

Московский государственный технический университет  
имени Н. Э. Баумана

С. Н. Банников

## **РАЗРАБОТКА ПРОСТОГО ВЕБ-ПРИЛОЖЕНИЯ НА ЯЗЫКЕ C#**

*Методические указания к лабораторной работе по курсу  
«Разработка приложений на языке C#»*

Москва

Издательство МГТУ им. Н. Э. Баумана

2017

УДК xxx.xx.x  
ББК xx.xx  
XNN

*Рецензент .....*

Банников С.Н.

Разработка простого веб-приложения на языке C# : метод. указания к лабораторным работам по курсу «Разработка приложений на языке C#» / С. Н. Банников – М.: Изд-во МГТУ им. Н. Э. Баумана, 2016. - 87 с.: ил.

Изложены основы программирования на языке C# на примере создания простого веб-приложения на платформе ASP.NET (подмножество Microsoft.NET Framework) с использованием библиотеки WebForms. Дано описание синтаксиса языка в объеме, необходимом для реализации этого приложения. Приведена инструкция по работе с системой контроля версий исходных текстов TFS. Третий и четвертый разделы книги содержат справочные материалы по языку C#.

Для студентов, обучающихся по специальности «Компьютерные системы, комплексы и сети».

УДК xxx.xx.x  
ББК xx.xx

*Учебное издание*

Банников Сергей Николаевич

Разработка простого веб-приложения на языке C#

Редактор, Корректор

Компьютерная верстка

Подписано в печать xx.xx.xxxx. Формат

Усл. печ. л. Тираж 0 экз. Изд №. Заказ.

Издательство МГТУ им. Н. Э. Баумана

Типография МГТУ им. Н. Э. Баумана

105005, Москва, 2-я Бауманская ул., 5

© МГТУ им. Н. Э. Баумана, 2017

## **ВВЕДЕНИЕ**

Язык программирования C# является достаточно востребованным среди сообщества разработчиков программного обеспечения во всем мире. По различным оценкам (RedMonk, IEEE Spectrum, TIOBE Software), язык C# занимает от 4-го до 6-го места, в процентном отношении это составляет от 5% до 30%. Разработчики для платформы Microsoft.NET Framework пользуются спросом на рынке труда, а язык C# является основным языком программирования для данной платформы.

Целями данной лабораторной работы являются:

1. Закрепление навыков работы со средой разработки Microsoft Visual Studio
2. Закрепление навыков работы с системой контроля версий Team Foundation Service
3. Закрепление навыков использования синтаксических элементов языка C#
4. Знакомство с базовыми возможностями библиотеки Web Forms и программной архитектуры ASP.NET
5. Создание простого веб-приложения.

Перед выполнением данной лабораторной работы необходимо выполнить лабораторную работу «Разработка простого калькулятора на языке C#».

## **1. Система контроля версий TFS**

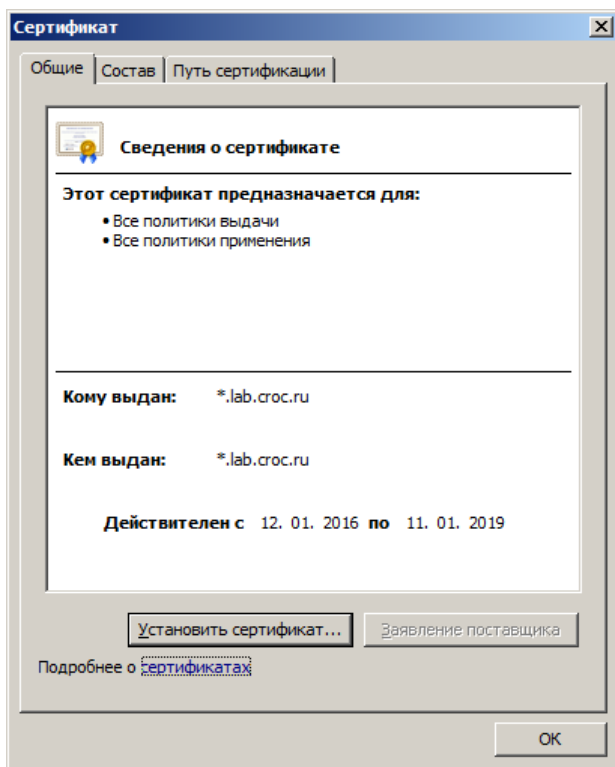
### **1.1. Общие сведения**

При выполнении лабораторных работ по курсу «Разработка приложений на языке C#» используется система контроля версий исходных текстов программ Microsoft Team Foundation Server (TFS).

Система контроля версий позволяет выполнять сравнение различных вариантов (версий) исходных текстов, ведет историю изменения всех файлов и поддерживает совместную работу над текстами программ. Настройка подключения к TFS осуществляется в два этапа – подключение к portalу TFS и настройка среды разработки Visual Studio.

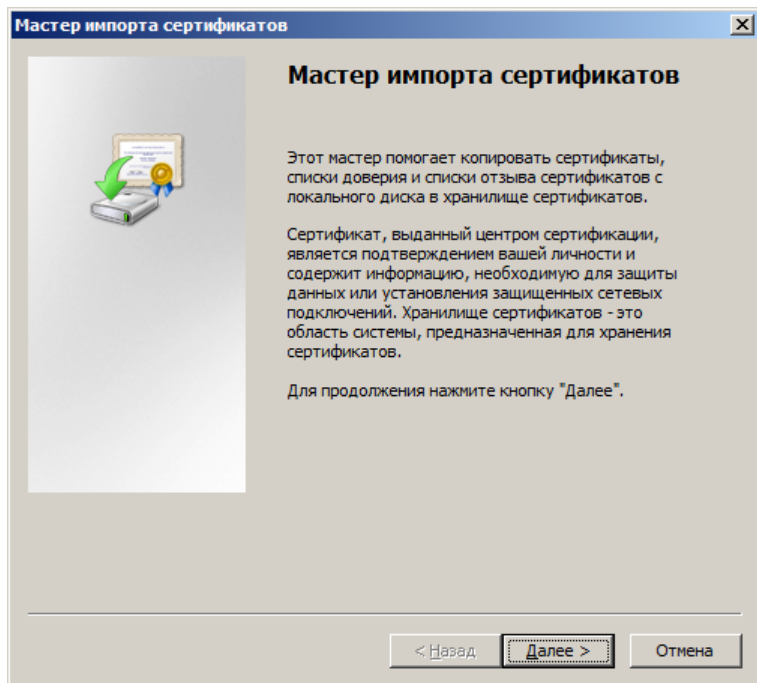
## **1.2. Подключение к portalу**

Внешнее подключение к TFS осуществляется по протоколу HTTPS. Для настройки соединения следует установить сертификат сервера (файл **lab.croc.ru.cer**). Его следует загрузить из рабочей группы Jive и сохранить локально на компьютере. После этого его следует запустить. Отобразится окно просмотра сертификата (Рисунок 1).



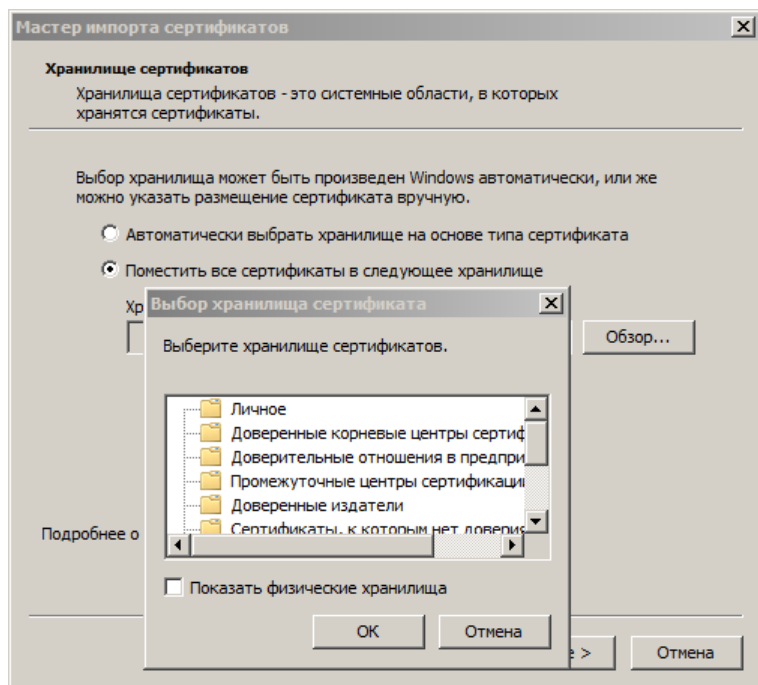
**Рисунок 1. Просмотр сертификата**

Следует нажать кнопку «Установить сертификат». В первом окне мастера нажать кнопку «Далее» (Рисунок 2).



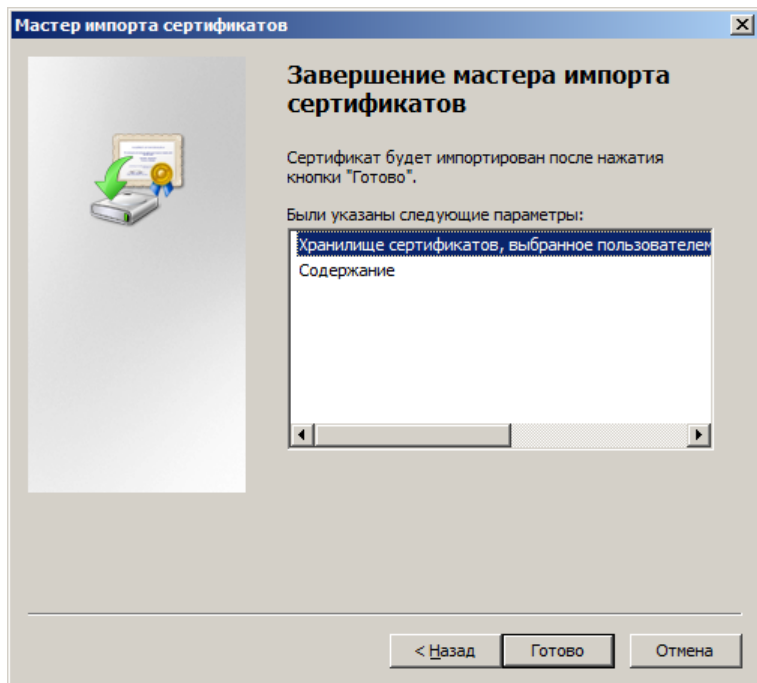
**Рисунок 2. Начало процесса импорта сертификата**

В появившемся окне выбрать пункт «Поместить все сертификаты в следующее хранилище», нажать кнопку «Обзор» и выбрать раздел «Доверенные корневые центры сертификации» (Рисунок 3).



**Рисунок 3. Выбор места установки сертификата**

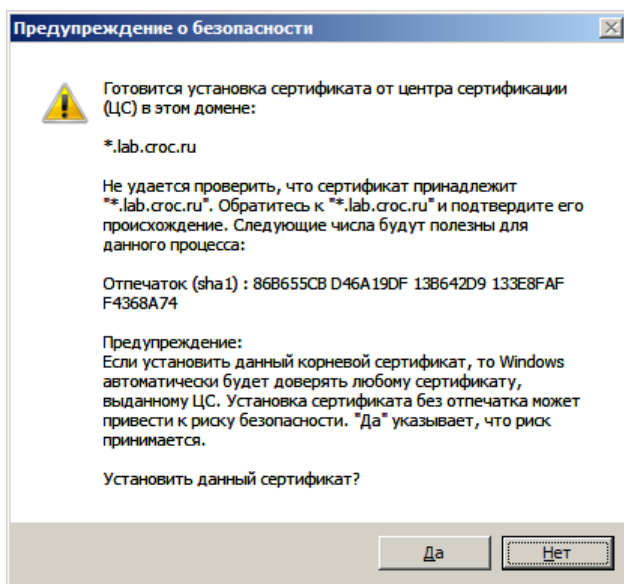
В следующем окне следует нажать кнопку «Готово» для завершения импорта (Рисунок 4).



**Рисунок 4. Завершение импорта**

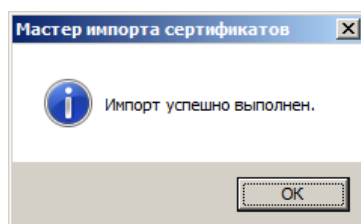
Так как речь идет о корневом сертификате, выводится дополнительное предупреждение безопасности (Рисунок 5).





**Рисунок 5. Предупреждение о корневом сертификате**

После успешного импорта сертификата выводится сообщение (Рисунок 6).



**Рисунок 6. Успешное завершение импорта**

Для проверки корректности импорта следует открыть портал TFS в браузере по адресу, предоставленному преподавателем, например <https://csharp.lab.croc.ru/tfs>. Если при открытии страницы не выводится предупреждений безопасности, значит, импорт сертификата осуществлен корректно (Рисунок 7).

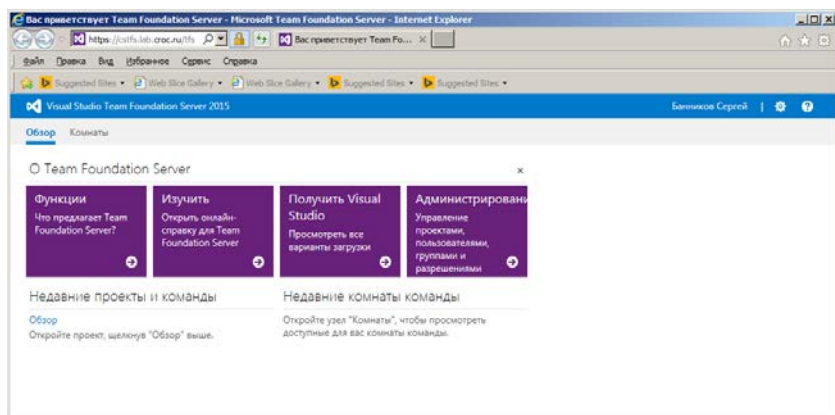



Рисунок 7. Портал Team Foundation Server

### 1.3. Настройка среды разработки

Порядок подключения к репозиторию исходных текстов на базе Microsoft Team Foundation Server (TFS) следующий:

1. Создать новый пустой каталог, в котором будут размещаться исходные тексты из репозитория TFS. Для определенности пусть это будет каталог C:\CS.
2. Запустить Microsoft Visual Studio 2015
3. Перейти в окно Team Explorer (главное меню View | Team Explorer)
4. Нажать значок  для перехода в режим управления соединениями (Manage Connections)
5. Выбрать пункт Manage Connections | Connect to Team Project (Рисунок 8)

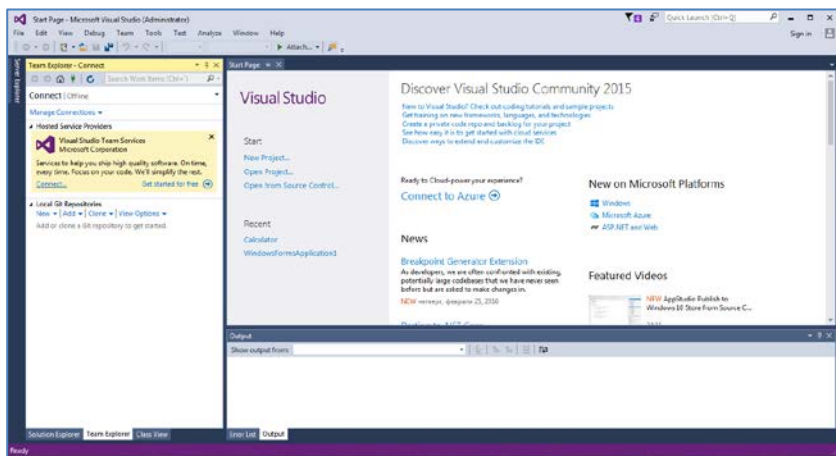
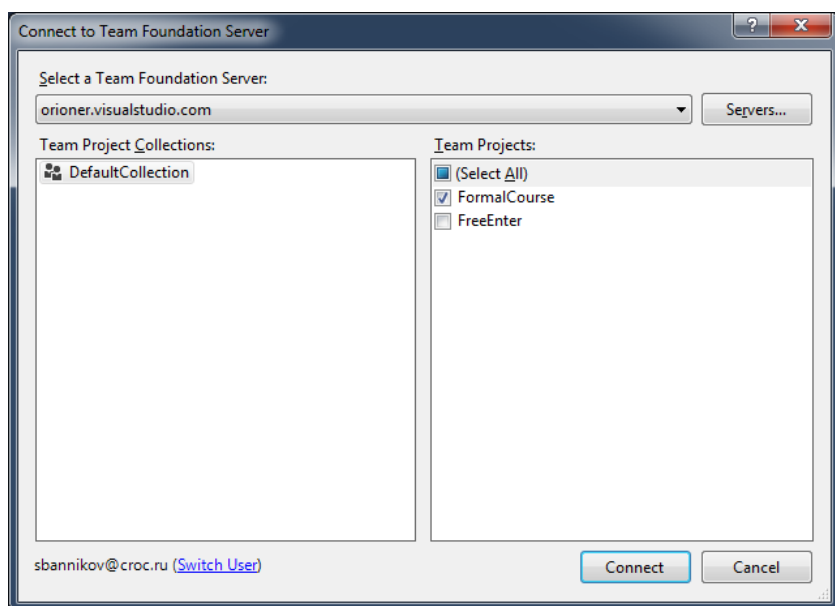


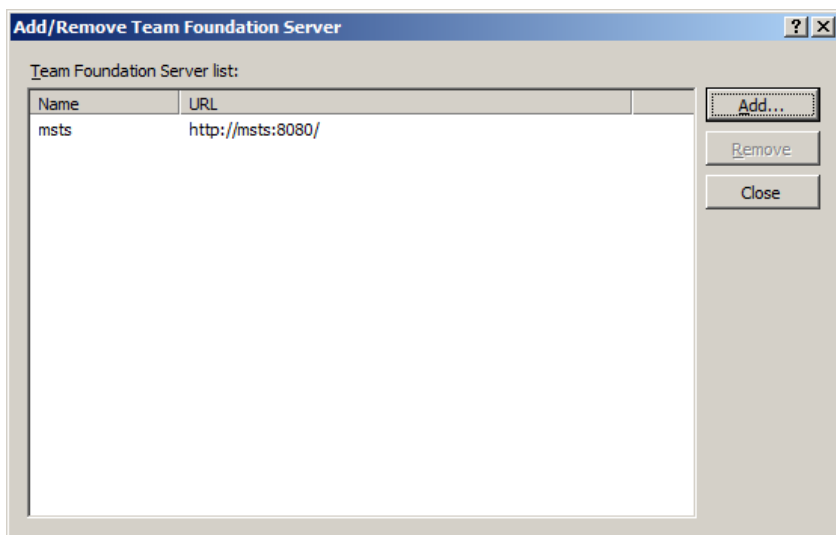
Рисунок 8. Панель Team Explorer

6. В окне Connect to Team Foundation Server нажать кнопку [Servers...] (Рисунок 9)



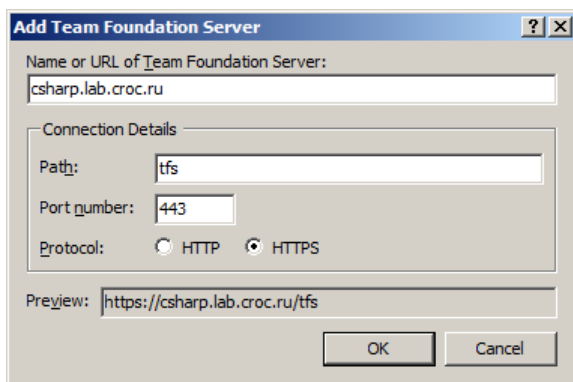
**Рисунок 9. Соединение с сервером**

7. В окне Add/Remove Team Foundation Server нажать кнопку [Add...] (Рисунок 10)



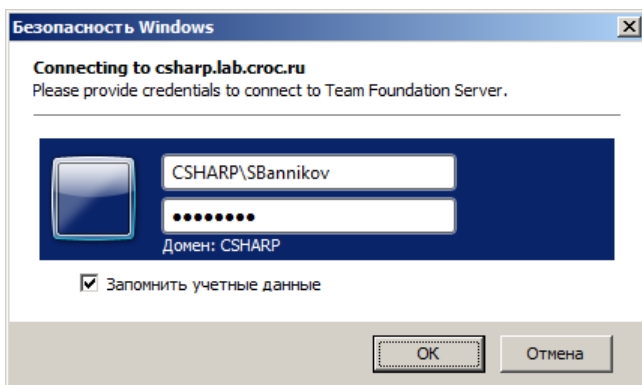
**Рисунок 10. Список доступных серверов**

8. В окне Add Team Foundation Server ввести имя сервера, выбрать режим подключения (HTTP или HTTPS) и нажать кнопку OK (Рисунок 11). Имя сервера (параметр Name or URL of Team Foundation Server) и режим подключения (Protocol) следует уточнить у преподавателя.



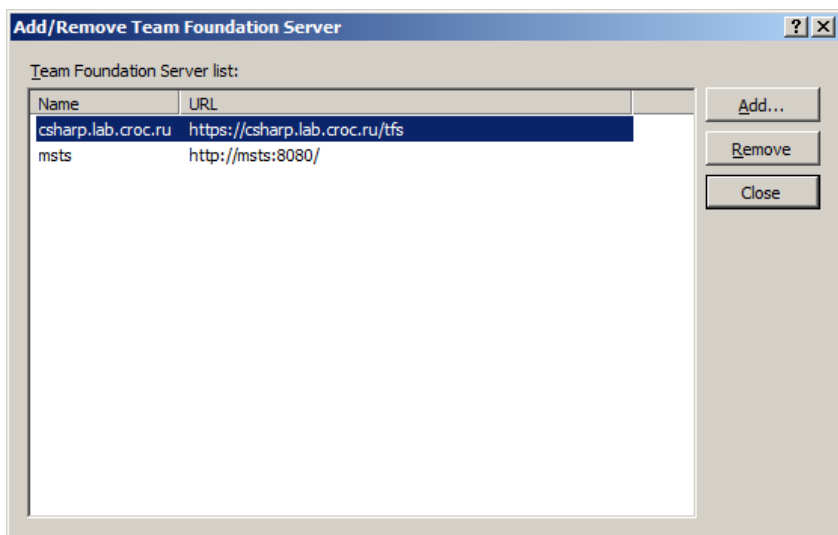
**Рисунок 11. Параметры подключения к TFS**

9. Далее выводится запрос безопасности Windows (Рисунок 12). Необходимо ввести данные учетной записи в виде SERVER\USERNAME (где SERVER – краткое имя сервера TFS, в данном случае – CSHARP; USERNAME – имя учетной записи студента) и пароль учетной записи, после чего нажать кнопку ОК. Имя учетной записи студента и пароль учетной записи следует получить у преподавателя.



**Рисунок 12. Запрос учетных данных**

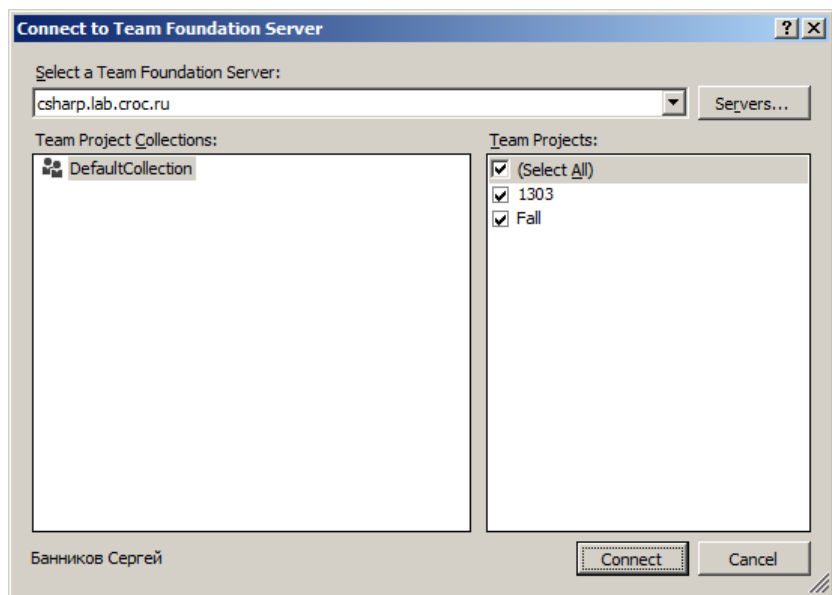
10. Сервер будет добавлен в список серверов. Для того чтобы закрыть окно Add/Remove Team Foundation Server, необходимо нажать кнопку Close (Рисунок 13).



**Рисунок 13. Список доступных серверов с добавленным сервером**

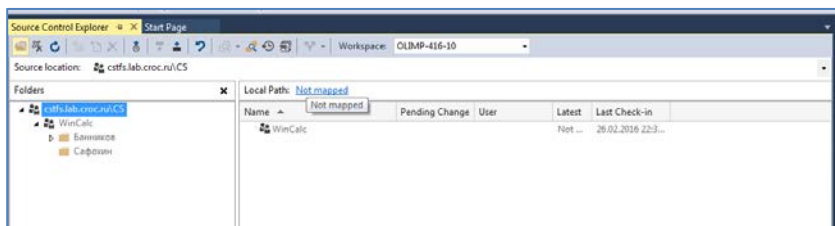
11. В окне Connect to Team Foundation Server в поле Select a Team Foundation Server следует выбрать вновь добавленный сервер. Выбрать доступную коллекцию проектов (Team Project Collections) – обычно это Default Collection. Выбрать все доступные проекты (Select All). Нажать кнопку Connect (Рисунок 14).





**Рисунок 14. Установление соединения с сервером**

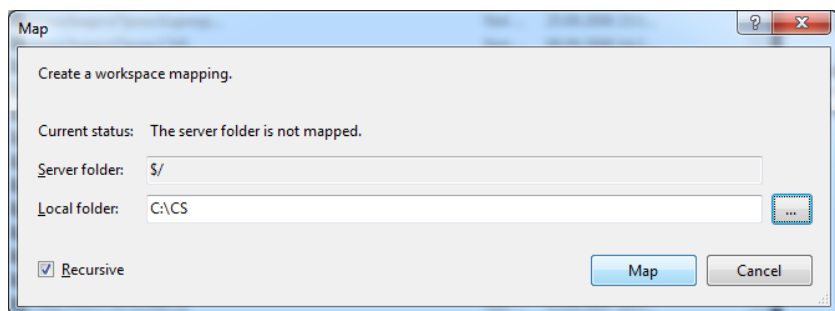
12. Выбрать окно управления исходными текстами. Для этого используется пункт главного меню View | Other Windows | Source Control Explorer или кнопка Source Control Explorer в окне Team Explorer (Рисунок 15).



**Рисунок 15. Панель Source Control Explorer**

13. В левой панели Folders выбрать корневой узел. В правой панели нажать на ссылку Not Mapped (Рисунок 15).

14. В окне Map выбрать локальный каталог, созданный выше. Нажать кнопку Map (Рисунок 16). В указанный каталог будет осуществлена загрузка всех исходных текстов.



**Рисунок 16. Отображение репозитория на локальную папку**

15. Для открытия проекта (project) или решения (solution) следует выбрать соответствующий файл в окне Source Control Explorer при помощи двойного щелчка мыши (Рисунок 17).

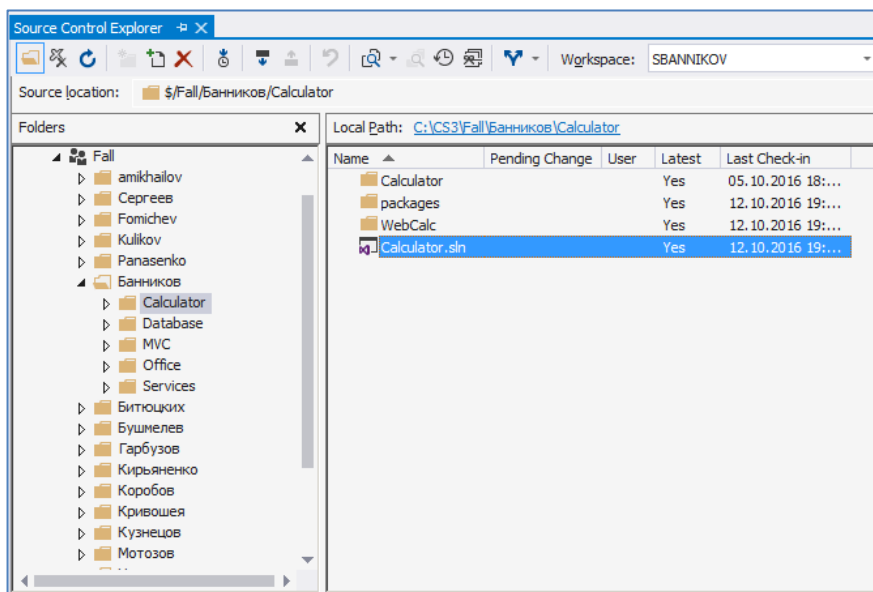


Рисунок 17. Окно Source Control Explorer

## 1.4. Учетные данные при подключении к TFS

При подключении к TFS учетные данные (имя пользователя и пароль) сохраняются в диспетчере учетных записей. Если необходимо подключиться к репозитарию от имени другого пользователя (например, при работе в классе), то необходимо удалить сохраненную учетную запись из диспетчера учетных записей. Для этого необходимо открыть панель управления. Далее следует выбрать пункт «Учетные записи пользователей» (Рисунок 18).

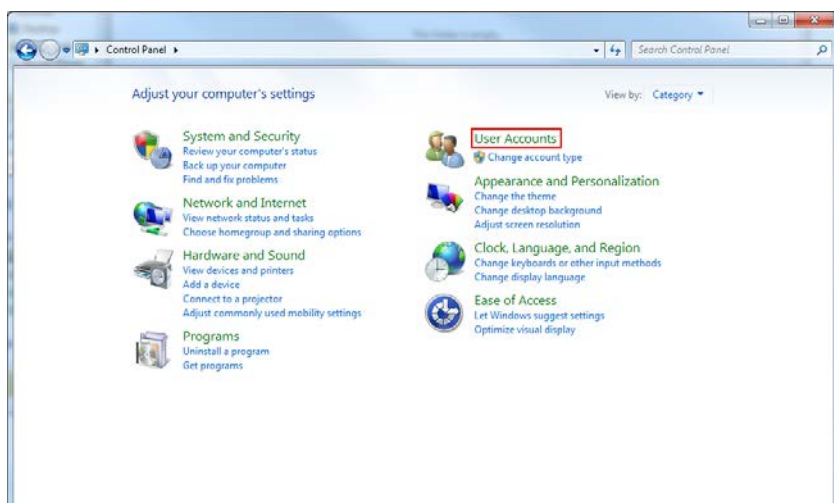
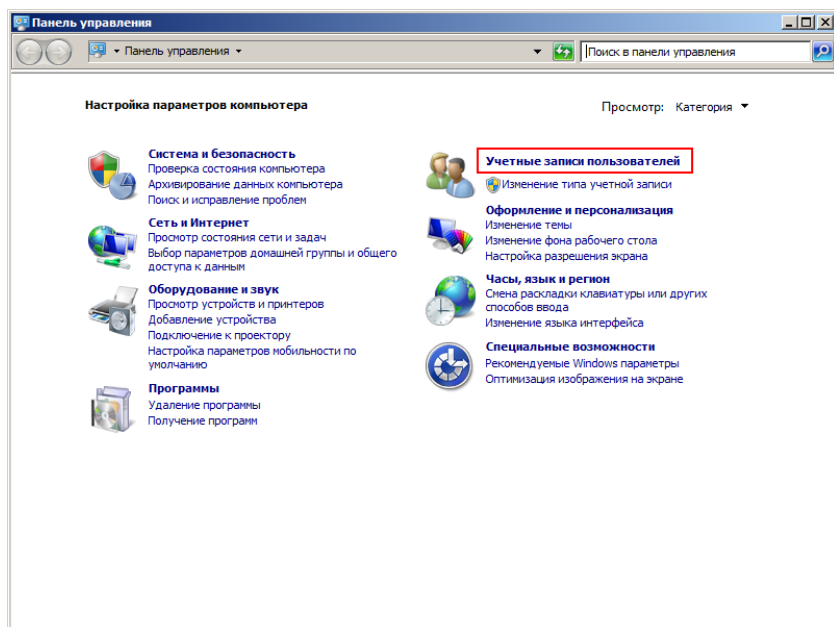
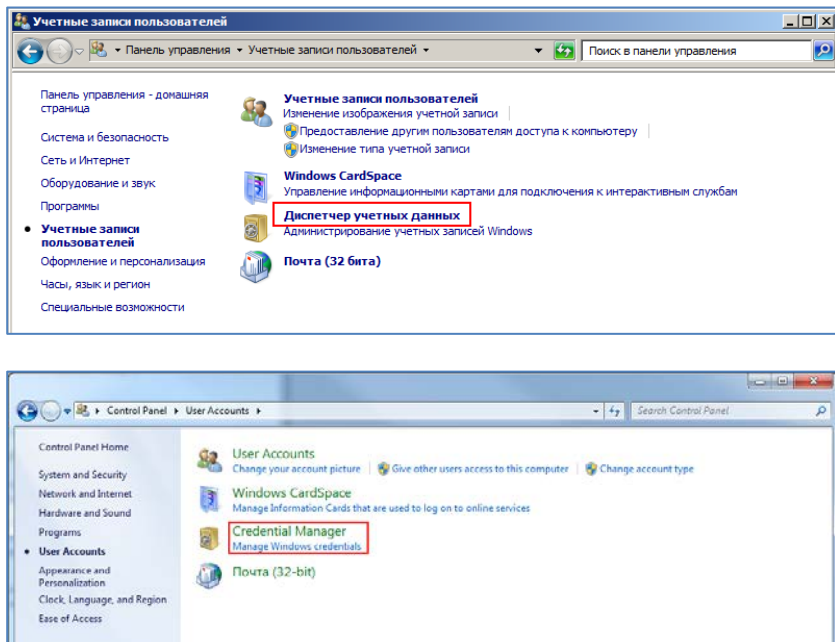


Рисунок 18. Панель управления

В окне учетных записей пользователей следует выбрать пункт «Диспетчер учетных данных» (Рисунок 19).



**Рисунок 19. Учетные записи пользователей**

В окне диспетчера учетных данных следует найти подключение к серверу TFS (csharp.lab.croc.ru). Следует обратить внимание, что таких подключений может быть несколько (Рисунок 20).

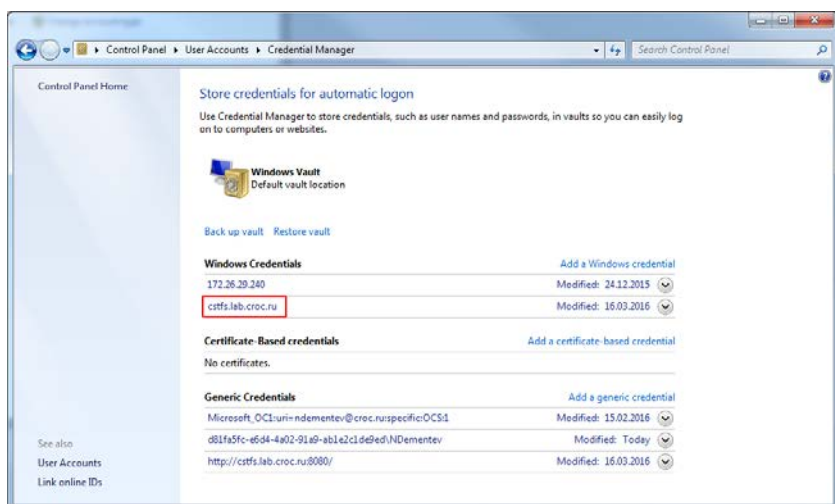
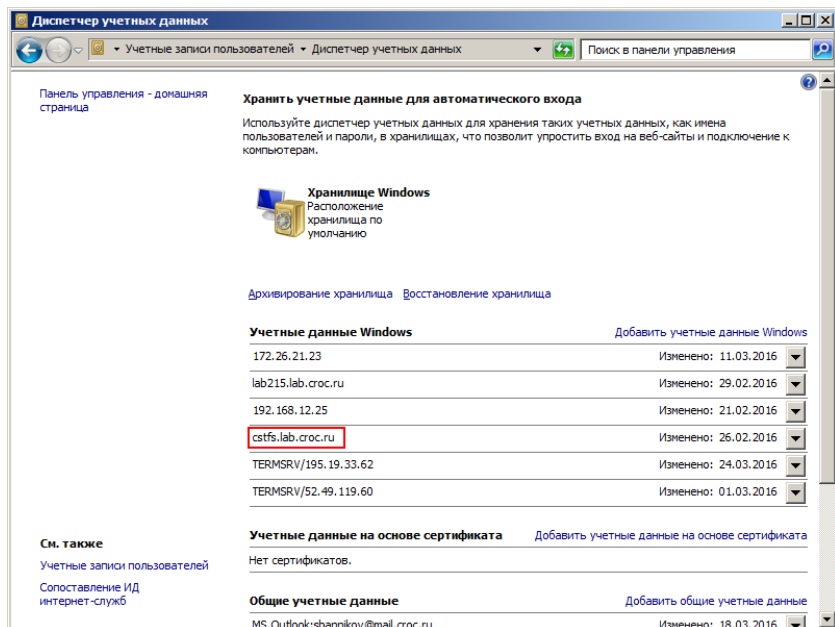


Рисунок 20. Диспетчер учетных записей

Для каждого найденного подключения следует выбрать пункт «Удаление из хранилища». В появившемся предупреждающем окне следует согласиться с удалением (Рисунок 21). Данную операцию следует повторить для всех учетных данных, относящихся к репозиторию TFS – серверу [csharp.lab.croc.ru](http://csharp.lab.croc.ru).

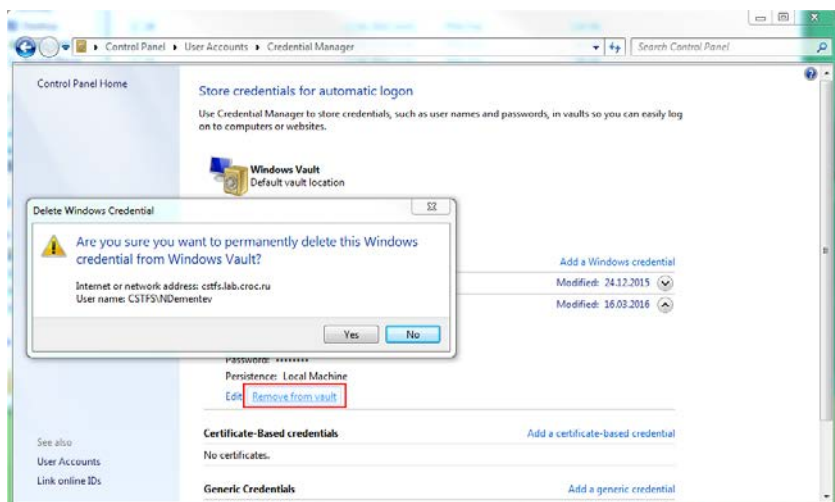
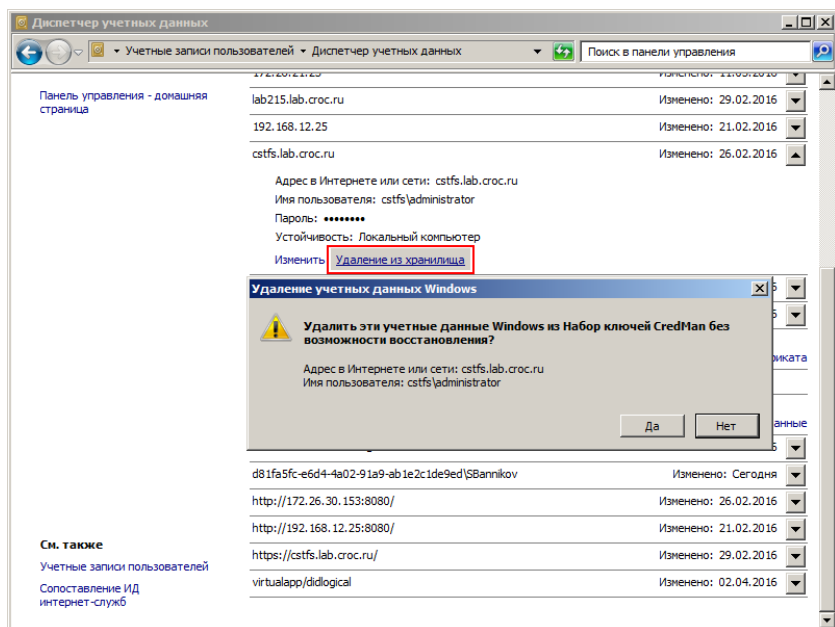
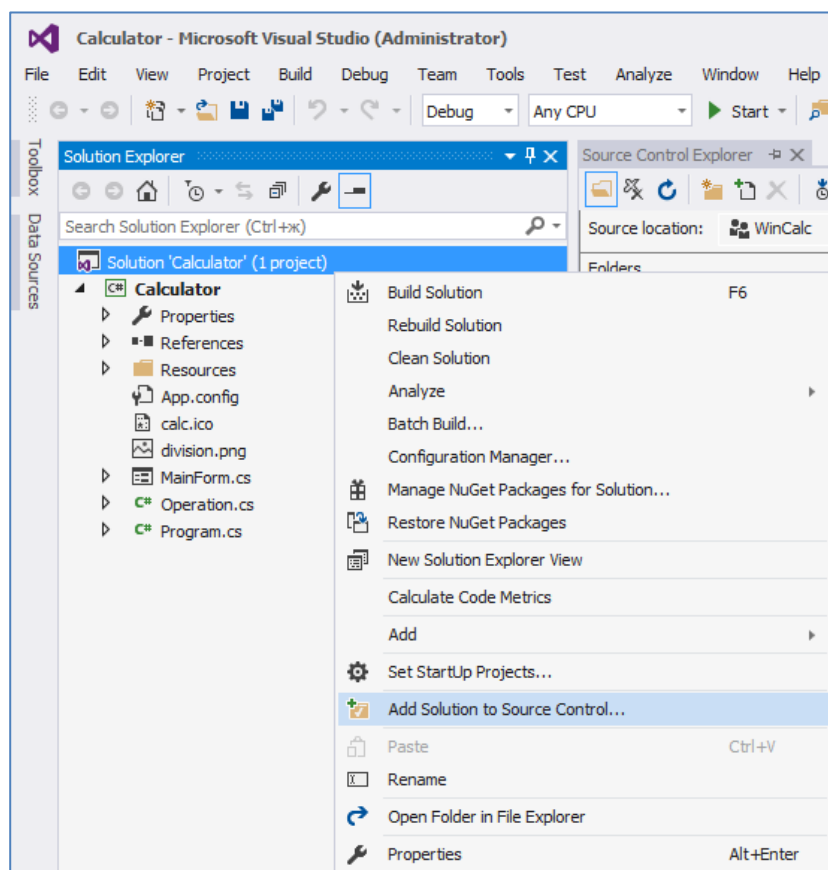


Рисунок 21. Удаление сохраненных данных

## 1.5. Использование

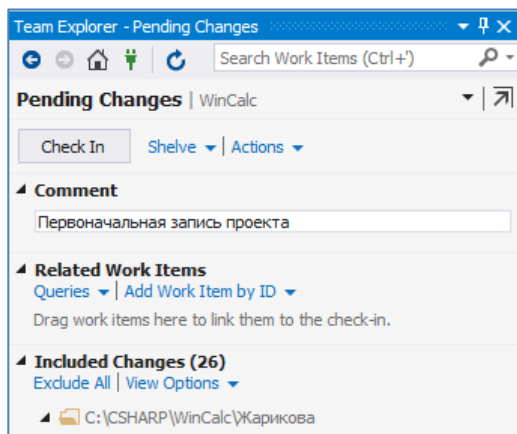


1. Для того чтобы поместить уже существующий проект в репозиторий, необходимо скопировать его в свою папку (в соответствии с фамилией студента). Далее следует в окне Solution Explorer выделить файл решения и в контекстном меню по правой клавише мыши выбрать пункт Add Solution To Source Control (Рисунок 22).



**Рисунок 22. Добавление проекта в репозиторий**

2. Для записи файлов в репозиторий следует в окне Team Explorer переключиться в режим Pending Changes (если это не произошло автоматически), заполнить поле комментария (Comment) и нажать кнопку Check In (Рисунок 23).



**Рисунок 23. Запись изменений в репозиторий**

3. При редактировании любого файла его выписка из репозитория (Check Out) происходит автоматически. После внесения изменений следует выполнить запись (Check In) так же, как и при первоначальной записи.
4. Отмена сделанных изменений и возврат к предыдущей сохраненной в репозитории версии осуществляется в окне Source Control Explorer при помощи пункта контекстного меню Undo Pending Changes.
5. Сравнение различных версий файлов осуществляется в окне Source Control Explorer при помощи пункта контекстного меню Compare.

## **1.6. Возможные проблемы при сборке проекта**

В данном разделе содержится описание типовых проблем, которые могут возникать при сборке проекта.

### **1.6.1. Системная сборка не может быть загружена**

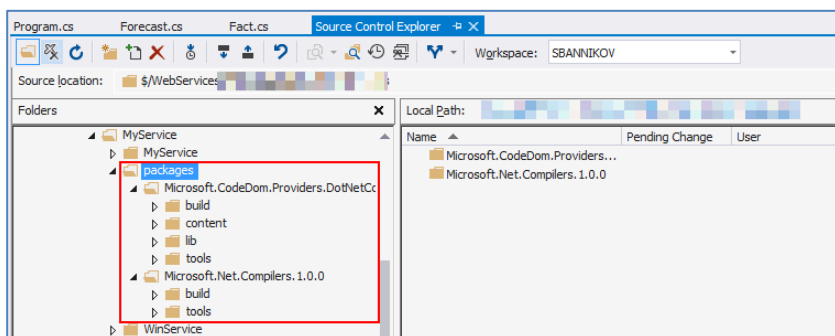
При сборке проекта выводится сообщение

The "Microsoft.CodeAnalysis.BuildTasks.Csc" task could not be loaded from the assembly

xxx\packages\Microsoft.Net.Compilers.1.0.0\build\..\tools\Microsoft.Build.Tasks.CodeAnalysis.dll. Could not load file or assembly

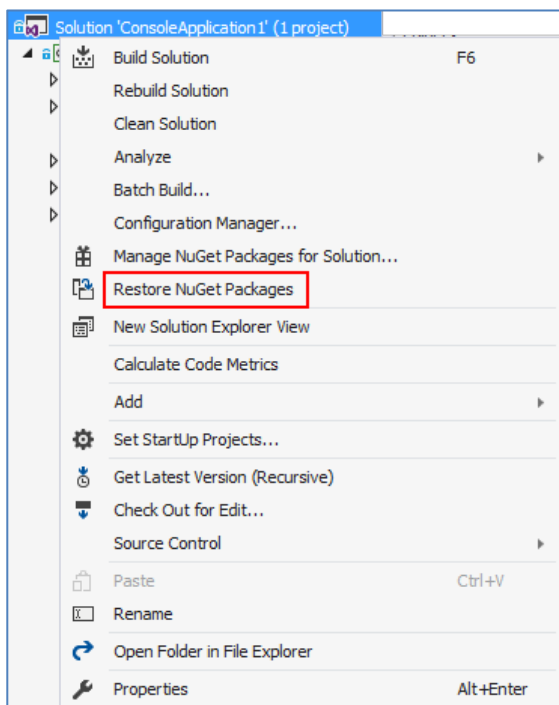
'xxx\packages\Microsoft.Net.Compilers.1.0.0\tools\Microsoft.Build.Tasks.CodeAnalysis.dll' or one of its dependencies. Не удается найти указанный файл. Confirm that the <UsingTask> declaration is correct, that the assembly and all its dependencies are available, and that the task contains a public class that implements Microsoft.Build.Framework.ITask.

Проблема в том, что в репозиторий TFS оказались записаны не только исходные коды приложения, но и пакеты NuGet, как показано ниже (Рисунок 24).



**Рисунок 24. Пакеты NuGet, записанные в репозиторий**

Пакеты NuGet загружаются при сборке проекта автоматически, и их записывать в репозиторий TFS не требуется. Если же по какой-то причине пакеты не были загружены автоматически, то при помощи команды Restore NuGet Packages в контекстном меню по правой клавише мыши на файле решения (solution) пакеты будут загружены принудительно (Рисунок 25).



**Рисунок 25. Восстановление пакетов NuGet**

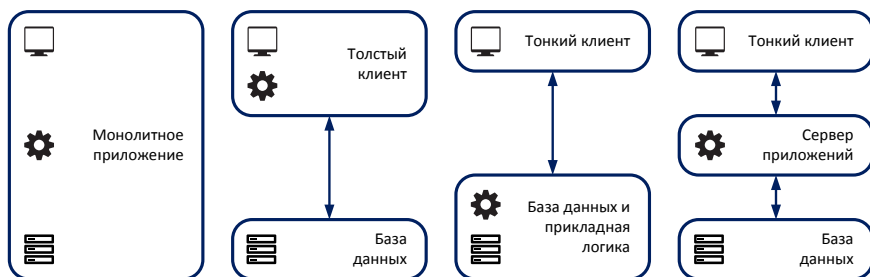
Для устранения данной ошибки следует удалить из TFS папку packages.

## **2. Разработка простого веб-приложения на базе ASP.NET Web Forms**

### **2.1. Архитектуры приложений**

Архитектура программного приложения в целом – это распределение отдельных функций приложения между программными (и соответственно, аппаратными) компонентами программной системы. Если рассматривать любую программную систему, то в ней всегда

можно выделить три основные базовые функции – хранение данных, обработка данных и визуализация данных. Типовые варианты распределения этих функций между программными компонентами приведены ниже (Рисунок 26).



**Рисунок 26. Варианты архитектур приложений**

Ранее была разработана простая калькулятор – пример монолитного приложения, в котором объединены все три функции – хранение данных (свойство  $x$  и пр.), обработка данных (обработчики событий) и визуализация данных (главная форма).

Далее предлагается перейти к разработке простого веб-приложения на базе ASP.NET. Это уже является примером разделения функций приложения. Функция визуализации реализуется при помощи веб-браузера, который обрабатывает HTML-разметку и отображает экран приложения. Хранение данных осуществляется на стороне веб-сервера, в роли которого в данном случае выступает встроенный веб-сервер операционной системы Windows – Internet Information Services (IIS). Вся обработка данных также осуществляется на стороне сервера, а на стороне клиента осуществляются только вызовы соответствующих операций на сервере.

## 2.2. Постановка задачи

Предлагается создать простое веб-приложение, реализующее часть функциональности игры «Слова из слова». Суть игры состоит в том, что требуется из букв заданного слова (обычно достаточно длинного, например «обороноспособность») составить все возможные слова. В рамках данной лабораторной работы будут реализованы следующие функциональные блоки данной игры:

- Формирование на основе заданного слова набора элементов управления – кнопок
- Обработка нажатия на кнопки и формирование слова
- Проверка слова на наличие в словаре (в качестве словаря будет использоваться текстовый файл).
- Если слово присутствует в словаре, оно вносится в список найденных слов, и пользователь может ввести новое слово.
- Для исправления ошибок набора должна быть предусмотрена кнопка для очистки набранного слова и для удаления последней введенной буквы.

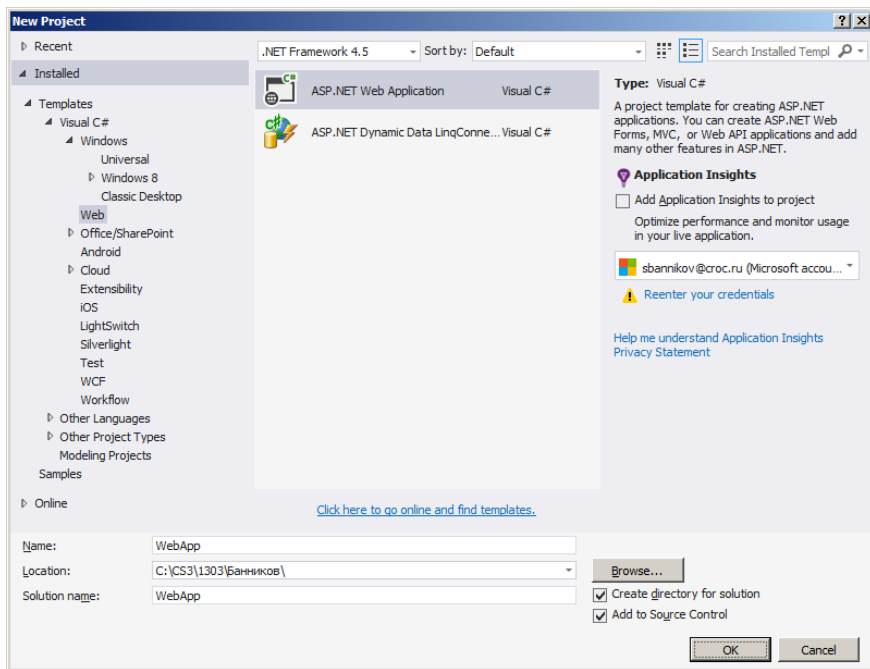
## 2.3. Создание проекта

Создание любого приложения в интегрированной среде разработки Microsoft Visual Studio начинается с создания нового проекта. Перед созданием нового проекта рекомендуется открыть окно консоли управления пакетами (Package Manager Console) при помощи пункта главного меню «Tools | NuGet Package Manager | Package Manager Console». Если при этом будет выведено предупреждение о запуске неподписанного приложения, следует выбрать режим «А» - запускать всегда.

Для создания проекта используется пункт главного меню «File | New | Project» (Рисунок 27). В появившемся окне следует выбрать тип

проекта (ASP.NET Web Applications), корневую папку для его размещения, задать имя проекта (например, WebApp) и имя решения (обычно совпадает с именем проекта, если планируется, что решение будет содержать только один проект, как в нашем случае). Для добавления проекта в систему управления исходными текстами необходимо включить режим «Add to Source Control». Для создания отдельного каталога для проекта внутри решения крайне рекомендуется включить также режим «Create directory for solution». Версию .NET Framework для данного задания можно выбрать произвольную, но рекомендуется выбирать версию не ниже 4.5. Режим «Add Application Insights to project» (сбор телеметрии о проекте при помощи облака Microsoft) следует отключить.





**Рисунок 27. Создание проекта ASP.NET**

После выбора типа проекта ASP.NET Web Application требуется задать шаблон для создаваемого приложения (Рисунок 28). Для ускорения процесса разработки рекомендуется выбрать шаблон Web Forms, при этом не следует включать размещение приложения в облаке Microsoft Azure (Host in the cloud).

Кроме того, имеет смысл отключить аутентификацию пользователей (для данного приложения она не требуется) при помощи кнопки «Change Authentication» (Рисунок 29). В появившемся окне следует выбрать режим «No Authentication».

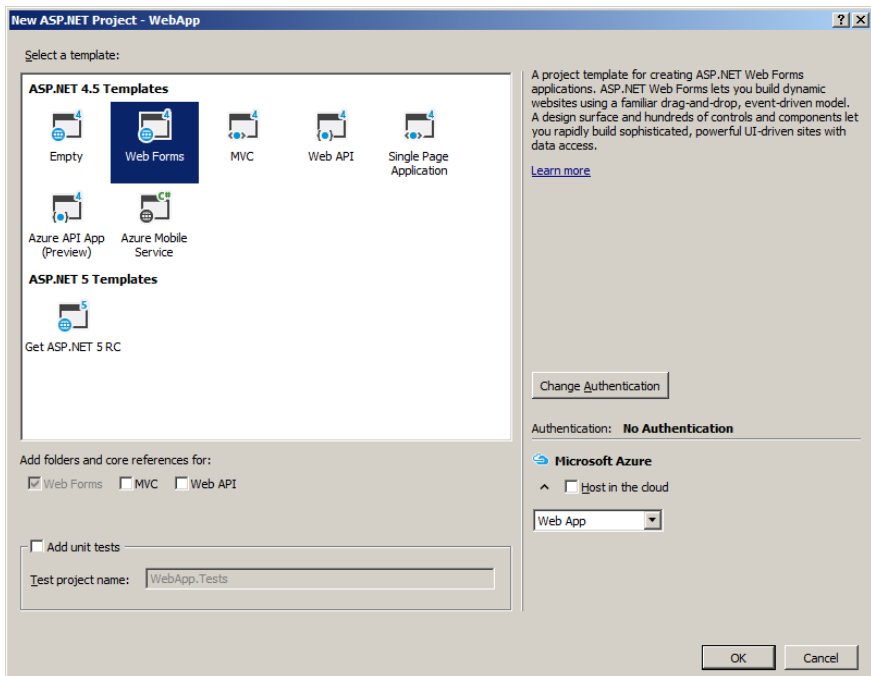


Рисунок 28. Выбор шаблона для создания проекта

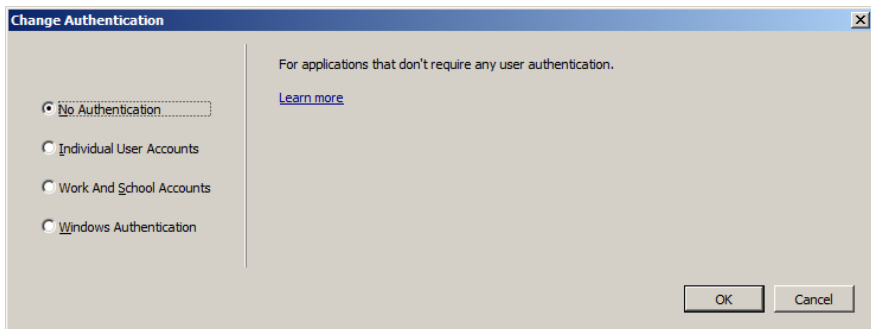


Рисунок 29. Выбор способа аутентификации пользователей в проекте

## 2.4. Запуск проекта

После завершения создания приложения имеет смысл сразу записать его в репозиторий исходных текстов ([3], раздел 2.1). Далее следует просмотреть и при необходимости изменить основные свойства проекта ([3], раздел 2.1). Как и в случае с разработкой калькулятор ([3]), сразу после создания приложение может быть запущено. Основным отличием является то, что при запуске в среде разработки обычно используется специальный веб-сервер IIS Express, а само приложение отображается в окне браузера.

Для управления порядком запуска и отладки веб-приложения в среде разработки в свойствах проекта предусмотрен отдельный раздел Web (Рисунок 30). В данном разделе можно настроить поведение приложения после запуска – переход на заданную страницу (Specific Page), на страницу, открытую в среде разработки в редакторе (Current Page) или на заданный адрес (Start URL). Следует обратить внимание, что приложение будет доступно не по стандартному TCP-порту протокола HTTP (порт 80), а по особому порту, который выбирается случайным образом из числа доступных портов.

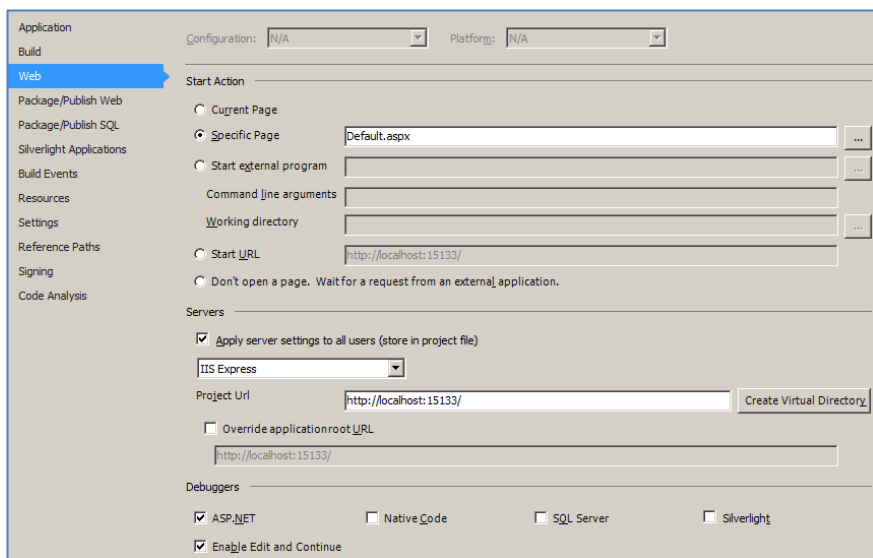
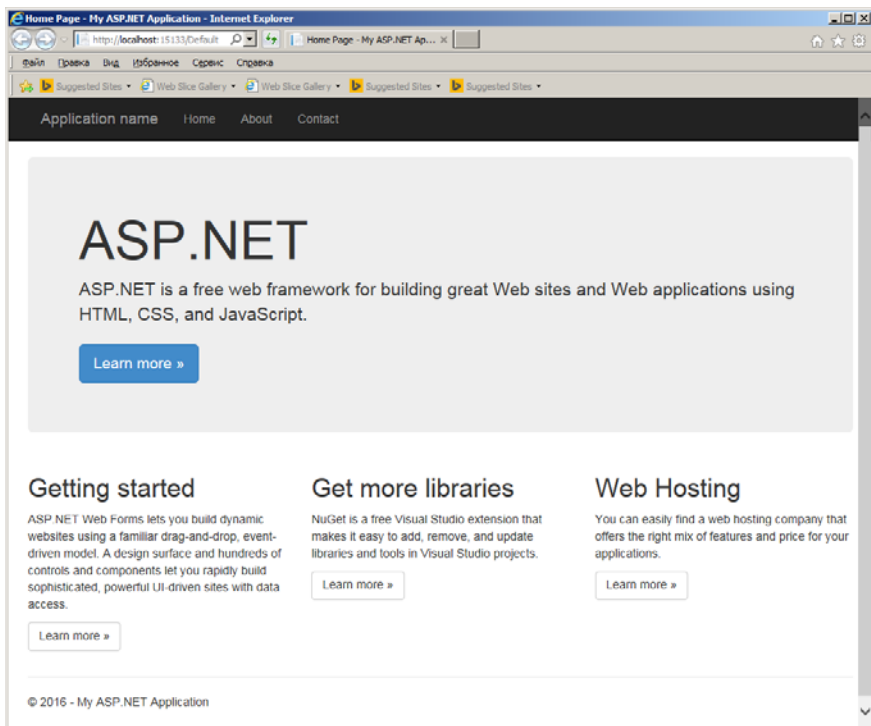


Рисунок 30. Свойства веб-приложения

## 2.5. Первичная настройка проекта

Стартовой страницей для только что созданного приложения является страница Default.aspx (Рисунок 31). Она состоит из верхнего навигационного меню, которое сейчас содержит пункты «Application name», «Home», «About» и «Contact». С каждым из этих пунктов связана страница, сформированная по шаблону проекта автоматически.



**Рисунок 31. Стартовая страница приложения по умолчанию**

Следует обратить внимание, что для веб-приложения в окне браузера используется не тот значок, который задан в свойствах приложения, а значок веб-приложения по умолчанию, который хранится в файле `favicon.ico`. Для того чтобы изменить этот значок, можно либо сохранить собственный значок под именем `favicon.ico`, либо изменить ссылку на значок так, чтобы она указывала на другой файл. Для реализации второго способа требуется внести соответствующее изменение в HTML-разметку главной страницы приложения. Главная страница приложения (или мастер-страница) – это файл, который задает основную визуальную структуру всего приложения. Все остальные страницы только заполняют определенные

зоны (place holders) главной страницы. В данном приложении главная страница называется Site.Master. Ссылка на значок приложения выглядит следующим образом:

```
<link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
```

Знак «~» указывает на то, что ссылка является относительной и показывает путь относительно корня веб-приложения. Заменяв favicon.ico на имя другого файла, можно поменять значок приложения, отображаемый в браузере. Правда, при запуске приложения в режиме отладки эта настройка может не всегда работать.

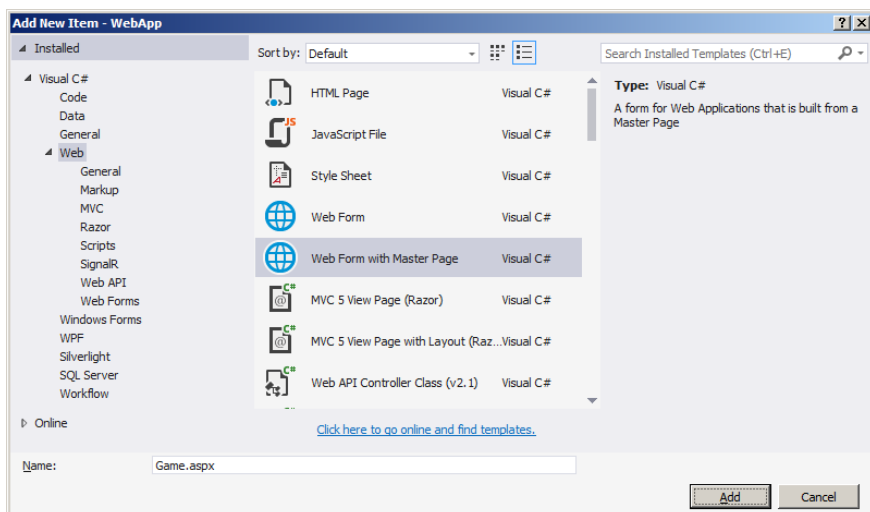
Вместо строки «Application name» следует указать название нашего приложения. Для этого в коде главной страницы следует найти следующую строку:

```
<a class="navbar-brand" runat="server" href="~/>Application name</a>
```

Фрагмент, выделенный желтым фоном, следует изменить на название нашего приложения, например «Простое приложение».

## 2.6. Создание новой страницы

Для выполнения поставленной задачи следует создать новую пустую страницу, на которой будут размещаться все необходимые элементы управления. Для этого в контекстном меню по правой клавише мыши на файле проекта выбирается пункт «Add | New Item». В разделе Web следует выбрать пункт «Web Form with Master Page» (Рисунок 32). В поле Name следует ввести название страницы, например, Game.aspx (расширение подставляется по умолчанию) и нажать кнопку Add.



**Рисунок 32. Создание новой страницы**

Далее выбирается главная страница, которая используется в качестве общего шаблона приложения. Следует выбрать страницу Site.Master (Рисунок 33) и нажать кнопку ОК.

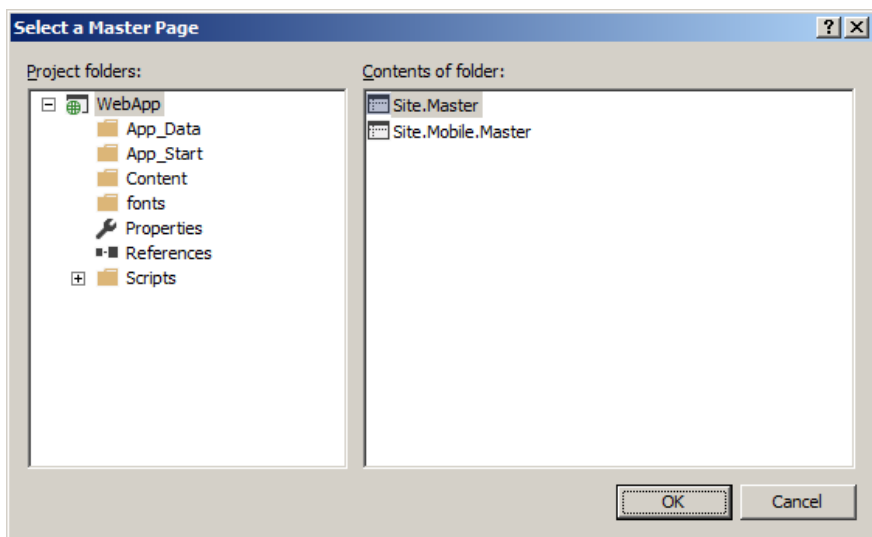


Рисунок 33. Выбор главной страницы

**Примечание.** Если бы при создании новой страницы был выбран пункт Web Form, то выбирать главную страницу не пришлось, но нам бы пришлось реализовывать необходимые общие элементы управления (в частности верхнее меню) самостоятельно.

Чтобы наша новая страница отображалась в верхнем навигационном меню приложения, его следует в него добавить путем редактирования главной страницы. В ее коде следует найти приведенный ниже блок кода и добавить туда строку, выделенную желтым.

```
<ul class="nav navbar-nav">
    <li><a runat="server" href="~/>Home</a></li>
    <li><a runat="server" href="~/Game">Игра</a></li>
    <li><a runat="server" href="~/About">About</a></li>
    <li><a runat="server" href="~/Contact">Contact</a></li>
</ul>
```

Следует обратить внимание, что ссылка на страницу указана в относительной форме относительно корня приложения (символ ~). Имя



страницы Game указано без расширения aspx – такое расширение используется по умолчанию, и указывать его в явной форме не обязательно. Текст внутри ссылки (тег <a>) может быть произвольным. После запуска приложение приобретает следующий вид (Рисунок 34).

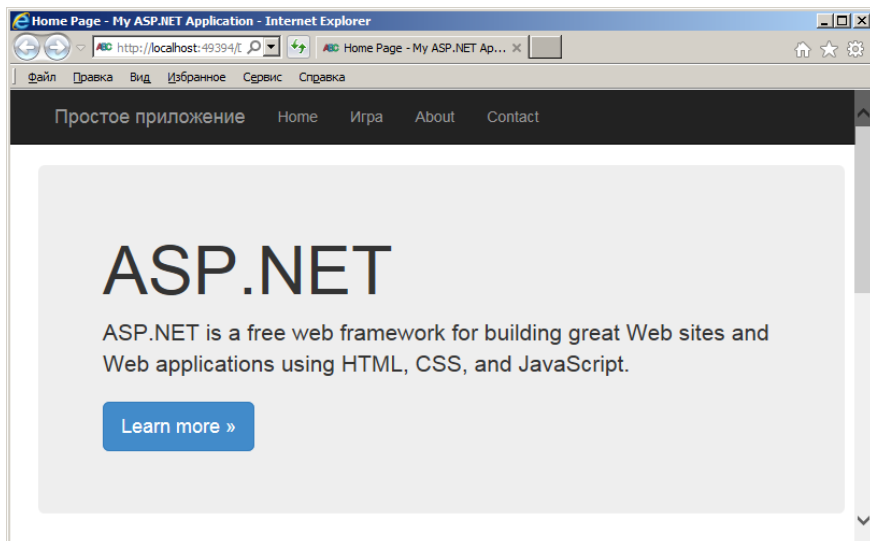


Рисунок 34. Приложение после изменения главного меню

## 2.7. Начальная разметка страницы

Перед тем как формировать исходный код нашей страницы, имеет смысл сделать эскиз интерфейса пользователя, который необходимо реализовать. Он может выглядеть, например, следующим образом (Рисунок 35). Кнопка [=] удаляет последнюю введенную букву. Кнопка [X] очищает все введенные буквы. Буквенные кнопки формируются динамически на основе исходного слова.

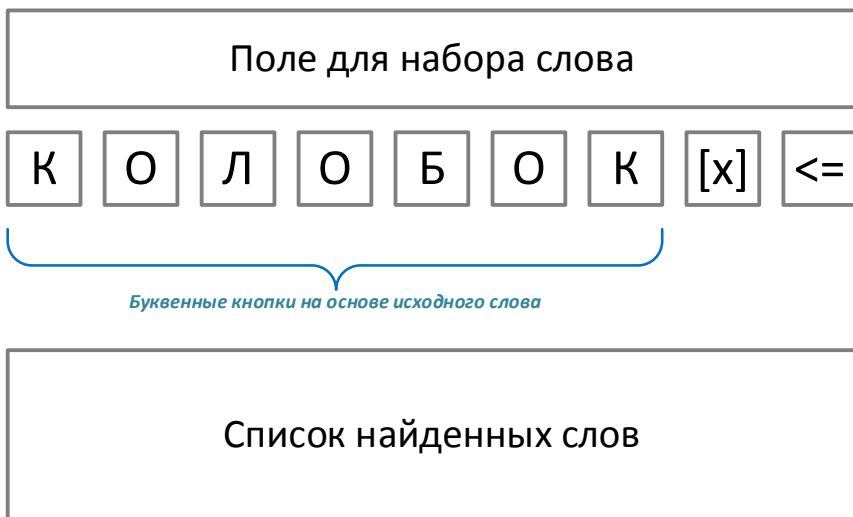


Рисунок 35. Эскиз пользовательского интерфейса

Если открыть созданную страницу Game.aspx, то по умолчанию выводится ее разметка, а именно:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="Game.aspx.cs"
Inherits="WebApp.Game" %>
<asp:Content ID="Content1" ContentPlaceHolderID="MainContent"
runat="server">
</asp:Content>
```

Разберем элементы разметки по порядку. Первая строка описывает данный файл как ASP-страницу (Page) с пустым заголовком (Title). Код страницы написан на языке C# (Language), используется главная страница Site.Master (атрибут MasterPageFile).

Параметр AutoEventWireup включает автоматическую привязку событий к элементам управления на основе шаблона имени. Так, обработчик события Load для страницы (Page) должен иметь имя Page\_Load, в этом случае не требуется явного указания на то, что

Page\_Load – обработчик события Load для страницы. Шаблон имени, таким образом, имеет вид <Объект>\_<Событие>.

Атрибут CodeBehind задает имя файла, содержащего исходный код страницы, выполняемый на стороне сервера. Атрибут Inherits определяет полное имя класса, включая пространство имен.

Далее описан элемент управления ASP.NET для размещения содержимого страницы. Вся создаваемая разметка должна содержаться внутри этого тега asp:Content. Следует обратить внимание на атрибут ContentPlaceHolderID – он задает идентификатор зоны (place holder) главной страницы. При загрузке страницы все содержимое тега asp:Content будет помещено в соответствующую зону главной страницы, заменив собой находящийся там тег asp:ContentPlaceHolder:

```
<asp:ContentPlaceHolder ID="MainContent" runat="server">  
</asp:ContentPlaceHolder>
```

Для начала предлагается заполнить название страницы (атрибут Title). Заполним его, например, следующим образом (выделено зеленым)

```
<%@ Page Title="Слова из слова" Language="C#"   
MasterPageFile="~/Site.Master" AutoEventWireup="true"   
CodeBehind="Game.aspx.cs" Inherits="WebApp.Game" %>
```

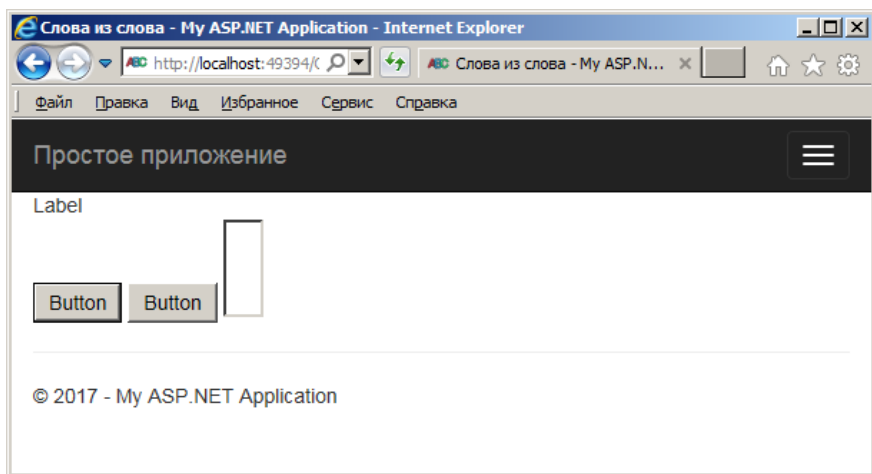
После этого необходимо разместить требуемые элементы управления внутри тега asp:Content. Можно просто набрать соответствующие теги вручную, а можно воспользоваться помощью набора инструментов (Toolbox) и вытащить необходимые элементы из него. В качестве поля для набора слов используем компонент Label. Для динамического размещения кнопок предусмотрит отдельную панель Panel. Далее разместим две кнопки Button и, наконец, список для найденных слов ListBox. В результате получится следующий код:

```

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent"
runat="server">
    <asp:Label ID="Label1" runat="server" Text="Label">
    </asp:Label>
    <asp:Panel ID="Panel1" runat="server"></asp:Panel>
    <asp:Button ID="Button1" runat="server" Text="Button" />
    <asp:Button ID="Button2" runat="server" Text="Button" />
    <asp:ListBox ID="ListBox1" runat="server"></asp:ListBox>
</asp:Content>

```

Если запустить сейчас наше приложение, то страница Game будет выглядеть следующим образом (Рисунок 36). Для удобства отладки можно указать страницу Game как стартовую при отладке приложения (Рисунок 30).



**Рисунок 36. Страница с элементами управления**

Понятно, что в таком виде страница совершенно не похожа на первоначальный замысел. Для настройки свойств элементов управления имеет смысл переключиться в режим дизайна (Design), как показано ниже (Рисунок 37). Кнопка перехода в режим дизайна находится в левом нижнем углу панели исходного кода, рядом с ней находятся две другие кнопки: Source для просмотра исходного кода

(в этом режиме окно находится сейчас) и Split для совместного просмотра кода и его графического представления.

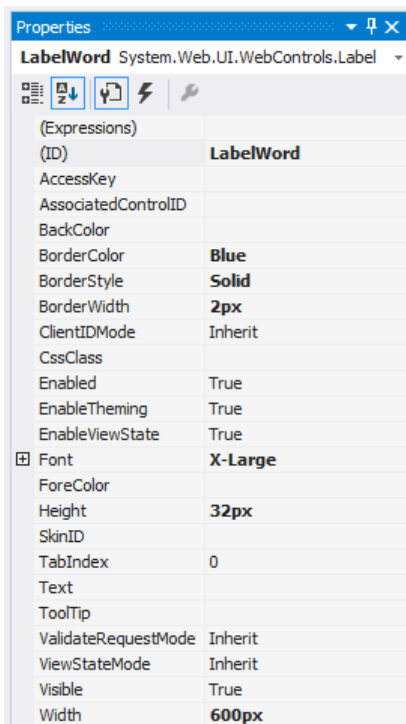


**Рисунок 37. Переключение в режим дизайна**

В режиме дизайна следует выделить метку (Label) и задать следующие ее свойства (Таблица 1, Рисунок 38). Цвета и размеры можно задавать в соответствии с собственными представлениями о графическом дизайне.

**Таблица 1. Заполняемые свойства метки**

Свойства	Значение	Примечание
ID	LabelWord	Идентификатор метки
BorderColor	Blue	Цвет рамки
BorderStyle	Solid	Стиль рамки (сплошная линия)
BorderWidth	2px	Ширина рамки
Font-Size	X-Large	Размер шрифта
Height	32px	Высота в пикселях
Text	значение не задано	Текст метки (будет формироваться динамически)
Width	600px	Ширина в пикселях



**Рисунок 38. Свойства метки в дизайнера**

Аналогичным образом заполняются свойства кнопки для очистки одной буквы (Таблица 2), кнопки для очистки всего введенного слова (Таблица 3) и для списка слов (Таблица 4).

**Таблица 2. Свойства кнопки для очистки одной буквы**

Свойства	Значение	Примечание
ID	ButtonBack	Идентификатор кнопки
BackColor	Yellow	Цвет фона кнопки
Font-Size	Large	Размер шрифта
Height	48px	Высота в пикселях
Text	[<=]	Текст кнопки, в HTML-коде символ «<» будет заменен на последовательность &lt;;

Свойства	Значение	Примечание
Width	48px	Ширина в пикселях

Таблица 3. Свойства кнопки для очистки всего слова

Свойства	Значение	Примечание
ID	ButtonClear	Идентификатор кнопки
BackColor	Red	Цвет фона кнопки
Font-Size	Large	Размер шрифта
Height	48px	Высота в пикселях
Text	[X]	Текст кнопки
Width	48px	Ширина в пикселях

Таблица 4. Свойства списка слов

Свойства	Значение	Примечание
ID	ListWords	Идентификатор списка
Width	600px	Ширина в пикселях

Для созданной панели необходимо также поменять идентификатор (назовем ее ButtonPanel). В результате код содержимого формы будет выглядеть следующим образом:

```
<asp:Label ID="LabelWord" runat="server" BorderColor="Blue"
BorderStyle="Solid" BorderWidth="2px" Font-Size="X-Large"
Height="32px" Width="600px"></asp:Label>
<asp:Panel ID="ButtonPanel" runat="server"></asp:Panel>
<asp:Button ID="ButtonBack" runat="server" Text="[&lt;=]"
BackColor="Yellow" Font-Size="Large" Height="48px" Width="48px"
/>
<asp:Button ID="ButtonClear" runat="server" Text="[X]"
BackColor="Red" Font-Size="Large" Height="48px" Width="48px" />
<asp:ListBox ID="ListWords" runat="server" Width="600px">
</asp:ListBox>
```

## 2.8. Динамическая генерация кнопок

Перейдем к динамической генерации кнопок внутри панели. Для этого необходимо открыть код страницы Game – файл Game.aspx.cs. Сейчас он выглядит следующим образом (Рисунок 39):

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApp
{
    public partial class Game : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }
    }
}

```

**Рисунок 39. Первоначальный код страницы**

Веб-страница с точки зрения языка программирования C# является объектом класса-наследника класса `System.Web.UI.Page`. В данном случае класс-наследник называется `Game`. В нем описан обработчик события загрузки страницы. В рамках данного обработчика у нас имеется возможность динамически создавать новые элементы управления, чем мы сейчас и займемся. Для начала следует описать константу, которая будет содержать исходное слово, из букв которого будут составляться все иные слова:

```
private const string word = "ПРОТОПЛАЗМА";
```

Далее внутри обработчика события `Page_Load` опишем цикл для динамического создания кнопок внутри панели `ButtonPanel` (Рисунок 40).

```

// Перебор всех символов слова
for (int i = 0; i < word.Length; i++)
{
    // Создание новой кнопки и заполнение ее свойств
    Button b = new Button()

```



```

{
    // Текст кнопки - очередная буква слова
    Text = word[i].ToString(),
    // Кнопки нумеруются по порядку, начиная с 0
    ID = "button" + i,
    // Размеры кнопки задаются в пикселях
    Width = new Unit("48px"),
    Height = new Unit("48px")
};
// Добавление кнопки в панель
ButtonPanel.Controls.Add(b);
}

```

**Рисунок 40. Цикл для динамического создания кнопок**

Данный цикл последовательно перебирает все буквы слова word. Для извлечения очередной буквы используется синтаксис обращения к строке как к элементу массива (так называемый индекатор). При этом элементами массива являются символы – объекты типа Char. Для преобразования их в строку используется метод ToString(). Кнопка (переменная b) создается при помощи конструктора без параметров и инициализируется при помощи инициализатора. Инициализатор – это синтаксис, который позволяет присвоить необходимые значения полям и свойствам объекта сразу после его создания. Функционально этот фрагмент кода эквивалентен следующему:

```

Button b = new Button();
b.Text = word[i].ToString();
b.ID = "button" + i;
b.Width = new Unit("48px");
b.Height = new Unit("48px");

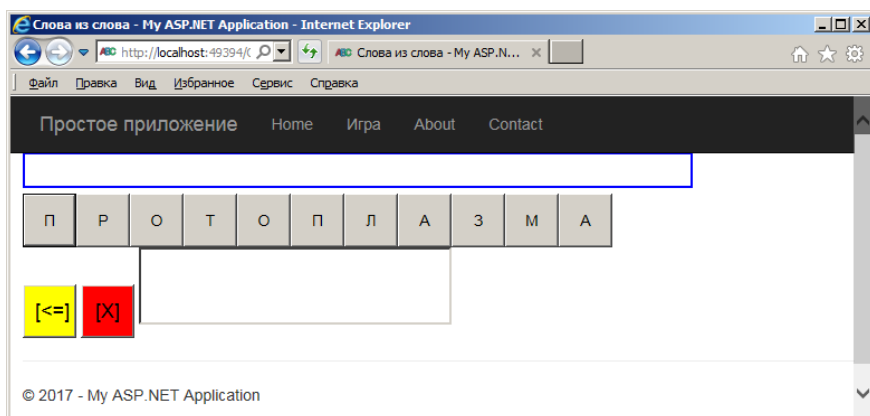
```

Следует обратить внимание, что свойство ID является строковым, при этом в выражении, стоящем справа от оператора присваивания, второй аргумент является целым числом. В этом случае выполняется неявное преобразование типа – целое число преобразуется в строковое представление при помощи неявного вызова метода ToString().

Ширина и высота могут быть заданы целыми числами, но для задания их в определенных единицах измерения (в данном случае в пикселях) используется вспомогательный класс Unit.

В конце тела цикла осуществляется добавление кнопки в коллекцию дочерних элементов управления, размещаемых внутри панели, которая выступает как контейнер.

После всех этих изменений приложение выглядит следующим образом (Рисунок 41). Пора вернуться к дизайну.



**Рисунок 41. Динамическая генерация кнопок**

## **2.9. Доработка дизайна приложения**

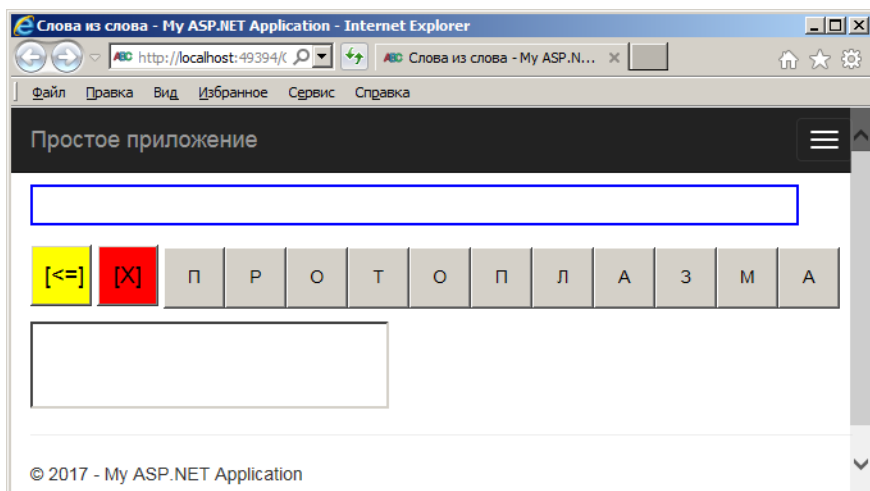
Настало время уделить внимание дизайну. До сих пор компоненты размещались на странице последовательно, но это не совсем верно. Так как код страницы – это на самом деле HTML с добавлением специфичных для ASP.NET тегов (префикс asp), то для получения требуемого результата необходимо применять стандартные для HTML методы дизайна. Обычно используются таблицы (тег <table>), но в нашем случае можно попробовать обойтись параграфами (тег <p>).

Если заключить каждую горизонтальную группу элементов в отдельный параграф, то в результате получается следующий код (Рисунок 42):

```
<p />
<p>
  <asp:Label ID="LabelWord" runat="server" BorderColor="Blue"
    BorderStyle="Solid" BorderWidth="2px" Font-Size="X-Large"
    Height="32px" Width="600px"></asp:Label>
</p>
<p>
  <asp:Panel ID="ButtonPanel" runat="server">
    <asp:Button ID="ButtonBack" runat="server" Text="["&lt;="]
      BackColor="Yellow" Font-Size="Large" Height="48px"
      Width="48px" />
    <asp:Button ID="ButtonClear" runat="server" Text="X]"
      BackColor="Red" Font-Size="Large" Height="48px" Width="48px"
      />
  </asp:Panel>
</p>
<p>
  <asp:ListBox ID="ListWords" runat="server" Width="600px">
  </asp:ListBox>
</p>
```

**Рисунок 42. Дизайн с использованием параграфов**

После запуска приложение приобретает следующий вид (Рисунок 43).



**Рисунок 43. Внешний вид доработанного дизайна**

Разумно было бы расположить управляющие кнопки не слева от буквенных, как это получилось сейчас, а после них, справа. Для этого требуется заменить одну строчку кода формы (Рисунок 40, выделено желтым). Вместо использования метода `Add`, который добавляет кнопку в конец панели, следует использовать метод `AttAt`, который добавит ее перед элементом с заданным номером (нумерация осуществляется с 0):

```
ButtonPanel1.Controls.Add(i, b);
```

После такой модификации приложение примет вид (). Пора заняться обработчиками нажатия на кнопки.

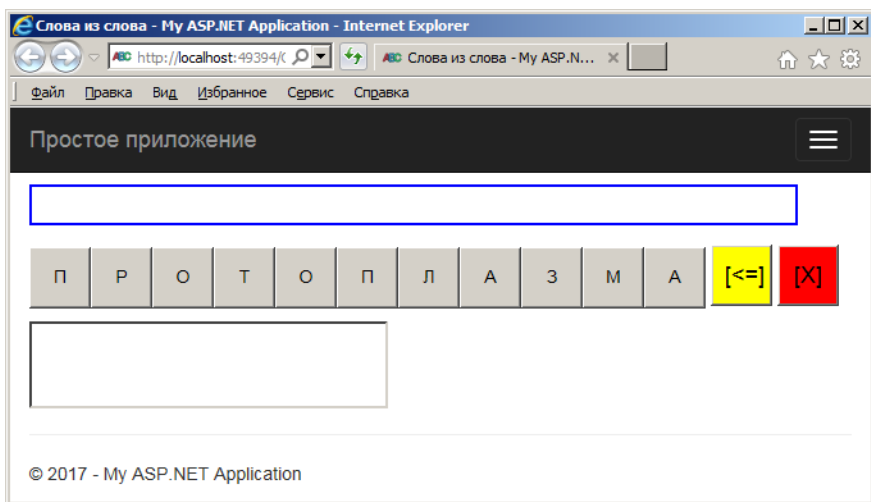


Рисунок 44. Изменение порядка следования управляющих кнопок

## 2.10. Обработчики кнопок

Для создания обработчиков управляющих кнопок можно применить уже известный способ ([3], раздел 2.6). Двойной щелчок мыши в режиме дизайна страницы автоматически создает обработчик события Click и осуществляет переход к редактированию его кода. Создадим этим способом два обработчика:

```
protected void ButtonBack_Click(object sender, EventArgs e)
{
}
```

```
protected void ButtonClear_Click(object sender, EventArgs e)
{
}
```

Следует обратить внимание, что после создания обработчиков ссылка на них появилась в коде страницы (выделено желтым):

```
<asp:Button ID="ButtonBack" runat="server" Text="[<=]"
BackColor="Yellow" Font-Size="Large" Height="48px" Width="48px"
OnClick="ButtonBack_Click" />
```

Обработчик буквенной кнопки нельзя создать таким же образом, так как в дизайне буквенные кнопки отсутствуют – они создаются автоматически. Следовательно, заготовку для обработчика придется создать вручную:

```
protected void ButtonLetter_Click(object sender, EventArgs e)
{
}
```

Для того чтобы подключить его к динамически создаваемым кнопкам, в цикл создания кнопок (Рисунок 40Рисунок 41) необходимо добавить соответствующую строку:

```
b.Click += ButtonLetter_Click;
```

Сформируем код этих обработчиков.

### 2.10.1. Обработчик нажатия на буквенную кнопку

Для начала необходимо определить, какая именно кнопка вызвала событие Click. Для этого используется параметр sender. Чтобы получить доступ к свойствам кнопки, используется явное приведение типа:

```
Button b = (Button)sender;
```

Далее производится добавление буквы (определяем ее по содержимому кнопки) к текстовому полю LabelWord:

```
LabelWord.Text += b.Text;
```

Так как каждую буквенную кнопку можно нажать только один раз, блокируем эту кнопку при помощи ее свойства Enabled:

```
b.Enabled = false;
```

Тут имеет смысл подумать о том, как будет работать обработчик ButtonBack\_Click. Он должен удалить последнюю введенную букву

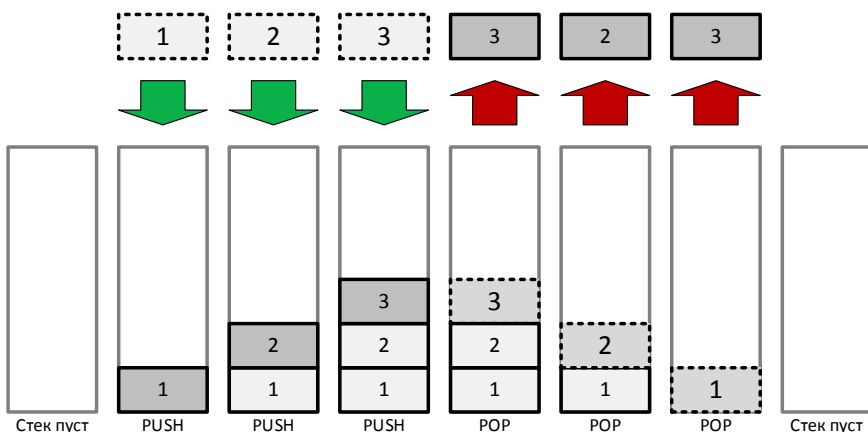
из текстового поля и разблокировать соответствующую кнопку. Но так как одна и та же буква в слове может встречаться несколько раз, определить необходимую кнопку поиском невозможно. Следовательно, необходимо запоминать порядок нажатия буквенных клавиш. Для этого обычно применяется структура данных под названием стек.

### **2.10.2. Стек – основные понятия**

Стеком называется структура данных, представляющая собой список элементов, организованный по принципу LIFO (англ. last in – first out, «последним пришёл – первым вышел»). Понятие стека впервые было введено Аланом Тьюрингом в 1946 году. Стек поддерживает две основные операции:

- PUSH – положить элемент в стек
- POP – вытолкнуть верхний элемент из стека

Принцип работы стека проиллюстрирован ниже (Рисунок 45).



**Рисунок 45. Иллюстрация принципа работы стека**

Для реализации стека может быть использован шаблон класса `Stack<>`. В отличие от обычного класса, шаблон класса представляет собой более высокий уровень абстракции, позволяющий создавать классы, реализующие одни и те же алгоритмы над различными типами данных. Тип данных является параметром шаблона, и в результате возникает конкретный класс, который обрабатывает объекты заданного типа данных в соответствии с шаблонами методов. Описание стека для наших целей будет выглядеть так:

```
private Stack<int> buttons = new Stack<int>();
```

Следует обратить внимание, что использована конструкция с одной временной инициализацией поля при помощи конструктора класса. Используется стек, содержащий целые числа, так как в него предполагается помещать целочисленные индексы кнопок (это облегчит реализацию обработчика `ButtonBack_Click`, как будет показано ниже).

Для того чтобы положить элемент в стек, используется его метод `Pop`:



```
buttons.Push(ButtonPanel.Controls.IndexOf(b));
```

Для получения индекса кнопки используется метод `IndexOf` коллекции элементов управления `Controls` панели кнопок `ButtonPanel` (следует напомнить, что буквенные кнопки расположены внутри панели `ButtonPanel`, поэтому используется ее коллекция `Controls`, а не одноименная коллекция страницы). Этот метод возвращает целочисленный индекс элемента управления в коллекции. Индекс начинается с 0, но в данном случае знать это совершенно не обязательно, так как он будет использоваться для адресации кнопки без каких-либо преобразований.

Полностью код обработчика нажатия на буквенную кнопку выглядит следующим образом (Рисунок 46).

```
protected void ButtonLetter_Click(object sender, EventArgs e)
{
    Button b = (Button)sender;
    LabelWord.Text += b.Text;
    b.Enabled = false;
    buttons.Push(ButtonPanel.Controls.IndexOf(b));
}
```

Рисунок 46. Обработчик нажатия на буквенную кнопку

### 2.10.3. Удаление последней введенной буквы

Обработка нажатия на кнопку, которая должна удалять последнюю набранную букву, состоит из следующих шагов:

- Удалить последнюю букву из текстового поля
- Разблокировать соответствующую букву

Для удаления буквы используются простые строковые методы (следует помнить, что нумерация символов в строке начинается с 0):

```
LabelWord.Text = LabelWord.Text.Substring(0,
LabelWord.Text.Length - 1);
```

Для определения номера кнопки, которую надо разблокировать, используем операцию выталкивания из стека:

```
int id = buttons.Pop();
```

Включение кнопки потребует явного приведения типа, так как свойство Enabled у базового класса Control отсутствует:

```
((Button)Controls[id]).Enabled = true;
```

Так как пользователь может нажать кнопку [ $\leq$ ] и в том случае, если ни одна буквенная кнопка еще не была нажата, имеет смысл проверить, что хотя бы одна буква была нажата. Это можно сделать двумя способами – проверить длину строки LabelWord.Text или количество элементов стека buttons.Count. Ниже используется первый способ (Рисунок 47).

```
protected void ButtonBack_Click(object sender, EventArgs e)
{
    if (LabelWord.Text.Length > 0)
    {
        LabelWord.Text = LabelWord.Text.Substring(0,
            LabelWord.Text.Length - 1);
        int id = buttons.Pop();
        ((Button)ButtonPanel.Controls[id]).Enabled = true;
    }
}
```

Рисунок 47. Обработчик, удаляющий последнюю введенную букву

#### 2.10.4. Очистка набранного слова

Для очистки набранного слова необходимо не только присвоить пустую строку соответствующему свойству метки LabelWord, но и разблокировать все ранее заблокированные кнопки. Это можно сделать, например, при помощи цикла while (Рисунок 48).

```
protected void ButtonClear_Click(object sender, EventArgs e)
{
    LabelWord.Text = "";
```

```
while (buttons.Count > 0)
{
    int id = buttons.Pop();
    ((Button)ButtonPanel.Controls[id]).Enabled = true;
}
}
```

**Рисунок 48. Обработчик для кнопки очистки набранного слова**

Если теперь запустить приложение, нажать какую-либо буквенную кнопку, а потом нажать кнопку [=] (Рисунок 49), то вместо нормального функционирования программы мы получим необработанное исключение (Рисунок 50). В чем же его причины?

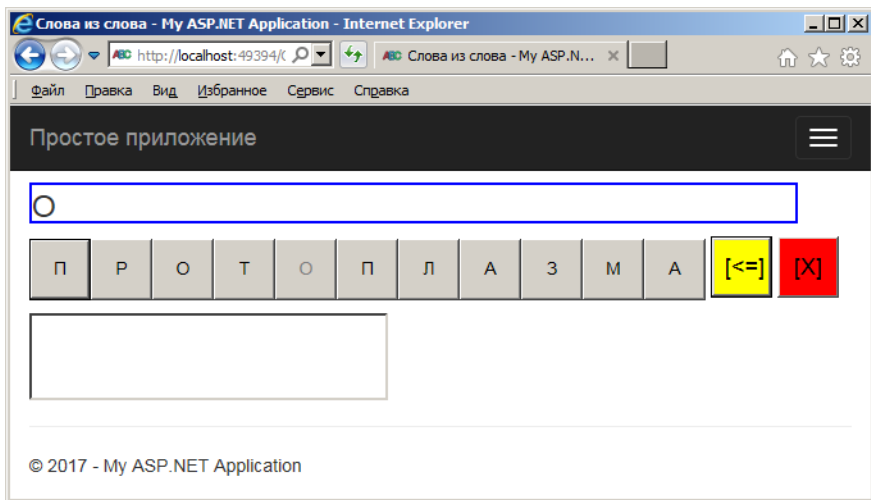


Рисунок 49. Запущенное приложение

```
// Определение идентификатора последней нажатой кнопки
```

```
int id = buttons.Pop();
```

```
// Включение соответствующей кнопки
```

```
((Button)Controls[id]).E
```

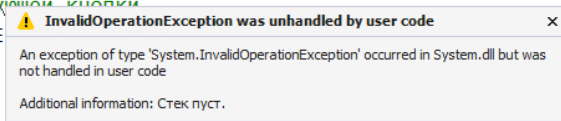


Рисунок 50. Необработанное исключение – стек пуст

### 2.10.5. Жизненный цикл страницы ASP.NET

Мы столкнулись с понятием «жизненный цикл (ЖЦ) страницы».

Каждый запрос к серверу, то есть каждое нажатие кнопки или иное событие приводит к обращению к веб-серверу по протоколу HTTP, при этом используется метод GET или POST (например, в зависимости от того, осуществляем ли мы переход по гиперссылке или нажимаем кнопку OK в форме). Запрос обрабатывается веб-сервером, роль которого выполняет служба Internet Information Services (IIS). Если запрос распознается как относящийся к ASP.NET Framework (например, по расширению страницы – ASPX), то он передается в

обработчик ASP.NET, который принимает решение, прочитать ли готовую страницу из кэша или запустить жизненный цикл страницы. В начале жизненного цикла осуществляется определение класса страницы, конструируется объект этого класса и выполняются этапы жизненного цикла страницы (Таблица 5). В результате выполнения жизненного цикла объект страницы удаляется, а сформированный HTML-код передается клиенту (браузеру).

**Таблица 5. Этапы жизненного цикла страницы ASP.NET**

Этап	Описание
Запуск (Start)	На начальном этапе устанавливаются свойства страницы, например Request и Response. На этом этапе страница также определяет, является ли запрос обратной передачей (POST) или новым запросом, и устанавливает свойство IsPostBack. Кроме этого, на этом этапе устанавливается свойство страницы UICulture. После завершения данного этапа вызывается событие страницы PreInit.
Инициализация страницы (Initialization)	Во время инициализации страницы элементы управления страницы являются доступными, устанавливаются все свойства элементов управления UniqueID. На странице также применяются темы. Если текущий запрос является обратным запросом, данные обратного запроса к этому моменту еще не загружены, а значения свойств элементов управления не восстановлены к значениям в состоянии просмотра. После завершения данного этапа вызывается событие страницы Init.
Загрузка (Load)	Во время загрузки, если текущий запрос является обратным запросом, в свойства элементов управления будут переданы данные, восстановленные из состояния просмотра и состояния управления. После завершения данного этапа вызывается событие страницы Load.

Этап	Описание
Обработка событий обратного запроса (Postback event handling)	Если запрос является обратным, вызывается соответствующие обработчики событий.
Проверка (Validation)	Во время проверки вызывается метод Validate всех проверяющих элементов управления, который устанавливает свойство IsValid отдельных проверяющих элементов управления и страницы. После завершения данного этапа вызывается событие страницы PreRender.
Формирование (Rendering)	Перед формированием производится сохранение состояния просмотра страницы и всех элементов управления. На этапе формирования страницы вызывает метод Render для каждого элемента управления. Методу Render передается объект класса HtmlTextWriter, который осуществляет запись полученного HTML-кода в выходной поток, задаваемый свойством OutputStream класса HttpResponse. Класс HttpResponse используется для возврата результата выполнения страницы клиенту (свойство Response страницы).
Выгрузка (Unload)	К этому моменту страница полностью сформирована, отправлена клиенту (браузеру), все свойства страницы (например, Response и Request) выгружены, выполнена полная очистка, объект страницы готов к удалению. После завершения данного этапа вызывается событие страницы Unload.

В процессе жизненного цикла страницы вызываются определенные события, которые могут быть использованы разработчиком для изменения внешнего вида страницы (Таблица 6). Это также проиллюстрировано ниже графически (Рисунок 51).

Таблица 6. События жизненного цикла страницы ASP.NET

Событие	Назначение
<b>PreInit</b>	<p>Во время этого события можно использовать свойство <code>IsPostBack</code>, для того, чтобы определить вызвана ли эта страница в первый раз или в результате обратного запроса. В плане управления страницей разработчик может: создавать динамически элементы управления, динамически устанавливать шаблон дизайна или тему оформления, считывать или устанавливать свойства объекта <code>Profile</code>.</p> <p>Стоит особо отметить, что на данном этапе, если страница была вызвана в результате обратного запроса, свойства элементов управления еще не восстановлены – они будут восстановлены только к событию <b>Load</b>. В случае, если разработчик самостоятельно задаст значения свойства на этом этапе, впоследствии эти значения могут быть изменены.</p>
<b>Init</b>	На этом этапе разработчик может считывать или инициализировать свойства элементов управления.
<b>Load</b>	На этом этапе разработчик может считывать или изменять свойства элементов управления.
<b>PreRender</b>	Последняя возможность внести изменения во внешний вид страницы
<b>Unload</b>	<p>Освобождение занятых ресурсов (закрытие открытых соединений с базой данных, завершение работы с файлами и т.п.)</p> <p>Важно, что на этом этапе страница уже сформирована и отправлена клиенту, и попытка внести какие-либо изменения (например, вызвав метод <code>Response.Write()</code>), приведет к исключению.</p>

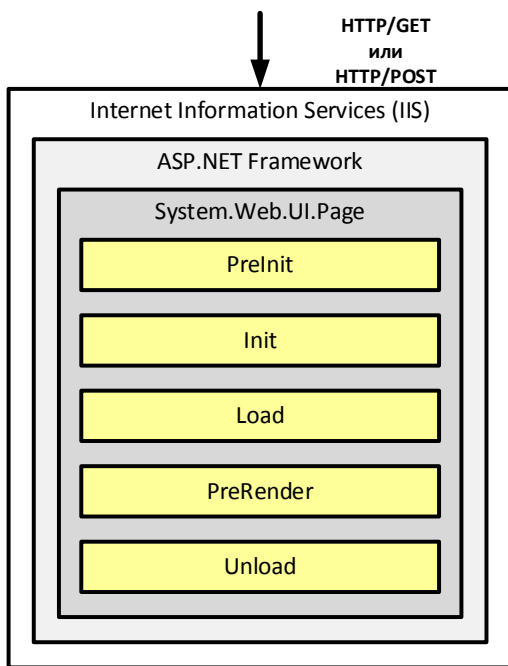


Рисунок 51. Жизненный цикл страницы ASP.NET

### 2.10.6. Передача данных при помощи сессии

Из всего вышеперечисленного следует, что свойство `buttons`, описанное в странице, не сохраняется между вызовами страницы, так как объект страницы создается каждый раз заново. В то же время значение свойства `LabelWord.Text` каким-то образом сохраняется между вызовами страницы, что видно при выполнении приложения (Рисунок 49). Это обеспечивается при помощи механизма сохранения и восстановления данных при обратном запросе. Для того чтобы передавать пользовательские данные таким же образом, как и данные элементов управления, используется свойство страницы `Session`. Это свойство позволяет сохранять именованные объекты таким образом, что они передаются клиенту вместе с остальным HTML-кодом и помещаются в скрытое поле веб-формы. После об-



ратного запроса это скрытое поле обрабатывается ASP.NET, и из его содержимого восстанавливаются все переданные туда объекты.

Для того, чтобы воспользоваться объектом Session, необходимо поле buttons заменить на одноименное свойство, которое будет обращаться к объекту Session для хранения данных нашего стека. Реализация свойства может выглядеть следующим образом (Рисунок 51). Необходимо обратить внимание на то, что при первом обращении к свойству выполняется автоматическое создание нового стека и сохранение его в объекте Session.

```
private Stack<int> buttons
{
    get
    {
        if (Session["buttons"] == null)
        {
            // данные сессии не найдены, создадим стек
            var stack = new Stack<int>();
            Session["buttons"] = stack;
            return stack;
        }
        else
        {
            // данные сессии найдены, вернем их
            return (Stack<int>)Session["buttons"];
        }
    }
    set
    {
        Session["buttons"] = value;
    }
}
```

Рисунок 52. Реализация свойства buttons

Для того чтобы синтаксис был более компактным, переменная stack описана с применением ключевого слова **var**. Это ключевое слово означает, что фактический тип переменной определяется по значению, при помощи которого она инициализируется. Использование **var** требует обязательной инициализации переменной при объяв-

лении. В данном случае эта конструкция полностью эквивалентна следующему оператору:

```
Stack<int> stack = new Stack<int>();
```

Объект Session может индексироваться целыми числами или строками. В данном случае использована индексация строками – так удобнее, чем заводить отдельный список констант (пока их планируется всего одна).

### 2.10.7. Обработка исключений в веб-приложениях

Исправление сделанной ошибки не отменяет необходимости добавить обработку исключений во все обработчики. Для веб-приложений имеется та особенность, что исключение выбрасывается на сервере, а отобразить его имеет смысл на клиенте. Исключение, выбрасываемое в веб-приложении, можно сохранить в каком-либо протоколе на сервере, можно передать для отображения на клиенте (браузере), а можно сделать и то и другое. В данном случае предлагается просто отобразить текст исключения внизу нашей страницы Page для того, чтобы пользователь был проинформирован о возникшей проблеме.

Для размещения текста исключения необходимо добавить еще один элемент управления – метку. Она может выглядеть следующим образом (для того, чтобы текст ошибки визуально выделялся, он окрашен в красный цвет при помощи атрибута ForeColor):

```
<p>  
  <asp:Label ID="ErrorLabel" runat="server" ForeColor="Red">  
  </asp:Label>  
</p>
```

Обработка исключения при этом сводится к заполнению поля ErrorLabel.Text текстом исключения, как показано ниже (этот блок ко-

да необходимо добавить во все обработчики событий нашей страницы, включая обработчик Page\_Load):

```
try
{
    ...тут располагается собственно код обработчика события...
}
catch (Exception ex)
{
    ErrorLabel.Text = ex.Message;
}
```

Рисунок 53. Обработчик исключений в веб-приложении

### 2.10.8. Управление состоянием буквенных клавиш

После всех этих модификаций приложение выглядит следующим образом (Рисунок 54). Нажатие кнопок [=] и [X] не приводят к выбросу исключения и обрабатываются корректно. Но видно, что блокируется только последняя нажатая буквенная клавиша.

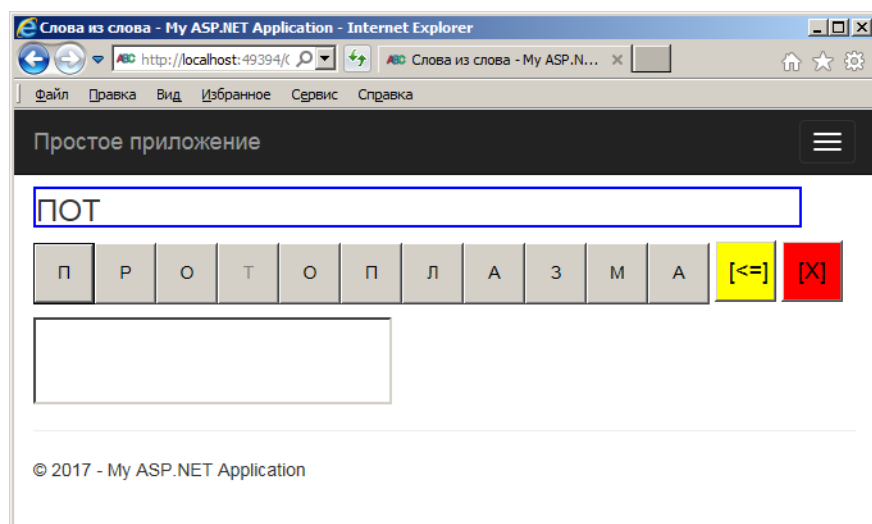


Рисунок 54. Текущий внешний вид приложения

Так как буквенные клавиши создаются динамически, то их состояние не может быть сохранено стандартными средствами ASP.NET, и нам придется позаботиться об этом самостоятельно. При этом, если обратиться к разделу 2.10.5, становится понятно, что пока наше приложение реализовано не совсем корректно. Динамическое создание элементов управления следовало бы делать на шаге PreInit. Но наши буквенные кнопки добавляются не прямо на страницу, а на панель ButtonPanel. Сама же панель будет создана и проинициализирована как раз на шаге Init. Поэтому код по созданию буквенных кнопок следует разместить именно в обработчике Page\_Init, который с учетом обработки исключений приобретает следующий вид (Рисунок 55).

```
protected void Page_Init(object sender, EventArgs e)
{
    try
    {
        for (int i = 0; i < word.Length; i++)
        {
            Button b = new Button()
            {
                Text = word[i].ToString(),
                ID = "button" + i,
                Width = new Unit("48px"),
                Height = new Unit("48px")
            };
            b.Click += ButtonLetter_Click;
            ButtonPanel.Controls.AddAt(i, b);
        }
    }
    catch (Exception ex)
    {
        ErrorLabel.Text = ex.Message;
    }
}
```

**Рисунок 55. Обработчик события инициализации страницы**

В обработчик события загрузки страницы вместо того кода, который там находился ранее (повторим, он перенесен в обработчик Page\_Init), следует поместить код, который будет блокировать все уже нажатые буквенные клавиши. Для этого используется цикл foreach по стеку buttons (стек позволяет себя использовать в том числе и как объект для последовательного перебора). Код обработчика выглядит следующим образом (Рисунок 56).

```
protected void Page_Load(object sender, EventArgs e)
{
    try
    {
        foreach (int id in buttons)
        {
            ((Button)ButtonPanel.Controls[id]).Enabled = false;
        }
    }
    catch (Exception ex)
    {
        ErrorLabel.Text = ex.Message;
    }
}
```

**Рисунок 56. Обработчик события загрузки страницы**

После всех этих изменений приложение с точки зрения управления буквенными кнопками работает корректно (Рисунок 57).

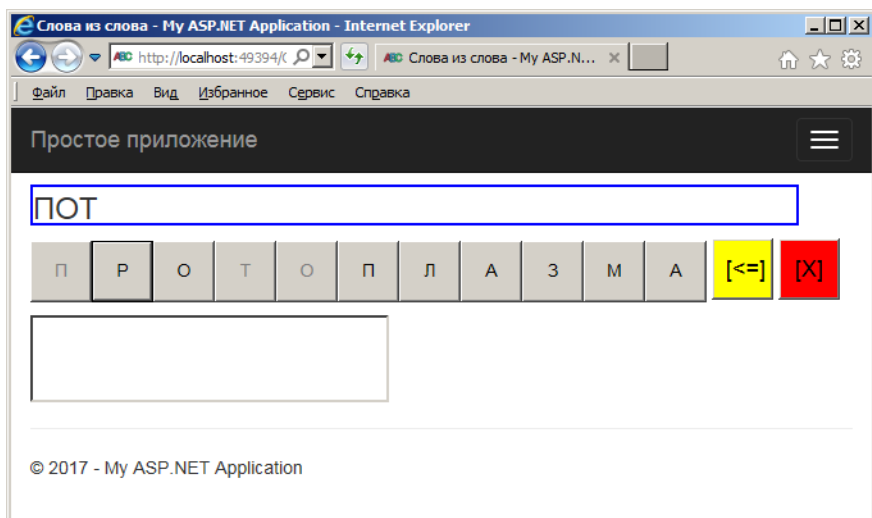


Рисунок 57. Корректная блокировка буквенных кнопок

## 2.11. Проверка введенного слова по словарю

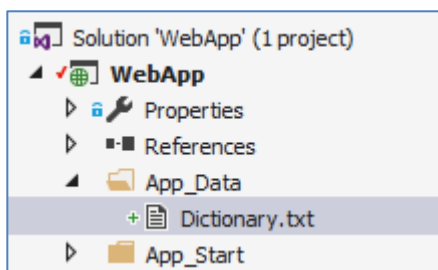
### 2.11.1. Добавление файла словаря

Для того чтобы наш прототип игры мог считаться функционально законченным, осталось сделать следующее – загрузить словарь слов и при вводе слова проверять его по словарю. Если введенное слово присутствует в словаре, то оно добавляется в список найденных слов (а введенное слово очищается). Если оно уже присутствует в списке найденных слов, то повторно оно туда не добавляется, и очистка введенного слова тоже не производится. Это позволит корректно вводить слова, начало которых совпадает с другим существующим словом (например, слова КОЛ и КОЛОК).

Для хранения словаря будем использовать простой (можно даже сказать, примитивный) текстовый файл, на каждой строке которого будет располагаться одно слово. Добавим текстовый файл с именем Dictionary.txt в проект, разместив его в папке App\_Data (Рисунок 58).

Каталог App\_Data специально предназначен для хранения файлов данных, и веб-приложение настроено таким образом, что файлы, расположенные в данном каталоге, являются недоступными для внешнего клиента.

Следует напомнить, что добавление нового файла к проекту осуществляется при помощи выбора в контекстном меню файла проекта пункта «Add | New Item», текстовые файлы находятся в разделе General.



**Рисунок 58. Добавление текстового файла в проект**

Файл следует заполнить словами самостоятельно; для отладки можно использовать следующий набор слов:

- ПЛАЗМА
- ПОТ
- РОТ
- ТОР

Следует сразу отметить, что для того, чтобы файл данных был помещен в соответствующий каталог приложения в процессе сборки проекта, необходимо в его свойствах (Properties) установить режим Copy to Output Directory в значение Copy if newer (Рисунок 59) или Copy Always (значение по умолчанию – Do not copy).

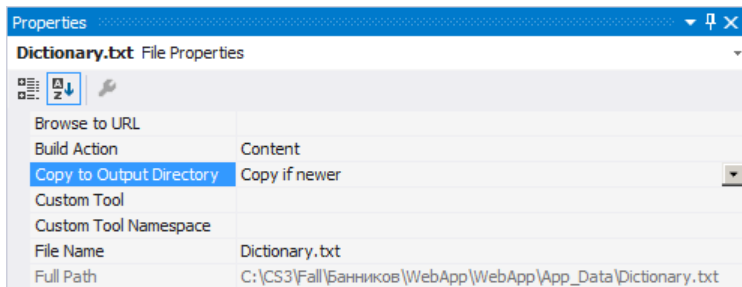


Рисунок 59. Свойства текстового файла

### 2.11.2. Загрузка файла словаря

Для того чтобы загрузить текстовый файл, воспользуемся стандартным методом `System.IO.File.ReadAllLines`. Он принимает единственный параметр – путь к файлу и возвращает массив считанных строк – то, что и требуется в данном случае.

Определим еще одно поле нашего класса `Game` для хранения словаря слов:

```
private string[] dict;
```

Чтобы определить путь к файлу, используем полезную возможность ASP.NET – получение физического пути к файлу по его виртуальному (относительно корня приложения) пути. В результате в конце кода обработчика `Page_Load` необходимо добавить всего две строки:

```
string name = Server.MapPath("~/App_Data/Dictionary.txt");  
dict = System.IO.File.ReadAllLines(name);
```

### 2.11.3. Поиск слов по словарю и заполнение списка слов

Поиск по словарю реализуется просто – для этого предусмотрен специальный метод `Contains`. Поиск в списке найденных слов придется выполнить вручную, так как список `ListWords` (точнее, его свойство `Items`) содержит не строки, а объекты класса `ListItem`. Поэтому придется выполнить перебор при помощи цикла `foreach`. Если



слово потребуется добавить в список, используется метод Add. Для очистки введенного слова (и восстановления состояния буквенных кнопок) вызовем обработчик ButtonClear\_Click, передав ему в качестве входных параметров значения, в свою очередь полученные обработчиком ButtonLetter\_Click. Итоговый код, который надо поместить в конец обработчика ButtonLetter\_Click, представлен ниже (Рисунок 59).

```
if (dict.Contains(LabelWord.Text))
{
    // слово найдено в словаре
    foreach (ListItem item in ListWords.Items)
    {
        if (item.Value == LabelWord.Text)
        {
            return;
        }
    }
    ListWords.Items.Add(LabelWord.Text);
    ButtonClear_Click(sender, e);
}
```

**Рисунок 60. Обработка словаря**

После запуска наше приложение функционирует точно так, как и задумано (Рисунок 60).

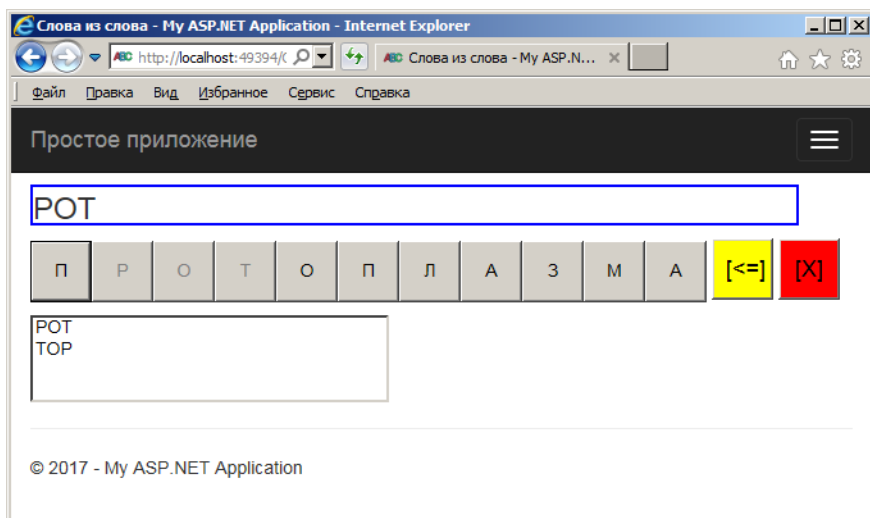


Рисунок 61. Готовое приложение

### 3. Терминология языка C#

Таблица 7. Термины языка C#

Термин	Определение
<b>IDE</b>	Интегрированная среда разработки (Integrated Development Environment). Приложение, которое предоставляет универсальный пользовательский интерфейс для различных средств разработки, в том числе компилятора, отладчика, редактора кода и конструкторов.
<b>Анонимный метод</b>	Анонимный метод — это блок кода, который передается в качестве параметра делегату.
<b>Базовый класс</b>	Класс, от которого наследует другой, "производный" класс.

Термин	Определение
<b>Вложенный тип</b>	Тип, объявленный внутри объявления другого типа.
<b>Делегат</b>	Делегат — это тип, который ссылается на метод. Когда делегату назначается методу, он ведет себя в точности, как этот метод.
<b>Деструктор</b>	Особый метод класса или структуры, который подготавливает экземпляр для уничтожения системой.
<b>Доступный член</b>	Член, доступ к которому можно получить из данного типа. Член, доступный для одного типа, не обязательно доступен для другого типа.
<b>Изменяемый тип</b>	Тип, после создания экземпляра которого свойства, поля и данные этого экземпляра могут изменяться. Изменяемыми являются почти все ссылочные типы.
<b>Интерфейс</b>	Тип, который содержит только подписи открытых методов, событий и делегатов. Объект, который наследует от интерфейса, должен реализовывать все методы и события, определенные в интерфейсе. Классы и структуры могут наследовать любое количество интерфейсов.
<b>Итератор</b>	Итератором называют метод, который позволяет объектам-получателям класса, содержащего коллекцию или массив, использовать оператор <b>foreach</b> для перебора коллекции или массива.
<b>Класс</b>	Тип данных, описывающий объект. Классы содержат данные и методы обработки этих данных

Термин	Определение
<b>Конструктор</b>	Особый метод класса или структуры, который инициализирует объект данного типа.
<b>Метод</b>	Именованный блок кода, который предоставляет поведение класса или структуры.
<b>Метод доступа</b>	Метод, который устанавливает или извлекает значение закрытого члена данных, связанное со свойством. Для свойств, доступных для чтения и записи, предусмотрены методы доступа <b>get</b> и <b>set</b> . Для свойств, доступных только для чтения, применяется только метод доступа <b>get</b> .
<b>Модификатор доступа</b>	Зарезервированное слово, например <b>private</b> , <b>protected</b> , <b>internal</b> или <b>public</b> , которое ограничивает доступ к типу или члену типа.
<b>Наследование</b>	Язык C# поддерживает наследование. Это означает, что класс, производный от другого класса, называемого базовым, наследует те же методы и свойства. В наследовании участвуют базовые и производные классы.
<b>Недоступный член</b>	Член, доступ к которому невозможно получить из данного типа. Член, недоступный для одного типа, не обязательно недоступен для другого типа.
<b>Неизменяемый тип</b>	Тип, после создания экземпляра которого свойства, поля и данные этого экземпляра не изменяются. Большинство типов значений являются неизменяемыми.
<b>Объект</b>	Экземпляр класса. Объект существует в памяти и содержит данные и методы для обработки этих данных.

Термин	Определение
<b>Оптимизация кода</b>	Повторное использование ранее введенного кода. Редактор кода может выполнить интеллектуальное форматирование кода, чтобы, например, преобразовать выделенный блок кода в метод.
<b>Поле</b>	Член данных класса или структуры, к которому можно получить непосредственный доступ.
<b>Производный класс</b>	Класс, который использует наследование для получения, расширения или изменения данных другого, "базового" класса.
<b>Свойство</b>	Член данных, доступ к которому осуществляется посредством метода доступа.
<b>Событие</b>	Член класса или структуры, который отправляет уведомления об изменении.
<b>Ссылочный тип</b>	Тип данных. Переменная, объявленная как ссылочный тип, указывает на расположение, в котором хранятся данные.
<b>Статический</b>	Для существования класса или метода, объявленного статическим, не требуется создавать его экземпляр с помощью ключевого слова <b>new</b> . В качестве примера статического метода можно назвать метод <code>Main()</code> .
<b>Стек вызова</b>	Ряд вызовов метода, который начинается с запуска программы и заканчивается оператором, выполняющимся в данный момент.

Термин	Определение
<b>Структура</b>	Составной тип данных, содержащий, как правило, несколько переменных, между которыми установлено некоторое логическое отношение. Структуры могут также содержать методы и события. Структуры не поддерживают наследование, но поддерживают интерфейсы. Структура является типом значения, тогда как класс — это ссылочный тип.
<b>Тип значения</b>	Тип значения — это тип данных, который располагается в стеке, в отличие от ссылочного типа, располагающегося в куче. Типами значения являются встроенные типы, в том числе числовые типы, а также тип структуры и тип "nullable". Ссылочными типами являются тип класса и строковый тип.
<b>Универсальные шаблоны</b>	Универсальные шаблоны позволяют определить класс и метод, который определяется с помощью параметра типа. Когда клиентский код создает экземпляр типа, в качестве аргумента он указывает определенный параметр.
<b>Член</b>	Поле, свойство, метод или событие, объявленное в классе или структуре.

## 4. Зарезервированные ключевые слова C#

Ключевые слова — это предварительно определенные зарезервированные идентификаторы, имеющие специальные значения для компилятора. Их нельзя использовать в программе в качестве идентификаторов, если только они не содержат префикс @. Например,

@if является допустимым идентификатором, но if таковым не является, поскольку if – это ключевое слово.

В частности, ключевые слова нельзя использовать как имена классов, пространств имен. А так как при создании проекта его имя становится именем его пространства имен по умолчанию, то и в качестве имени проекта ключевые слова лучше не использовать.

## 4.1. Основные ключевые слова

Ниже в таблице приведен список основных ключевых слов.

Таблица 8. Основные ключевые слова языка C#

Ключевое слово	Краткое описание
<b>abstract</b>	Абстрактный класс или член класса Абстрактный класс не имеет объектов этого класса Абстрактный метод класса не имеет реализации
<b>as</b>	Используется для приведения типов
<b>base</b>	Базовый (класс, метод)
<b>bool</b>	Логический тип данных. Возможные значения: <b>false</b> , <b>true</b>
<b>break</b>	Выход из цикла
<b>byte</b>	Целочисленный тип данных без знака в интервале от 0 до $2^8-1$ 255
<b>case</b>	Вариант ветвления в операторе switch
<b>catch</b>	Ловушка для исключения
<b>char</b>	Символьный тип данных – одна буква
<b>checked</b>	С проверкой
<b>class</b>	Класс
<b>const</b>	Константа
<b>continue</b>	Переход к проверке условия цикла
<b>decimal</b>	Десятичный тип данных

Ключевое слово	Краткое описание
<b>default</b>	Вариант ветвления по умолчанию в операторе switch
<b>delegate</b>	Делегат (метод, передаваемый в качестве параметра)
<b>do</b>	Цикл «после»
<b>double</b>	Вещественный тип данных с двойной точностью
<b>else</b>	Ветка условного оператора
<b>enum</b>	Перечислимый тип данных
<b>event</b>	Событие
<b>explicit</b>	Явное преобразование типа
<b>extern</b>	Внешний объект
<b>false</b>	Логическая константа «ложь»
<b>finally</b>	Блок кода, выполняющийся в самом конце вне зависимости от наличия исключений
<b>fixed</b>	Фиксация адреса
<b>Float</b>	Вещественный тип данных с одинарной точностью
<b>for</b>	Цикл со счетчиком
<b>foreach</b>	Цикл-итератор по всем элементам перечислимого объекта
<b>goto</b>	Оператор безусловного перехода к заданной метке (о ужас, он всё еще существует)
<b>if</b>	Условный оператор
<b>implicit</b>	Неявное преобразование типа
<b>in</b>	Часть конструкции цикла foreach
<b>int</b>	Целочисленный тип данных со знаком
<b>inter- face</b>	Интерфейс
<b>internal</b>	Внутренний
<b>is</b>	Проверка на соответствие заданному типу
<b>lock</b>	Блокировка одновременного (многопоточного) выполнения. Вынуждает фрагмент кода выполняться в единственном потоке, запрещает многозадачность на этом



Ключевое слово	Краткое описание
	участке
<b>long</b>	Целочисленный тип данных со знаком
<b>namespace</b>	Пространство имен
<b>new</b>	Создание нового объекта
<b>null</b>	Пустая ссылка, отсутствие значения
<b>object</b>	Объект – корневой класс в иерархии объектов .NET
<b>operator</b>	Оператор
<b>out</b>	Изменяемый параметр
<b>override</b>	Перекрытие метода
<b>params</b>	Параметры
<b>private</b>	Модификатор области видимости. Минимальная область видимости – только внутри класса
<b>protected</b>	Модификатор области видимости. Видимость только внутри класса и его потомков.
<b>public</b>	Модификатор области видимости. Полная видимость.
<b>readonly</b>	Член класса только для чтения (инициализация только внутри конструктора)
<b>ref</b>	Передача по ссылке
<b>return</b>	Возврат значения из метода
<b>sbyte</b>	Целочисленный тип данных со знаком
<b>sealed</b>	Класс, не имеющий наследников
<b>short</b>	Целочисленный тип данных со знаком
<b>sizeof</b>	Размер объекта
<b>stack-alloc</b>	Распределение стека в небезопасном участке кода
<b>static</b>	Статический элемент – относящийся к классу в целом, не имеющий контекста объекта класса
<b>string</b>	Строковый тип данных
<b>struct</b>	Структура

Ключевое слово	Краткое описание
<b>switch</b>	Оператор-переключатель
<b>this</b>	Ссылка на текущий экземпляр класса, используется также для объявления индексатора класса
<b>throw</b>	Выброс (генерация, формирование) исключения
<b>true</b>	Логическая константа «истина»
<b>try</b>	Проверка на наличие исключения
<b>typeof</b>	Определение типа объекта
<b>uint</b>	Целочисленный тип данных без знака
<b>ulong</b>	Целочисленный тип данных без знака
<b>unchecked</b>	Без проверки
<b>unsafe</b>	Небезопасно
<b>ushort</b>	Целочисленный тип данных без знака
<b>using</b>	Использование пространства имен (директива) Определение области действия объекта, после которого он должен быть уничтожен, а используемые им ресурсы освобождены (оператор)
<b>virtual</b>	Виртуальный (перекрываемый) метод
<b>void</b>	Нет значения
<b>volatile</b>	Поле, которое может быть непредсказуемо изменено в многопоточной среде. Исключается из оптимизации, которая предполагает однопоточное выполнение кода
<b>while</b>	Цикл «пока»

## 4.2. Контекстные ключевые слова

Контекстные ключевые слова были введены в язык C# в процессе его развития. Контекстные ключевые слова не являются жестко зарезервированными, но использовать их в качестве идентификаторов не рекомендуется.

Таблица 9. Контекстные ключевые слова языка C#

Ключевое слово	Краткое описание
<b>add</b>	Обработчик подписки на событие
<b>alias</b>	Псевдоним для одновременного подключения библиотек с идентичным полным именем, но различными версиями
<b>ascending</b>	По возрастанию
<b>async</b>	Асинхронная операция
<b>await</b>	Ожидание завершения асинхронной операции
<b>descending</b>	По убыванию
<b>dynamic</b>	Объявление переменной как динамической предотвращает проверку типов на этапе компиляции программы. Все проверки осуществляются в момент выполнения программы
<b>from</b>	Элемент запроса LINQ
<b>get</b>	Описание метода чтения значения свойства
<b>global</b>	Ссылка на глобальное (безымянное) пространство имен
<b>Group</b>	Элемент запроса LINQ. Определяет группировку записей
<b>into</b>	Элемент запроса LINQ. Используется для построения сложных запросов
<b>join</b>	Элемент запроса LINQ. Используется для выборки данных из нескольких таблиц
<b>let</b>	Элемент запроса LINQ.
<b>orderby</b>	Элемент запроса LINQ. Указывает на порядок сортировки записей

Ключевое слово	Краткое описание
<b>partial</b>	Частичный класс или метод. Частичный класс (интерфейс, структура) размещается в двух или более файлах. Частичный метод может быть определен (при помощи заголовка) в одном из таких файлов, а реализован в другом файле.
<b>remove</b>	Обработчик удаления подписки на событие
<b>select</b>	Элемент запроса LINQ
<b>set</b>	Описание метода записи значения свойства
<b>value</b>	В методе записи значения свойства используется как неявный входной параметр метода – записываемое значение
<b>var</b>	Описание переменной без указания ее типа. Тип переменной определяется по типу выражения, при помощи которого она инициализируется
<b>where</b>	Элемент запроса LINQ, определяющий условие выборки данных
<b>yield</b>	Используется в методах (свойствах, операторах), которые возвращают итераторы. Определяет очередной элемент данных, возвращаемый итератором.

## 5. Скалярные типы данных

Для целочисленных типов данных (Таблица 10) приведены количество бит в двоичном представлении числа, наличие знака и диапазон возможных значений (для наглядности в качестве дополнительного разделителя троек цифр использована запятая).

Для вещественных типов данных (Таблица 11) приведены количество бит в двоичном представлении числа, количество значащих цифр без потери точности и диапазон возможных значений.

**Таблица 10. Целочисленные типы данных**

Тип C#	.NET Framework	Бит	Знак	Диапазон
<b>sbyte</b>	System.SByte	8	Да	-128 ... 127
<b>byte</b>	System.Byte	8	Нет	0 ... 255
<b>short</b>	System.Int16	16	Да	-32,768 ... 32,767
<b>ushort</b>	System.UInt16	16	Нет	0 ... 65,535
<b>int</b>	System.Int32	32	Да	-2,147,483,648 ... 2,147,483,647
<b>uint</b>	System.UInt32	32	Нет	0 ... 4,294,967,295
<b>long</b>	System.Int64	64	Да	-9,223,372,036,854,775,808 ... 9,223,372,036,854,775,807
<b>ulong</b>	System.UInt64	64	Нет	0 ... 18,446,744,073,709,551,615

**Таблица 11. Вещественные типы данных**

Тип C#	.NET Framework	Бит	Цифры	Диапазон
<b>float</b>	System.Single	32	7	$\pm 1.5 \cdot 10^{-45} \dots \pm 3.4 \cdot 10^{38}$
<b>double</b>	System.Double	64	15-16	$\pm 5.0 \cdot 10^{-324} \dots \pm 1.7 \cdot 10^{308}$
<b>decimal</b>	System.Decimal	128	28-29	$\pm 7.9 \cdot 10^{-28} \dots \pm 7.9 \cdot 10^{28}$

## ЛИТЕРАТУРА

[1] C# 4.0. Полное руководство. Пер. с англ. / Герберт Шилдт. М.: Вильямс, 2015, 1056 с. ISBN 978-5-8459-1684-6

[2] Язык программирования C# 5.0 и платформа .NET 4.5. Пер. с англ. / Эндрю Троелсен, М.: Вильямс, 2015, 1312 с. ISBN 978-5-8459-1957-1

[3] Разработка простого калькулятора на языке C# : метод. указания к лабораторным работам по курсу «Разработка приложений на языке C#» / С. Н. Банников – М.: Изд-во МГТУ им. Н. Э. Баумана, 2016. - 134 с.: ил.

## Оглавление

Введение.....	3
1. Система контроля версий TFS .....	3
1.1. Общие сведения .....	3
1.2. Подключение к порталу .....	4
1.3. Настройка среды разработки .....	10
1.4. Учетные данные при подключении к TFS .....	19
1.5. Использование .....	24
1.6. Возможные проблемы при сборке проекта.....	27
1.6.1. Системная сборка не может быть загружена.....	27
2. Разработка простого веб-приложения на базе ASP.NET Web Forms .....	29
2.1. Архитектуры приложений.....	29
2.2. Постановка задачи .....	31
2.3. Создание проекта .....	31
2.4. Запуск проекта .....	34
2.5. Первичная настройка проекта .....	36
2.6. Создание новой страницы .....	38
2.7. Начальная разметка страницы .....	41
2.8. Динамическая генерация кнопок.....	47
2.9. Доработка дизайна приложения.....	50
2.10. Обработчики кнопок.....	53
2.10.1. Обработчик нажатия на буквенную кнопку .....	54

2.10.2. Стек – основные понятия .....	55
2.10.3. Удаление последней введенной буквы.....	57
2.10.4. Очистка набранного слова .....	58
2.10.5. Жизненный цикл страницы ASP.NET .....	60
3. Терминология языка C#.....	65
4. Зарезервированные ключевые слова C# .....	78
4.1. Основные ключевые слова .....	79
4.2. Контекстные ключевые слова.....	82
5. Скалярные типы данных .....	84
Литература.....	85