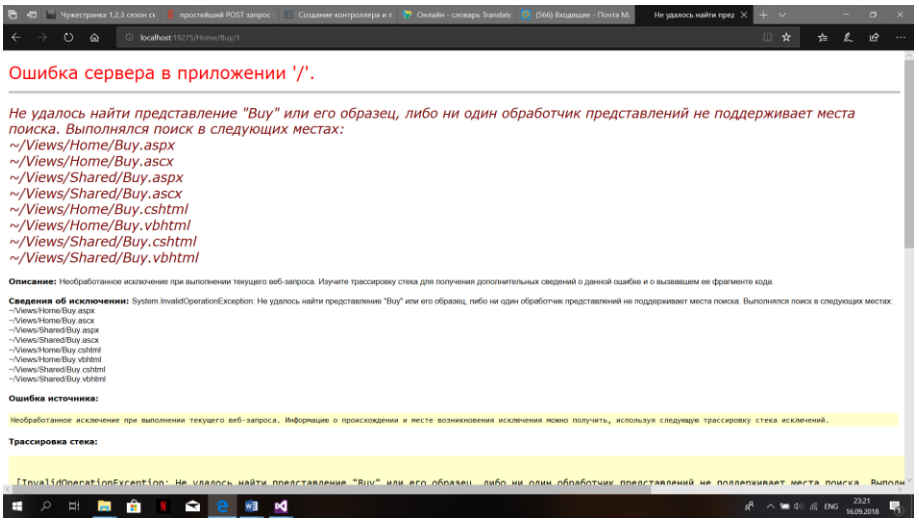


Рисунок 12

Это означает, что еще не создано представление для этих методов.

Таким образом, необходимо добавить представление Buy. Для этого нажмем на метод public ActionResult Buy(int id) правой кнопкой и добавим в проект новое представление (рисунок 13).

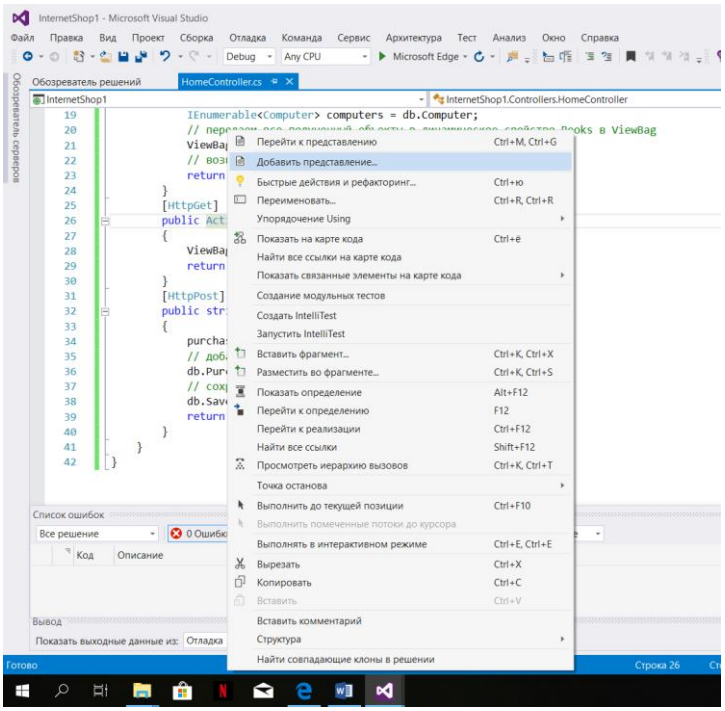
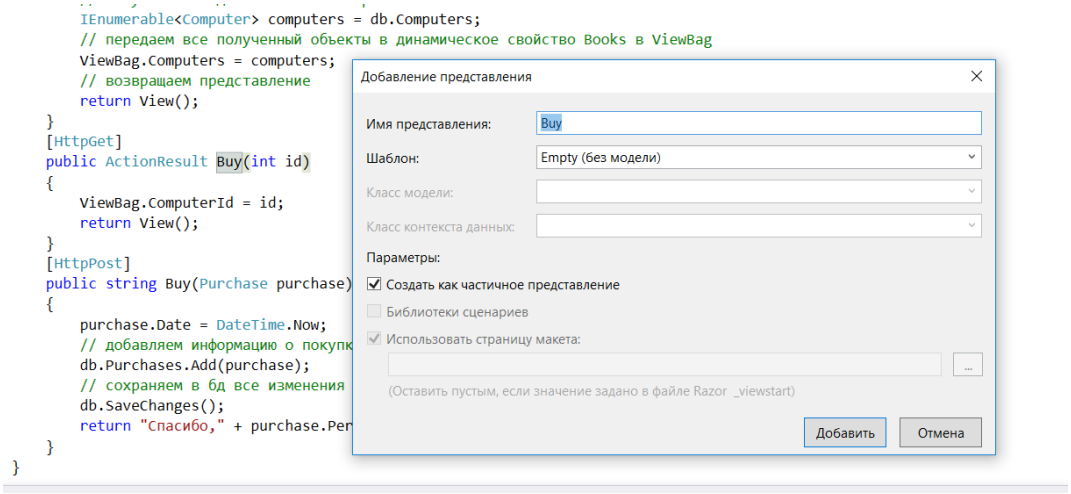


Рисунок 13

После выбора команды «Добавить представление» откроется диалоговое окно добавления представления, как показано на рисунке 14.



Задайте Имя представление Buy, выберите шаблон Empty (без модели) и установите флажок опции «Создать как частичное представление). При этом создастся пустой шаблон Empty с пустой страницей.

Введите код нового представления следующим образом, используя HTML:

```
@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Покупка</title>
</head>
```

```
<body>
    <div>
        <h3>Форма оформления покупки</h3>
        <form method="post">
            <input type="hidden" value="@ViewBag.ComputerId" name="ComputerId" />
            <table>
                <tr>
                    <td><p>Введите свое имя </p></td>
                    <td><input type="text" name="Person" /> </td>
                </tr>
                <tr>
                    <td><p>Введите адрес :</p></td>
                    <td>
                        <input type="text" name="Address" />
                    </td>
                </tr>
                <tr><td><input type="submit" value="Отправить" /> </td><td></td></tr>
            </table>
        </form>
    </div>
</body>
       </html>
```

Запустите проект и кликните на ссылку Купить. Сформируется шаблон формы для ввода пользователем своих данных (рисунок 15)

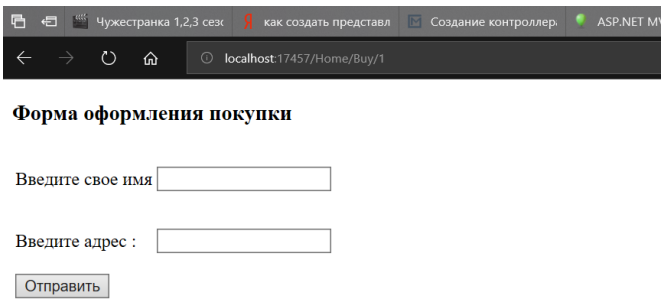


Рисунок 15

Проверьте как работает форма. Для этого введите любые данные, например, рисунок 16 и щелкните на кнопку Отправить. В результате получите ответ рисунок 17.

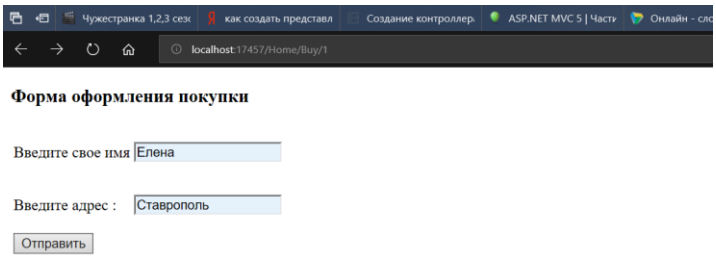


Рисунок 16

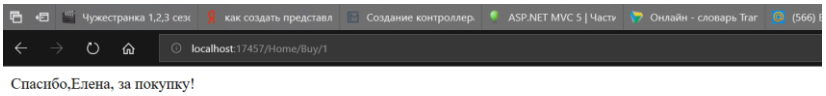


Рисунок 17

Таким образом, была создана форма ввода данных. При переходе по ссылке "/Home/Buy/2" контроллер будет получать запрос к действию Buy. И так как такой запрос представляет тип GET, пользователю будет возвращаться форма. После заполнения формы и нажатия на кнопку форма будет оправляться запросом POST, так как мы его определили в строке <form method="post" action="">. Контроллер снова будет получать запрос к методу Buy, только теперь будет выбираться для обработки запроса метод public string Buy(Purchase purchase).

Каким образом система MVC угадывает, что было передано с запросом post информацию о модели. В полях ввода <input type="text" name="Person" /> и <input type="text" name="Address" /> их свойства name соответствуют именам свойств модели Purchase. При нажатии кнопки и отправки запроса передаются значения этих полей. Система MVC, используя соглашения по умолчанию, считает эти значения значениями соответствующих свойств модели.

Стилизация сайта

Следующим этапом добавим к сайту элементы стилизации в виде стилей css, а также мастер-страницы, которые позволяют придать всем страницам сайта единообразный вид.

Для этого нам надо определить файл стилей для веб-приложения. Все файлы стилей принято хранить в папке Content. Поэтому вначале перейдем в проекте в Обозревателе решений к папке Content.

Теперь в новую папку добавим файл стилей. Назовем его Site.css. Для этого нажмем правой кнопкой мыши на папку Content и в появившемся меню выберем Add (Добавить)->New Item (Новый элемент). Затем в списке шаблонов найдем шаблон Таблица стилей (Style Sheet) и дадим новому файлу имя Site.css.

Если он был автоматически создан в проекте, то просто найдите файл Site.css и откройте его.

Добавьте в файл Site.css следующее содержание:

```
body {
    font-size: 13px;
    font-family: Verdana, Arial, Helvetica, Sans-Serif;
    background-color: #f7f7fa;
    padding-left:40px;
}

nav{
    display: block;
}

.menu {
    padding-left:10px;
}
.menu ul {
    list-style:none;
}
.menu li {
    display:inline;
}
.menu a:hover {
    color:red;
}
table {
   vertical-align:middle;
   text-align:left;
 }
.header {
    font-weight:bold;
}
 td {
    padding-right:10px;
 }
input {
    width: 150px;
}
```

Класс .menu в данном случае будет служить в качестве класса для навигационного меню на сайте.

Теперь описанные стили будут использоваться на страницах веб-приложения. Поскольку созданные ранее представления – это обычные веб-страницы, то на любой странице html можно добавить в секции head ссылки на файл стилей:

```
<head>
    <meta name="viewport" content="width=device-width" />
    <link type="text/css" rel="stylesheet" href="../../Content/Site.css" />
      </head>
```

Если в приложении одно представление, или если разные представления используют разные стили и другие ресурсы, то такой подход оправдан. Но если в приложении имеется несколько представлений, как в данном проекте, то эти представления должны быть одинаково стилизованы. В этом случае используется мастер-страница. Мастер-страница задает некий единый шаблон для других использующих его представлений.

Итак, создадим мастер-страницу. По сути это простое представление. Перейдем в Обозревателе решений в папку Views / Shared. После этого в папку Shared выберем файл _Layout.cshtml:

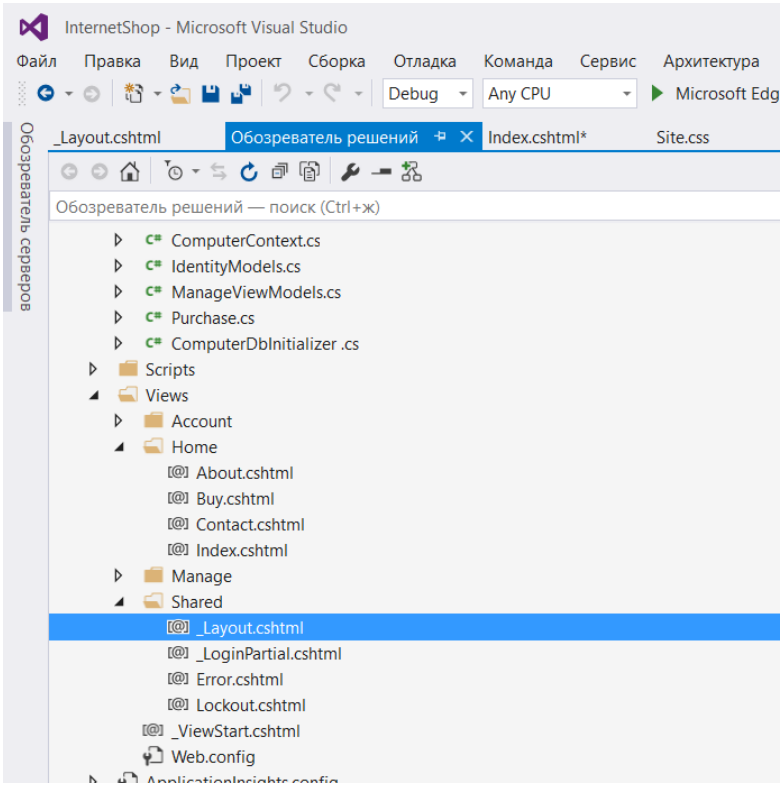


Рисунок 18

И добавим в этом следующий текст:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
</head>

<body>
    <nav>
        <ul class="menu">
            <li>@Html.ActionLink("Главная", "Index", "Home")</li>
        </ul>
    </nav>
    @RenderBody()
</body>
```

```
</html>
```

Меню сайта было создано с помощью методов Html.ActionLink, который генерирует элемент-ссылку и принимает название ссылки, метод контроллера и имя контроллера. В конце идет метод RenderBody(), при вызове ресурса в эту часть мастер-страницы и будут подставляться содержимое остальных представлений.

Теперь адаптируем представления к использованию их мастер-страницей. Изменим представление Index.cshtml следующим образом:

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}


    <div>
        <h3>Распродажа компьютеров</h3>
        <table>
            <tr>
                <td><p>Модель компьютера</p></td>
                <td><p>Страна</p></td>
                <td><p>Цена</p></td>
                <td></td>
            </tr>
            @foreach (var b in ViewBag.Computers)
            {
                <tr>
                    <td><p>@b.Name</p></td>
                    <td><p>@b.Country</p></td>
                    <td><p>@b.Price</p></td>
                    <td><p><a href="/Home/Buy/@b.Id">Купить</a></p></td>
                </tr>
            }
        </table>
           </div>
```

Запустите на отладку проект и увидите, что на странице добавилась ссылка Главная (рисунок 19).

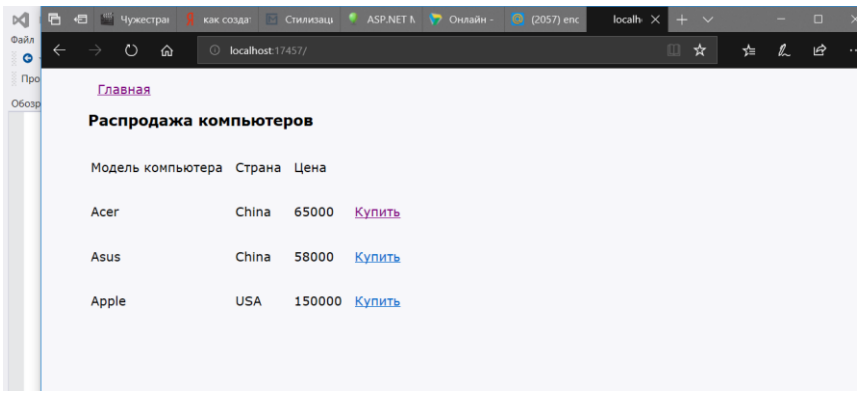


Рисунок 19

Также отредактируем представление Buy.cshtml:

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div>
    <h3>Форма оформления покупки</h3>
    <form method="post" action="">
        <input type="hidden" value="@ViewBag.ComputerId" name="ComputerId" />
        <table>
            <tr>
                <td><p>Введите свое имя </p></td>
                <td><input type="text" name="Person" /> </td>
            </tr>
            <tr>
                <td><p>Введите адрес :</p></td>
                <td>
                    <input type="text" name="Address" />
                </td>
            </tr>
            <tr>
                <td><input type="submit" value="Отправить" /> </td>
                <td></td>
            </tr>
        </table>
    </form>
      </div>
```

Как видно на рисунке 20, Форма оформления покупки уже тоже имеет ссылку Главная

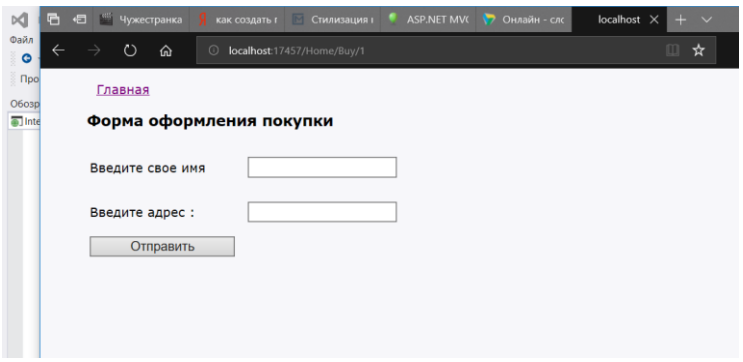


Рисунок 20

Таким образом, сущность изменений состоит в том, что в секции Layout = "~/Views/Shared/_Layout.cshtml" указывается путь к мастер-странице. После этого не нужны секции head и body.

Лабораторная работа 4. Модели. Создание базы данных

Цель работы: построить базу данных, научиться пользоваться фреймворк Entity Framework

Теоретическое обоснование

Все сущности в приложении принято выделять в отдельные модели. В зависимости от поставленной задачи и сложности приложения можно выделить различное количество моделей. При создании макета были созданы модели - класс для компьютера и класс для покупки компьютера.

Модели представляют собой простые классы и располагаются в проекте в каталоге *Models*. Модели описывают логику данных.

В предыдущих лабораторных работах, для работы с базой данных использовался фреймворк Entity Framework, который позволяет абстрагироваться от написания sql-запросов, от строения базы данных и полностью сосредоточиться на логике приложения.

Методические указания к выполнению

Для подключения к базе данных через Entity Framework, необходим посредник - **контекст данных**. Контекст данных представляет собой класс, производный от класса DbContext. Контекст данных содержит одно или несколько свойств типа DbSet<T>, где T представляет тип объекта, хранящегося в базе данных. Ранее нами был создан контекст данных для работы с моделями Компьютеры и Покупка:

```
using System.Data.Entity;

namespace InternetShop.Models
{


        public class ComputerContext : DbContext
        {
            public DbSet<Computer> Computers { get; set; }
            public DbSet<Purchase> Purchases { get; set; }
        }

    }
```

С помощью свойств Books и Purchases мы получаем доступ к данным соответствующих моделей, которые хранятся в базе данных.

Для хранения данных приложению нужна база данных. Мы можем использовать различные СУБД, но, как правило, в качестве базы данных в связке с ASP.NET MVC используется база данных MS SQL Server, на примере которого мы и посмотрим весь процесс создания БД и подключения к ней.

Мы можем создать базу данных прямо в проекте, либо же создать ее на сервере MS SQL. Для хранения баз данных в проекте предназначена папка App_Data. Итак, нажмем на папку App_Data правой кнопкой мыши и в появившемся контекстном меню выберем Дабавить/Новый элемент.... В появившемся окне добавления нового элемента выберем SQL Server Database и назовем новую базу данных InternetShop.mdf (рисунок 1):

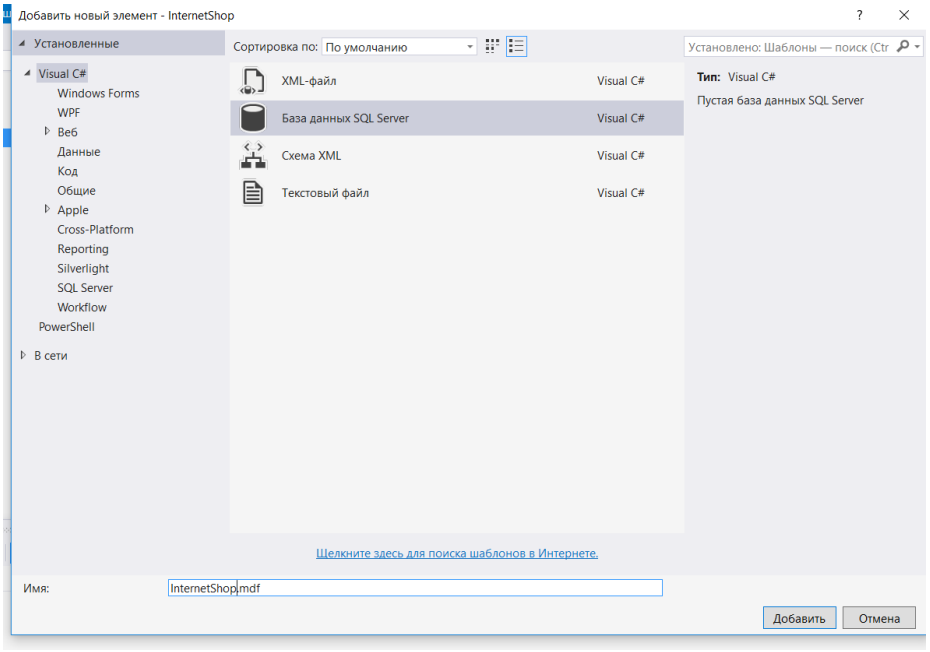
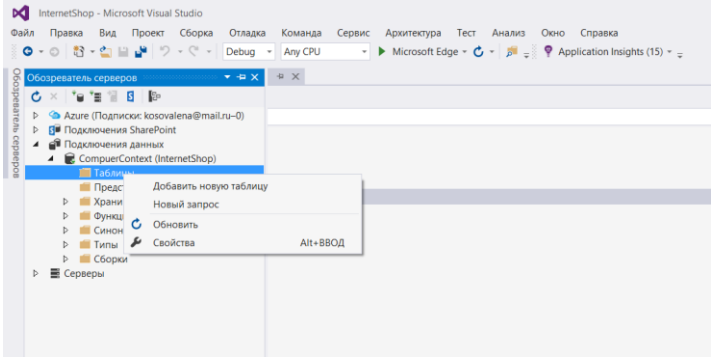


Рисунок 1 – Добавление базы данных

Добавим первую таблицу. Для этого нажмем правой кнопкой мыши на узел Таблицы и в появившемся меню выберем пункт Добавить новую таблицу.



После этого отобразится окно дизайнера таблицы, в котором надо определить названия и типы столбцов новой таблицы. По соглашениям о наименованиях таблицы при работе с Entity Framework названия таблиц должны соответствовать имени модели. Например, если модель называется Computers, то таблица будет называться Computers. А Entity Framework автоматически распознает, что таблица Computers соответствует классу Computer.

Также поскольку предполагается, что столбец Id будет инкрементироваться с добавлением каждого нового объекта, то установим для него автоинкремент в окне Properties (рисунок 2):

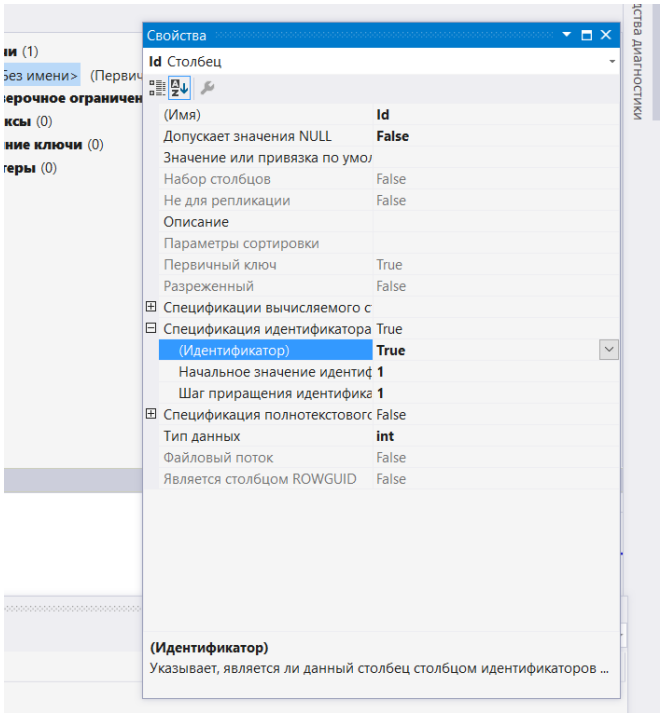
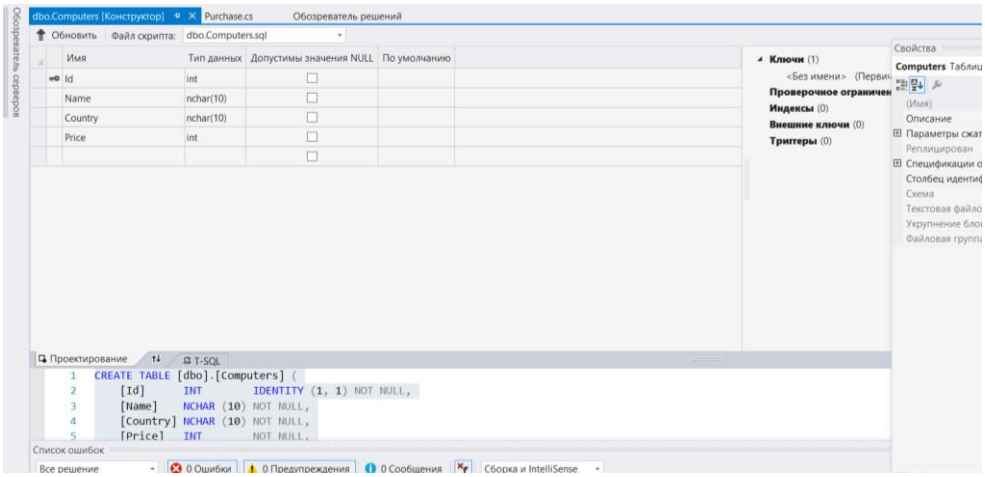
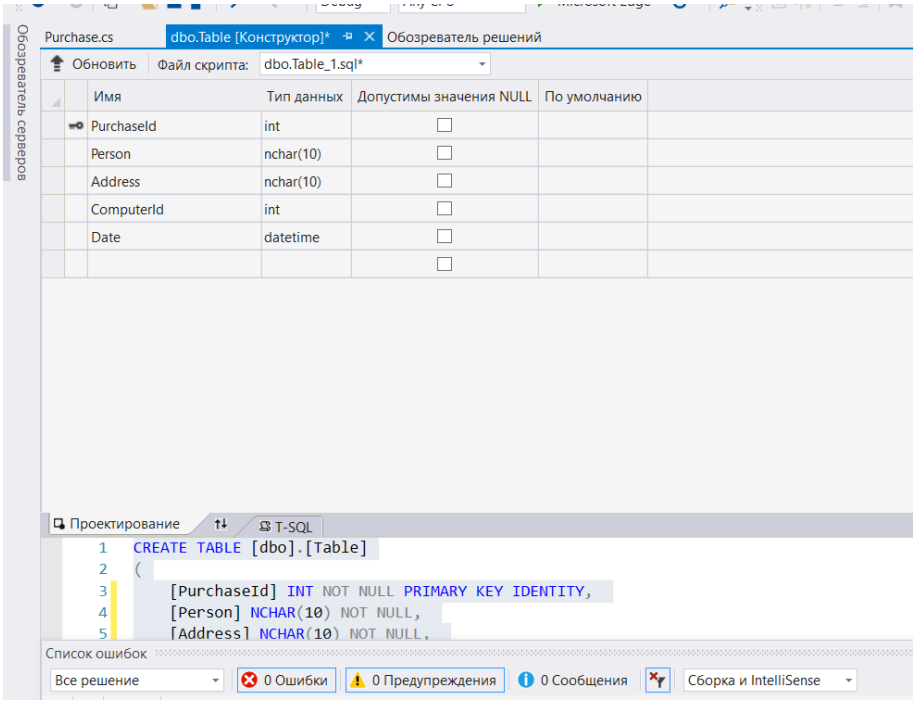


Рисунок 2 – Установка свойства автоинкремента

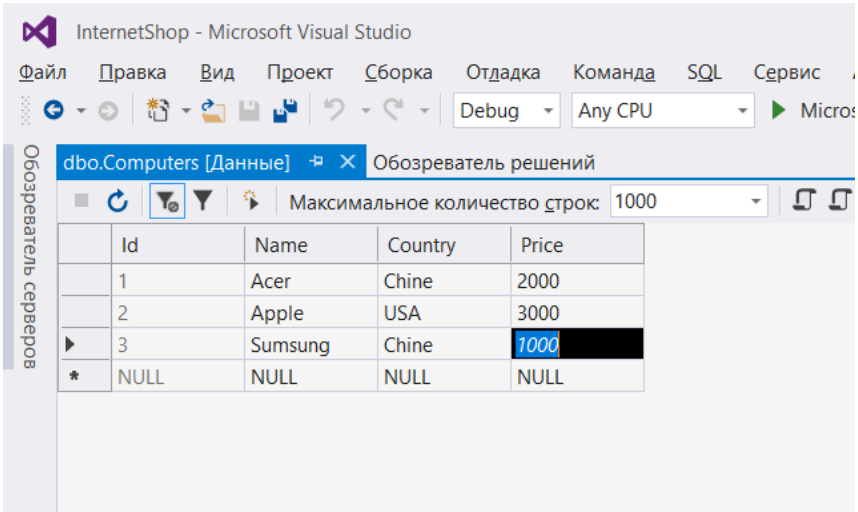
И последний шаг - генерация базы данных. Для этого нажмем на кнопку Обновить:



Аналогично создаем таблицу Покупка



Добавим в дизайнере баз данных в таблицу Computer несколько записей. Для этого нажмем в окне Обозреватель серверов на узел Computer и выберем в появившемся списке пункт Показать таблицу данных. Добавим, к примеру, следующий набор записей:



Теперь можем получить эти данные в контроллере Home и передать их в представление.

Для этого чтобы связать представление с передаваемым параметром, надо добавить в представление директиву @model с указанием типа передаваемых данных:

```
      @model IEnumerable<InternetShop.Models.Computer>
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<div>
    <h3>Распродажа компьютеров</h3>
    <table>
        <tr>
            <td><p>Модель компьютера</p></td>
            <td><p>Страна</p></td>
            <td><p>Цена</p></td>
            <td></td>
        </tr>
        @foreach (var b in Model)
            {
            <tr>
                <td><p>@b.Name</p></td>
                <td><p>@b.Country</p></td>
                <td><p>@b.Price</p></td>
                <td><p><a href="/Home/Buy/@b.Id">Купить</a></p></td>
            </tr>
        }
    </table>
      </div>
```

Лабораторная работа № 5. Работа с базой данных. Применение шаблонных хелперов

Цель работы: научиться применять шаблонные хелперы при работе с базой данных

Теоретическое обоснование

Фреймворк ASP.NET MVC обладает также таким мощным инструментом как хелперы. Хелперы бывают следующих видов:

1. HTML-хелперы, позволяющие генерировать html-код.
2. Строчные хелперы
3. Шаблонные хелперы.

В данной лабораторной работе рассмотрим шаблонные хелперы.

Шаблонные хелперы:

Display. Создает элемент разметки, который доступен только для чтения, для указанного свойства модели: Html.Display("Name")

DisplayFor. Строго типизированный аналог хелпера Display: Html.DisplayFor(e => e.Name)

Editor. Создает элемент разметки, который доступен для редактирования, для указанного свойства модели: Html.Editor("Name")

EditorFor. Строго типизированный аналог хелпера Editor: Html.EditorFor(e => e.Name)

DisplayText. Создает выражение для указанного свойства модели в виде простой строки: Html.DisplayText("Name")

DisplayTextFor. Строго типизированный аналог хелпера DisplayText: Html.DisplayTextFor(e => e.Name)

Кроме данных шаблонов, которые используются для отдельного свойства модели, есть еще несколько шаблонов, которые позволяют сгенерировать разом все поля для определенной модели:

DisplayForModel. Создает поля для чтения для всех свойств модели: Html.DisplayForModel()

EditorForModel. Создает поля для редактирования для всех свойств модели: Html.EditorForModel()

Методика выполнения работы

1. Напишем контроллер, который содержит в себе действия по обработке данных: редактирование, удаление, сохранение и показ модели. Для этого в папке Controller создадим контроллер ComputersController и напишем следующий код.

```
using InternetShop.Models; using System;
using System.Collections.Generic; using System.Linq;
using System.Web; using System.Web.Mvc;

namespace InternetShop.Controllers
{
    public class ComputersController : Controller
    {
        // создали подключение к базе, этот код вызовется при создании объекта
        private ComputerContext db = new ComputerContext();

        // функция вызывается при удалении объекта из памятие,
        // в этот момент нужно закрыть подключение к базе
        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose(); // вызываем "деструктор" внутри которого будет закрыто подключение
            }
            base.Dispose(disposing);
        }

        // Computers/List
        // список компьютеров
        public ActionResult List()
        {
            return View(db.Computers.ToList());
        }
```

```
public ActionResult Add()
{
    return View(new Computer());
```

```
        }

        [HttpPost]
        public ActionResult Add(Computer model)
        {
            db.Computers.Add(model);
        db.SaveChanges();

            return RedirectToAction("List");
        }

        // GET Computers/Edit/ID - компьютера в базе
        [HttpGet]
        public ActionResult Edit(int Id)
        {
            var model = db.Computers.Find(Id);

            if (model == null)
            {
              // если по данному ID в базе не найдено ничего то отобразить страницу 404
                return HttpNotFound();
            }

            return View(model);
        }

        // POST сохранить модель в базу по средством POST формы
        [HttpPost]
        public ActionResult Edit(Computer model)
        {
            try
            {
                // "присоединяем" полученную модель к базе и сохраняем изменения
              db.Entry<Computer>(model).State    = System.Data.Entity.EntityState.Modified;
                db.SaveChanges();
            }
            catch(Exception e)
            {
                ModelState.AddModelError(""
              , e);return View(model);
            }

            return RedirectToAction("List");
        }

        [HttpGet]
        public ActionResult Delete(int Id)
        {
```

```
            var model = db.Computers.Find(Id);
            if (model == null)
            {
                return HttpNotFound();
            }

            return View(model);
        }

        [HttpPost]
        public ActionResult DeleteConfirm(int Id)
        {
            var model = db.Computers.Find(Id);
            if (model != null)
            {
                db.Computers.Remove(model);db.SaveChanges();
            }
            return RedirectToAction("List");
        }

    }
}
```

2. Создадим в Представлении (папка Views) для каждого действия контроллера представления.

Добавление модели компьютера. Зададим имя этому представления Add

```
@model IEnumerable<InternetShop.Models.Computer>@{
    ViewBag.Title = "Список компьютеров";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2>Редактировать каталог компьютеров</h2>
<a href="@Url.Action("Add")">Добавить компьютер</a>
<div>
    <table>
        <thead>
            <tr>
                <td><p>Модель компьютера</p></td>
                <td><p>Страна</p></td>
                <td><p>Цена</p></td>
                <td></td>
            </tr>
        </thead>
        <tbody>
            @foreach (var b in Model)
```

{
<tr>

```
                                <td><p>@b.Name</p></td>
                                <td><p>@b.Country</p></td>
                                <td><p>@b.Price</p></td>
                                <td>
                                    <a href="@Url.Action("Edit", new { Id = b.Id
        })">Редактирова
    ть</a>                      <a href="@Url.Action("Delete", new { Id =

        b.Id
    })">Удалить</a>
                                </td>
                            </tr>
                        }
                    </tbody>
                </table>
            </div>
```

Удаление модели компьютера. Зададим имя этому представления Delete

```
        @model
InternetShop.Models.Computer@{
            ViewBag.Title = "Delete";
            Layout = "~/Views/Shared/_Layout.cshtml";
        }

        <h2>Подтверждение</h2>

        <div>
      <p>Удалить  @Model.Name  (страна:  @Model.Country,
цена:@Model.Price)?</p>
      @using  (Html.BeginForm("DeleteConfirm",  "Computers",
FormMethod.Post))
            {
                <input type="hidden" name="Id" value="@Model.Id" />
                <button type="submit">Да</button>
                <a href="@Url.Action("List")">Нет</a>
            }
        <
/div>
```

Редактирование модели компьютера. Зададим имя этому представления Edit.

```
        model
InternetShop.Models.Computer@{
            ViewBag.Title = "Редактирование";
            Layout = "~/Views/Shared/_Layout.cshtml";
```

```
}
@* https://metanit.com/sharp/mvc/4.7.php *@

<h2>Редактирование</h2>

<div>
    @using (Html.BeginForm("Edit", "Computers", FormMethod.Post))
    {
        <input type="hidden" value="@Model.Id" name="Id" />
        <label>Модель компьютера</label>
        <br />
        <input type="text" name="Name" value="@Model.Name" />
        <br />
        <label>Страна</label>
        <br />
        <input type="text" name="Country" value="@Model.Country" />
        <br />
        <label>Цена</label>
        <br />
        <input type="text" name="Price" value="@Model.Price" />
        <br />
        <button type="submit">Сохранить</button>

    }
  </div>
```

Список компьютеров. Зададим имя этому представления List

```
    @model
IEnumerable<InternetShop.Models.Computer>@{
        ViewBag.Title  =  "Список  компьютеров";
  Layout = "~/Views/Shared/_Layout.cshtml";
    }
    <h2>Редактировать каталог компьютеров</h2>
    <a href="@Url.Action("Add")">Добавить компьютер</a>
    <div>
        <table>
            <thead>
                <tr>
                    <td><p>Модель компьютера</p></td>
                    <td><p>Страна</p></td>
                    <td><p>Цена</p></td>
                    <td></td>
                </tr>
            </thead>
            <tbody>
                @foreach (var b in Model)
                {
                    <tr>
```

```
                    <td><p>@b.Name</p></td>
                    <td><p>@b.Country</p></td>
                    <td><p>@b.Price</p></td>
                    <td>
                        <a href="@Url.Action("Edit", new { Id = b.Id
        })">Редактирова
ть</a>                  <a href="@Url.Action("Delete", new { Id = b.Id

        })">Удалить</a>
```

```
                    </td>
                </tr>
            }
        </tbody>
</tabl
    e>
 </di
   v>
```

Просмотр модели компьютера. Зададим имя этому представленияShowModel

```
@{
    ViewBag.Title = "ShowModel";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>ShowModel</h2>
    @Html.DisplayForModel()
```

3. Добавим на главной странице ссылку «Редактировать» для открытия формы редактирования моделей компьютера. Для этого в папке представления выберем вложенную папку Shared и внесем в файл _Layout.cshtml строчку

```
<li>@Html.ActionLink("Редактирование", "List", "Computers")</li>
```

В целом код имеет вид:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>@ViewBag.Title</title>
<link  href="@Url.Content("~/Content/Site.css")"  rel="stylesheet"
```

```
type="text/css" />
        </head>

        <body>
            <nav>
                <ul class="menu">
                    <li>@Html.ActionLink("Главная",                    "Index",
                "Home")</li>
                <li>@Html.ActionLink("Редактирование",  "List",
"Computers")</li>
                </ul>
            </nav>
    @RenderBo
     dy()
      </body>
      </html>
```

4. Внесем в файл Site.css некоторые изменения для выделения четныхстрок таблицы для этого допишем код следующими строчками

```
        /*   убрать   границы   у
таблицы */table {
            border: 0;
            border-collapse: collapse;
        }

        /*   цвет   четных   строк   для   лучшей
читаемости */table tbody tr:nth-child(even) {
            background-color: #efefef;
        }
```