**What is Active Server Pages for .NET?**

**Introduction**

### Web-forms

1. Web-form of ASP.NET

2. Binding models and Web Forms in Visual Studio 2013

3. Using asynchronous methods in ASP.NET

4. Web-deployment of ASP.NET

## Model-View-Controller- MVC

## 1. Model-View-Controller for web

## 2. Model-View-Controller using experience, conclusion

## 3. Migration of applications ASP.NET MVC in containers Windows

## Web interface application programming interface

# 1.Getting started with ASP.NET Web API

# 2.One of the ASP.NET Web API

# 3.Availability management in projects ASP.NET Web API

4. ASP.NET Web API Conveyor

## Web-pages

# 1. Programming web pages ASP.NET (Razor) using Visual Studio

# 2. Redirecting and creating links in Razor Pages

# 3. HTML-helper cells.

### 1.Web-forms

# Practical task№1. Create ASP.NET MVC project

Practical task № 2. Create and process a view. site architecture

Practical task № 3,4. Create a simple data entry application

Practical task№ 5. Project creation

## 2 Model-View-Controller

## Practical task1 the Transition from ASP.NET in MVC 5 to Model-View-Controller 6

## Practical task2 Transferring configuration from ASP.NET in MVC 5 with MVC 6

## Practical task3 Introduction to Web application programming interface

## Practical task4 Routing System in the web application programming interface

## Practical task№ 5 Web application programming interface design in 7 steps

## 3. Web interface API. Web-pages

## Practical task1 An example of a web site on ASP.NET Web Forms

## Practical task2 Sample web site on the engine Razor

## Practical task3 Writing html code for an ASP.NET Web Forms page

## Practical task4 HTML form helpers

## Practical task5 Creating HTML Helpers

**What is Active Server Pages for .NET (ASP.NET)?**

**Introduction**

The Microsoft .NET Framework is a platform for building, deploying, and launching Web services and applications. It provides a high-performance, standards-based, multilingual environment that allows you to integrate existing applications with next-generation applications and services, as well as solve the problems of deploying and using Internet applications. The .NET Framework consists of three main parts — the common language runtime, the hierarchical set of unified class libraries, and the component version of ASP called ASP.NET.

ASP.NET is part of the .NET technology used to write powerful client-server Internet applications. It allows you to create dynamic HTML pages. ASP.NET is the result of a combination of the older ASP technology (active server pages) and the .NET Framework. It contains many ready-made controls, using which you can quickly create interactive web sites. You can also use services provided by other sites transparently to users of your site. In general, ASP.NET features are limited only by your imagination.

Let's discuss what dynamic HTML pages are and how they differ from static ones. The static page contains HTML hypertext markup code. When the author of the page writes it, he determines how the page will look to all users of the page. Page content will always be the same no matter who decides to view it and when. HTML language is quite enough to display information that rarely changes and does not depend on who views it. The HTML page is simple ASCII text, therefore, the client can work in any operating system.

It is quite clear that if you create a web page, describing its structure by means of HTML, it will be completely static in terms of content. That is, when viewed in a browser, it will carry exactly the same information that was written to it at the time of creation, and the data transmitted by the user can not be used to modify the content of the pages displayed to him: he can only see what was previously recorded in the final set of files.

But what if we want to display the current Euro rate or weather forecast on the page? If we wrote an HTML page yesterday, it will be obsolete today. Therefore, we must be able to create dynamic pages. Dynamic page content is information whose content is determined by who it is intended for, and which differs from view to view. It allows for two-way exchange of information – from the client to the server and back.

Dynamic web-pages are usually called web-pages that undergo a processing cycle on the server before being sent to the client. In the simplest case, it can be some program that modifies the static pages requested by the client, using the parameters of the received request and some data storage. Even with such a primitive organization, the "unsolvable" problem from the previous paragraph finds an obvious solution: it is enough to prepare only one static page – a template – and before sending the page programmatically substitutes in it the value obtained today from the Bank or the weather Bureau.

Most pages in the early stages of Internet development were static. The number of dynamic pages has been growing for the last 10 years. And it is clear, Internet users want not only to read ready information, and to be active actors. For example, they order goods in the online store, write diaries, participate in competitions. Information portals update the news every minute. Dynamic pages can adapt to a specific user, as well as respond to his actions in the browser. In what way? A lot of technologies have been invented for this. For example, in order to identify the user and save his settings for this site, cookies are used.

There are languages that can dynamically change the content of a web page. On the one hand, these are script languages that run directly on the client. Examples of scripting languages are JavaScript and VBScript. Scripts in these languages are embedded in the HTML that the server sends to the browser. Client-side scripts are marked with and tags . The browser interprets this code and shows the user the result. The code itself can be viewed through the view Source browser. Naturally, these programs cannot be large. For example, if you want to search a database, we can't send all of its contents to the user. But scripts can check the correctness of the request entered in the form, then you do not have to restart the server processing incorrect requests. Some programmers create JavaScript animation effects. One student intuit.ru I wanted to find a script that would send SMS-messages. Alas, this is impossible. Client-side scripting is not sufficient to create full dynamic pages. Even if the page uses JavaScript, animated images .gif, it's called static.

A dynamic web page must be created on the fly by a program running on the web server. The mechanism of cgi(Common Gateway Interface) gateways is widely used. First, the user receives a static page with the form. You know that there is an action attribute in the FORM tag. It specifies the address (URL) of the executable application. The server contains executable files of programs written, for example, in C/C++ or Delphi, which receive data from the input stream or from environment variables via HTTP Protocol and write a ready-made page to the standard output stream.

The user is sent an HTML code in response to the request, which was specially generated for him. This can be, for example, a search result in a search engine. CGI scripts can be written in an interpreted language (Perl) or even a command-line script. The input and output streams are remapped. The Internet server accepts the data entered by the user. After processing the received data, the resulting page is returned to the user. When cgi is executed, the program is loaded into the server memory, and when completed, it is deleted. When 100 clients access the server at the same time, 100 processes are created in memory to host the code for each of them. This has a negative impact on scalability. Let us remind you that scalability is the possibility of a smooth growth of the response time of the software system to the request with the growth of the number of concurrent users.

### To solve this problem, Microsoft has proposed an alternative-ISAPI (Internet Server Application Programming Interface)-extensions and filters. DLL libraries are used instead of executable files. The DLL code is in memory all the time and for each request creates not processes, but threads of execution. All threads use the same code. The ISAPI application runs in the IIS server process. This improves performance and scalability.

### You can create ISAPI extensions in Visual Studio C++ 6.0 by using the wizard.

### From ISAPI also has shortcomings related to software development. If we change the source code of the dll, we have to compile it and put it in the executable directory of the server. But since the previous version of the dll is in memory, you must stop the server to access the file change. At this time, clients will not be able to receive any document in the server, and, of course, will not be satisfied.

### Server-side scripting languages-php and asp. Asp technology was developed by Microsoft in the 90s.

### Asp code execution is supported by the server's ISAPI extension. In the configuration dialog of the IIS server will determine how to handle files with different extensions. An asp file is defined in server installations to handle THE url with the extension.dll. Asp files are sent to it for processing. The input comes asp, and the output has a stream of HTML-code.

### Installation process

ASP .NET 2.0 can be installed on computers running Windows 2000 with Service Pack 4, Windows XP with Service Pack 2, and later versions of Windows. Ready sites are preferred to be installed on Windows Server 2003.

You can use any development environment or even a text editor to develop your application, as long as you have access to IIS. If you want to take advantage of the full power of the Microsoft .NET Framework and ASP.NET and at the same time, to spend as little effort as possible, you need to use the development environment, specially designed for programming ASP.NET 2.0.

If you purchase Visual Studio .NET 2005, you'll only need it to work. The .NET Framework is contained on disks. It includes Visual Web Developer, which allows you to create professional web applications, as well as desktop applications in different programming languages. Microsoft products are released on DVD, but there is a set of two CDS from "Megasoft". Visual Studio .NET 2005 requires about 2 Gigabytes of disk space. When this is installed ASP.NET 2.0, the development environment, SQL Server Express, built-in web server, Crystal Reports, with special controls for ASP.NET 2.0.

1. **Web forms**

**1.Web forms ASP.NET**

Platform ASP.NET MVC is a framework for creating websites and web applications using the MVC pattern implementation.

The concept of pattern (template) MVC (model - view - controller) involves dividing the application into three components:

Controller (controller) is the class that provides the connection between the user and system performance and data storage. It receives the data entered by the user and processes it. And depending on the results of processing sends the user a certain output, for example, in the form of a view.

A view is the visual part or user interface of an application itself. Typically, the html-page that the user sees by going to the site.

A model represents a class that describes the logic of the data used.

The General scheme of interaction of these components can be represented as follows:

In this diagram, the model is an independent component - any changes to the controller or view do not affect the model. The controller and view are relatively independent components and can often be changed independently.

Due to this implements the concept of division of responsibility in connection with the easier it is to organize work on individual components. In addition, as a result, the application has better testability. And if the visual part or the frontend is important for us, then we can test the view independently of the controller. Or we can focus on the backend and test the controller.

Specific implementations and definitions of this pattern may vary, but due to its flexibility and simplicity, it has become very popular in recent years, especially in the field of web development.

The platform presents its implementation of the pattern ASP.NET MVC. 2013 was marked by the release of a new version ASP.NET MVC-MVC 5, as well as the release of Visual Studio 2013, which provides tools to work with MVC5.

Although many aspects of MVC 5 will not differ too much from MVC 4, much of one version is quite applicable to the other, but at the same time there are significant differences:

The concept of authentication and authorization has changed in MVC 5. Instead SimpleMembershipProvider system was introduced ASP.NET Identity, which uses components of OWIN and Katana.

To create a responsive and extensible interface, MVC 5 uses the Bootstrap css framework

Added authentication filters, but also introduces the functionality of the override filters

MVC 5 also adds routing attributes

These are the most important innovations in MVC 5. In addition, there are a number of less significant, such as the default use of Entity Framework 6, some changes when creating a project (the concept of One ASP.NET), additional components, etc.

In any case, all the skills gained while working with MVC 4 can be successfully applied when using MVC 5, taking into account, of course, innovations.

**REVIEW QUESTIONS**

1.What are web forms?

2. Fundamentals of controllers

3. View

4. Model

5. Features of the platform ASP.NET MVC

**Task**

1. Create a new project

2. Adding controls to a form

3. Setting up controls

1. **Binding models and Web Forms in Visual Studio 2013**

In this tutorial series shows the basic aspects of using model binding with web forms ASP.NET. Model binding makes it easy to interact with the data more efficient than working with data source objects (e.g., ObjectDataSource and SqlDataSource). This series starts with introductory material and moves to more advanced in later tutorials.

The model binding template works with any data access technology. This guide uses the Entity Framework, but you can use the data access technologies that you are most familiar with. From a data-bound server control, such as a Grid View, List View, Details View, and Form View control, specify the method names for selecting, updating, deleting, and creating data. In this tutorial, you will specify a value for Select Method.

This method provides the logic to retrieve the data. In the next tutorial, Update Method and Insert Method and Delete Method will set the values.

You can load the entire project in C# or Visual Basic. The downloadable code works with Visual Studio 2012 or Visual Studio 2013. It uses the Visual Studio 2012 template, which is slightly different from the Visual Studio 2013 template in this tutorial.

In this guide, you start the application in Visual Studio. You can also make the application available over the Internet by deploying it to your hosting provider .Microsoft offers free hosting services for up to 10 websites

free Azure trial account. For information about deploying a web project, Visual Studio, web service applications, Azure applications, see web deploy ASP.NET using Visual Studio series. This tutorial also shows how to use the Entity Framework Code First Migrations to deploy a SQL Server database to an Azure SQL database.

Software versions used in this guide

Microsoft Visual Studio 2013 or Microsoft Visual Studio Express 2013 for the Web

This tutorial also works with Visual Studio 2012, but there are some differences in the user interface and project template.

You will create

In this guide, you will need:

Create data objects that reflect University with students enrolled in courses

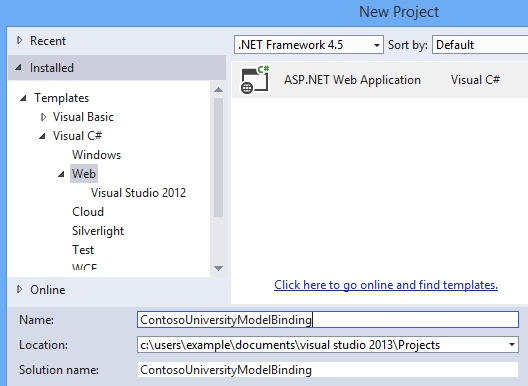
Building database tables from objects

Filling the database with test data

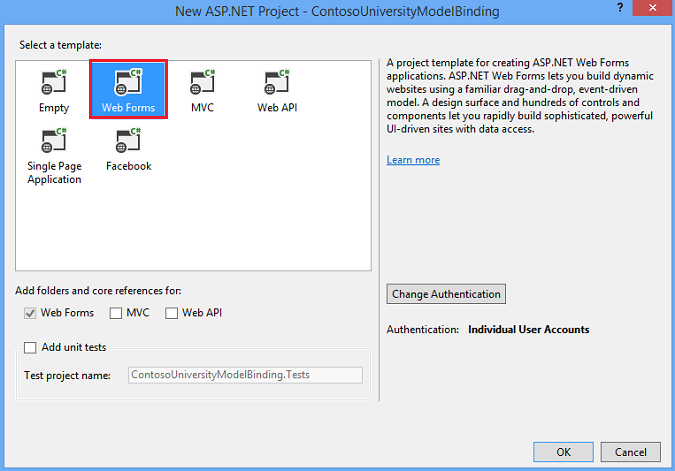
Displaying data in web forms

Project settings

In Visual Studio 2013, create a new web application ASP.NET called ContosoUniversity Model Binding.



Select the web form template and leave the default values. Click OK to configure the project.



First, you will make a number of small changes to the appearance of the node. Open The Site.Master file and change the header does not include the Contoso University is my app ASP.NET.

aspx

<title><%: Page.Title %> - Contoso University</title>

Then, change the title text from the appname to Contoso University.

aspx

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" runat="server" href="~/">Contoso University</a>

</div>

Also in Site.Master, change the navigation links displayed in the header to reflect on the pages that relate to this site. You don't need either about page or contact page to remove these links. Instead, you will need a link to the page with the name of the students. This page has not been created yet.

aspx

<ul class="nav navbar-nav">

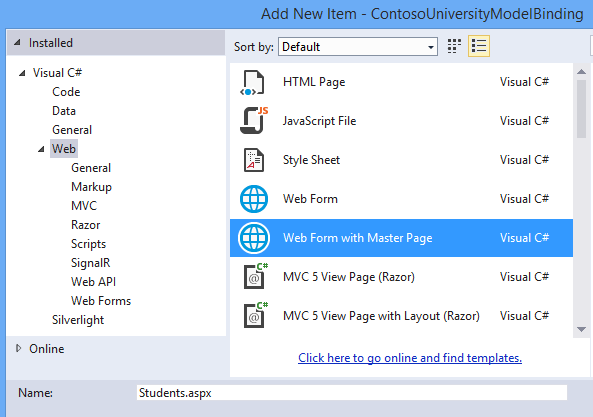
<li><a runat="server" href="~/">Home</a></li>

<li><a runat="server" href="~/Students">Students</a></li>

</ul>

Save and close the Site.Master.

You will now create web forms to display student data. Right-click the project, and add a new item. Select a web form from the main page of the template and name it Students.aspx.

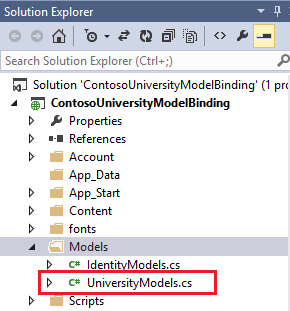


Select Site.Master as the home page for the new web form.

Creating a data model and database

Code First Migrations will use to create objects and the corresponding database tables. These tables will store information about students and their courses.

In the Models folder, add a new class named University Model s.cs.



In this file, define the SchoolContext, student, enrollment, and course classes as follows:

C#

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.ComponentModel.DataAnnotations;

namespace ContosoUniversityModelBinding.Models

{

public class SchoolContext : DbContext

{

public DbSet<Student> Students { get; set; }

public DbSet<Enrollment> Enrollments { get; set; }

public DbSet<Course> Courses { get; set; }

}

public class Enrollment

{

[Key]

public int EnrollmentID { get; set; }

public int CourseID { get; set; }

public int StudentID { get; set; }

public decimal? Grade { get; set; }

public virtual Course Course { get; set; }

public virtual Student Student { get; set; }

}

public class Course

{

[Key]

public int CourseID { get; set; }

public string Title { get; set; }

public int Credits { get; set; }

public virtual ICollection<Enrollment> Enrollments { get; set; }

}

public enum AcademicYear

{

Freshman,

Sophomore,

Junior,

Senior

}

}

The SchoolContext class is derived from DbContext, which manages database connections and changes to data.

In the student class, note that the attributes that were applied to the FirstName, LastName, and year properties. These attributes will be used to validate the data in this guide. To simplify the code for this project demonstration, these properties have been marked as having data validation attributes. In a real project, you can apply validation attributes to all properties that need validated data, such as properties in registration and course classes.

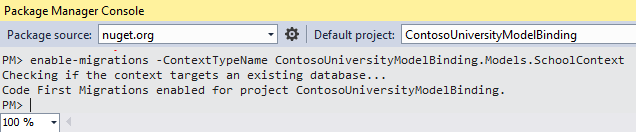
Save the University Model s.cs.

To configure the database, you will use the Code First Migrations tools based on these classes.

In the package Manager console, run the following command:

enable-migrations-ContextTypeName ContosoUniversityModelBinding.Models.SchoolContext

If the command succeeds, you will receive a message that States that you have enabled migrations



Note that the new file is named Configuration.cs will be created. In Visual Studio, this file is automatically opened after it is created. Contains a configuration class initial value method that allows you to pre-populate database tables with test data.

Add the following code to the initial value method. You will need to add using the instructions for ContosoUniversityModelBinding.Models namespace.

C#

namespace ContosoUniversityModelBinding.Migrations

{

using System;

using System.Data.Entity;

using System.Data.Entity.Migrations;

using System.Linq;

using ContosoUniversityModelBinding.Models;

internal sealed class Configuration : DbMigrationsConfiguration<SchoolContext>

{

public Configuration()

{

AutomaticMigrationsEnabled = false;

}

protected override void Seed(SchoolContext context)

{

context.Students.AddOrUpdate(

new Student {

FirstName = "Carson",

LastName = "Alexander",

Year = AcademicYear.Freshman },

new Student {

FirstName = "Meredith",

LastName = "Alonso",

Year = AcademicYear.Freshman },

new Student {

FirstName = "Arturo",

LastName = "Anand",

Year = AcademicYear.Sophomore },

);

context.SaveChanges();

context.Courses.AddOrUpdate(

new Course { Title = "Chemistry", Credits = 3 },

new Course { Title = "Microeconomics", Credits = 3 },

new Course { Title = "Macroeconomics", Credits = 3 },

new Course { Title = "Calculus", Credits = 4 },

new Course { Title = "Trigonometry", Credits = 4 },

new Course { Title = "Composition", Credits = 3 },

new Course { Title = "Literature", Credits = 4 }

);

context.SaveChanges();

context.Enrollments.AddOrUpdate(

new Enrollment { StudentID = 1, CourseID = 1, Grade = 1 },

new Enrollment { StudentID = 1, CourseID = 2, Grade = 3 },

new Enrollment { StudentID = 1, CourseID = 3, Grade = 1 },

new Enrollment { StudentID = 3, CourseID = 1 },

new Enrollment { StudentID = 4, CourseID = 1 },

new Enrollment { StudentID = 4, CourseID = 2, Grade = 4 },

new Enrollment { StudentID = 5, CourseID = 3, Grade = 3 },

new Enrollment { StudentID = 6, CourseID = 4 },

new Enrollment { StudentID = 7, CourseID = 5, Grade = 2 }

);

context.SaveChanges();

}

}

}

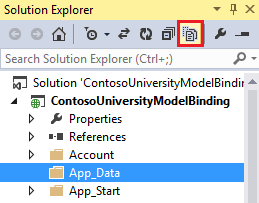
Save The Configuration.cs.

In the package Manager console, run the add-migration initial command.

Run the update-database command.

When you run this command, it is possible that the values of StudentID and CourseID have different values in the fill method. Open these tables in the database and find the existing values for StudentID and CourseID. Add these values to the code to populate the registration table.

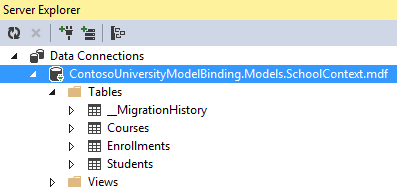
A database file has been added, but is currently hidden in the project. Click Show all files to display the file.



Note that the. MDF file will appear in the application\_folder.

файл базы данных

Double-click the. MDF file, and then open server Explorer. Now there are tables filled with data.



**Display data from related tables and students**

With the data in the database, you are now ready to retrieve that data and display it on a web page. You will use the GridView control to display data in columns and rows.

Open Students.aspx and find the Maincontent placeholder. In the frame, add a GridView control that includes the following code.

aspx

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" runat="server">

<asp:GridView runat="server" ID="studentsGrid"

ItemType="ContosoUniversityModelBinding.Models.Student" DataKeyNames="StudentID"

SelectMethod="studentsGrid\_GetData"

AutoGenerateColumns="false">

<Columns>

<asp:DynamicField DataField="StudentID" />

<asp:DynamicField DataField="LastName" />

<asp:DynamicField DataField="FirstName" />

<asp:DynamicField DataField="Year" />

<asp:TemplateField HeaderText="Total Credits">

<ItemTemplate>

<asp:Label Text="<%# Item.Enrollments.Sum(en => en.Course.Credits) %>"

runat="server" />

</ItemTemplate>

</asp:TemplateField>

</Columns>

</asp:GridView>

</asp:Content>

There are several important concepts in this markup code that you need to pay attention to. First, notice that the value is set for the SelectMethod property in the GridView element. This value specifies the name of the method that is used to retrieve data for the GridView element. In the next step, you will create this method. Second, notice that the ItemType property has the value of the Student class that you created earlier. By setting this value, you can reference the properties of this class in markup code.For example, the Student class contains a collection named registrations. You can use Item.Enrollments to retrieve this collection and then use LINQ syntax to get the amount of money registered in the account for each student.

In the code file, you must add the method that is specified for the SelectMethod value. Open Students.aspx.cs and add using statements

for ContosoUniversityModelBinding.Modules and System.Data.Entity namespaces.

C#

using ContosoUniversityModelBinding.Models;

using System.Data.Entity;

Then add the following method. Note that the name of this method matches the name you specified for SelectMethod.

C#

public IQueryable<Student> studentsGrid\_GetData()

{

SchoolContext db = new SchoolContext();

var query = db.Students.Include(s => s.Enrollments.Select(e => e.Course));

return query;

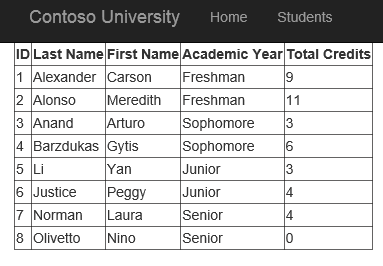
}

The Include clause improves query performance, but is not very important for queries to work. Without the Include clause, the data should be retrieved by using deferred loading, which is used to send a related separate query to the database each time that the data is retrieved. By providing Include clauses, the data is retrieved using a non-Debug load, which means all related data is retrieved using a single database query. In cases where a large part of the associated data is not used, pre-fetching can lead to reduced efficiency so as to obtain the additional data. However, in this case, a Seamless load provides the best performance because the related data is displayed for each record.

For more information about performance, be aware when you load related data, see the Lazy Eager and explicit loading of related data into reading related data by using the Entity Framework in the .NET MVC ASP Application section.

By default, the data is sorted by the values of the property marked as a key. You can add an OrderBy clause to specify a different sort value. In this example, the default StudentID property is used for sorting. The sort, pagination, and data filtering section will allow the user to select a column to sort.

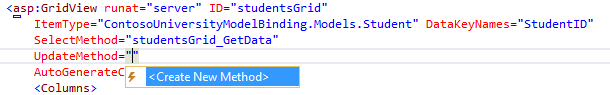
Start the web application and go to the students page. The student page displays the following student information:



**Automatic creation of model binding methods**

When working with this series of tutorials, you can simply copy the code from the tutorial into your project.However, the Disadvantage of this approach is that you may not have several features of the functionality provided by Visual Studio to automatically generate code for model binding methods. When working on your own projects, automatic code generation can save time and get an idea of how to implement the operation. This section describes the code generation feature. This section is for informational purposes only and does not contain any code that needs to be implemented in your project.

When you set a value for the SelectMethod, UpdateMethod, InsertMethod, or Delete Method property in the markup code, you can choose to create a new method parameter.



Visual Studio not only creates a method in the code of the program part with the correct signature, but also creates an implementation code to simplify the operation. If you first set the ItemType property before using the code generation function, the generated code will use that type for operations. For example, when you set the UpdateMethod property, the following code is automatically generated:

C#

// The id parameter name should match the DataKeyNames value set on the control

public void studentsGrid\_UpdateItem(int id)

{

ContosoUniversityModelBinding.Models.Student item = null;

// Load the item here, e.g. item = MyDataLayer.Find(id);

if (item == null)

{

// The item wasn't found

ModelState.AddModelError("", String.Format("Item with id {0} was not found", id));

return;

}

TryUpdateModel(item);

if (ModelState.IsValid)

{

// Save changes here, e.g. MyDataLayer.SaveChanges();

}

}

**REVIEW QUESTIONS**

1.A set of programming languages supported in Visual Studio 2013.

2.Some attributes and their properties are Compare,Range,RegularExpression.

3.Key terms: JavaScript, Visual C#, Visual Basic

**Task**

1. Apply model bindings.

2.Develop a simple visual C++ console project in Visual Studio 2013.

3.Apply the attributes in the model validation.

4.Develop a simple animated web page with changing content using JavaScript.

5.Develop a simple Windows project in Visual Basic in Visual Studio 2013.

1. **Using asynchronous methods in ASP.NET**

One of the key innovations of the latest versions of the .NET framework has become asynchronous. Although the framework used to allow the use of asynchronous methods, with the advent of the Task Parallel Library, the work with asynchronous code was extremely simplified, and the format of the work changed. New features have been added to create asynchronous methods using new keywords, such as as async and await.

When creating a new controller, we can already specify in the settings how we need the controller - synchronous or asynchronous. By default, Visual Studio adds standard controllers to the project, whose methods typically return an ActionResult object. But if we choose MVC 5 Controller with views, using Entity Framework, when adding a controller to the Controllers folder, a special field in the configuration window of the new controller will allow us to specify that the new controller will contain asynchronous methods:

Why do we need asynchronous methods in controllers? Asynchronous methods allow you to optimize the performance of your application and are primarily designed to handle requests that take or may take quite a long time, such as accessing a database or accessing an external network resource to obtain a large amount of data. Using asynchronous methods allows your application to run other queries in parallel with asynchronous code.

To understand the difference between synchronous and asynchronous methods, consider how IIS handles incoming requests. The web server supports a pool of threads that serve requests. When a user accesses a web resource, IIS allocates a thread from the pool to service the request. And while this thread will not process the request intended for it, it cannot process other requests.

However, assume that the controller method processing the request must complete the request to another resource or database. A request to a network resource or database itself may take some time. In synchronous processing, the thread processing the request is temporarily blocked until the network resource or database returns the data we need.

And if the request is blocked for a very long time, IIS starts to use new threads to serve other incoming requests. However, there are limits to the total number of threads. When the number of threads reaches the limit, incoming requests are placed in the waiting queue. However, there is a limit to the number of requests in the queue. And when that number exceeds the limit, IIS simply rejects all other requests with a 503 status code (Service Unavailable).

In asynchronous processing, the thread does not wait for the database to return data to it, but starts processing a request from another user. But when, finally, from a network resource or database will come the necessary data, the flow returns to the processing of the previously processed request in normal mode.

Let's go directly to the code. To create asynchronous methods, you use the async and await modifiers, which allow you to perform long-running operations without blocking the main thread.

Let's compare the example of calling synchronous and asynchronous method:

|  |  |
| --- | --- |
|  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  using System.Web.Mvc;  using AsyncContollers.Models;  using System.Threading.Tasks;  using System.Data.Entity;   namespace AsyncContollers.Controllers  {      public class HomeController : Controller      {          BookContext db = new BookContext();            public ActionResult Index()          {              IEnumerable<Book> books = db.Books;              ViewBag.Books = books;              return View();          }           // asynchronous method          public async Task<ActionResult> BookList()          {              IEnumerable<Book> books = await db.Books.ToListAsync();              ViewBag.Books = books;              return View("Index");          }      }  } |

Both methods perform the same operation - data extraction from the database and get identical results. But if the first synchronous Index method represents a familiar record for us, the asynchronous BookList method already looks unusual.

This method returns not an ActionResult object, but a Task<ActionResult>object. Task represents an asynchronous operation that runs for a long time.

In addition, to designate a method as asynchronous, the async keyword is placed before the returned type.

The third key point is the use of the await keyword. It is used in an asynchronous method to suspend the execution of this method until the awaited task completes. In our case, this task is to obtain data from the database.

But also keep in mind that await is used with methods that return a Task object. Therefore, the await db method is used to get data from the database.Books.ToListAsync (), which also extracts data from the database, but in asynchronous mode.

When should asynchronous methods be used? First of all, they are preferable to use when querying the database, to external network resources, but in the end, that better synchronicity or asynchrony is solved by the developer on the basis of a specific task.

**REVIEW QUESTIONS**

1.Modifiers are used to create asynchronous methods.

2.Compare synchronous and asynchronous метода.

3.ASP.NET MVC 4 and asynchronous C#capabilities.

4.Asynchronous operations and the lifecycle of the page.

5.What life-cycle events ASP.NET can be asynchronous?

**TASK**

1.The RegisterAsyncTask Method.

2. AddOnPreRenderCompleteAsync Method

1. **Web deployment ASP.NET**

In the simplest case, deploying a web application ASP.NET reduced to copying the directory structure for the application on the target computer and setup the environment. For simple applications, this is almost always the case. However, if your application uses databases or accesses other resources, you will have to perform a number of additional steps. The following are common factors that require additional configuration steps:

* Copy all the required application files to the target computer. Beyond that, you do not need to do anything. However, if you use global assemblies that are accessed through the GAC, you must ensure that they exist. If they do not exist, they must be installed using the gacutil command-line utility.exe, which is part of the .NET Framework.
* Create and configure a database for the application. It is important not only to create the database and its tables, but also to set up accounts to log on to the database server and database users. Do not forget that if you use integrated authentication to connect to a SQL Server database, the account under which you are running ASP.NET (application pool account or aspnet\_wp account.exe), you will need to configure the application as a database user. The web deployment approach can simplify database deployment. If you use a different approach, you must manually configure and populate the databases.
* Configure IIS as required by your application. Create the required application pools, make the application directory available for sharing as a virtual directory, and configure the virtual directory properly.
* Set the Windows account rights for the workflow user. The user on whose behalf the workflow is started (w3wp.exe), needs read access to application directories. If your application accesses other resources, such as the system registry or the event log, the workflow account will need to be configured with permission to access those resources.
* If you want to handle any URLS with file name extensions that are different from the extensions registered during installation ASP.NET by default, add IIS file mappings.
* Configure ASP.NET (and IIS 8.0 application-specific settings) in the web file.config for production environments. In other words, add (or modify) any desired connection strings and application settings, as well as security and authorization settings, session state settings, and globalization settings.
* In some cases, you must also modify the machine file.config. For example, if you are working in a web hosting environment and your application is running on multiple web servers for load balancing purposes, you must synchronize any encryption keys that are used to encrypt forms authentication mandates or view state on all of these computers. These keys are stored in the machine file.config and must be the same on each computer in the web farm so that one computer can decrypt the information encrypted by another computer that previously processed the request.
* The primary task is to deploy application content to an IIS 8 server. This article assumes that you have administrator rights and that you are logged on to the server as an administrator. If you are using a shared server or hosting provider, the provider will send you the account information that should be used when you deploy the website. For simplicity, we will assume that you are working with an administrator account, but if you are managing your own server, you should consider using a less privileged account.

## Deploy by copying files

* The easiest way to deploy a website is to copy files from a workstation to a server. Despite its simplicity, this approach requires direct access to the server. It is for this reason that some IT departments and companies that provide hosting services do not support this option. But it can be the simplest when you manage your own server or have special arrangements with the hosting company.

### Preparing IIS

* Before you deploy a website, you must prepare IIS. The main decision you need to make is where to place the content and how it affects the final URL. Let's start with the obvious approach-assume that you want the URL for the contents of this example to be as follows:
* http: / / <server name\_\_>: 80/Website Deployment/File Copy
* IIS needs to be prepared so that there is where to copy our file. In IIS Manager, select Default Web Site. As its name implies, it is the default site on the server. Click the right mouse button and from the context menu, select Explore (Windows Explorer) to open a Windows Explorer window for the default IIS directory, which is inetpub\wwwroot folder on the system volume (usually C:\).
* Create the WebsiteDeployment directory and create the FileCopy directory (to ensure that the path inetpub\wwwroot\WebsiteDeployment\FileCopy exists). Close the Explorer window to return to IIS Manager. Right-click the Default Web Site entry and select Refresh from the context menu to see the new directory.

### Web site

To demonstrate this deployment technique, we created a very simple website. The source code contains a single form ASP.NET with one label:

<**form** id="form1" runat="server">

<**div** class="style1">

<**h1**><**strong**>Простой веб-сайт</**strong**></**h1**>

</**div**>

<**br** />

Версия .NET Framework: <**asp:Label** ID="Label2" runat="server" Text="Label1" />

</**form**>

The code in this form defines the label text that displays the version of the .NET Framework that is used to serve this site:

**public** **partial** **class** \_**Default** : **System**.**Web**.**UI**.**Page**

{

**protected** **void** **Page\_Load**(**object** sender, EventArgs e)

{

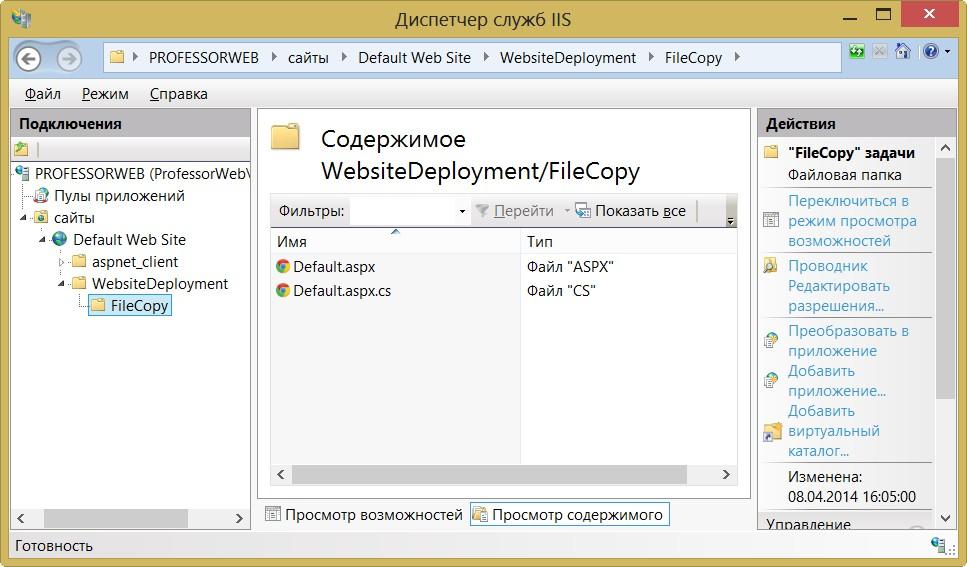
Label2.Text = System.Environment.Version.Major.ToString();

}

}

### Deploying a web site

* When you use this deployment technique, you simply copy the files to the created directory. Move the website files to the server in any appropriate way - via a shared network drive, a removable USB drive, etc. - and copy the Default files.aspx and Default.aspx.cs to the FileCopy directory created on the server.
* When the files are copied, return to the IIS Manager window on the server, right-click the FileCopy folder in the tree view, and select Refresh from the context menu. At the bottom of the screen, click the Content View button. In the center of the window, you should see two website files, as shown in the figure below:



* This is the most important part of this deployment technology - you must first create a directory structure that represents the URL that you want, and then copy the website files to the finished directories. Let's see how it looks in practice. To do this, select FileCopy in IIS Manager and click the Browse link on the right side of the window. A web browser opens with the downloaded URL of the folder you created.

By looking at the URL address, you can verify that the desired result was obtained. The browser downloaded the website from the following address:

http://localhost/WebsiteDeployment/FileCopy/

Remember that localhost is a special name for the current computer, and a URL that does not specify a port will use port 80. (In my case, I moved the site to port 8080 for the reasons mentioned in the previous article). You can verify this by directing your browser to the following URL:

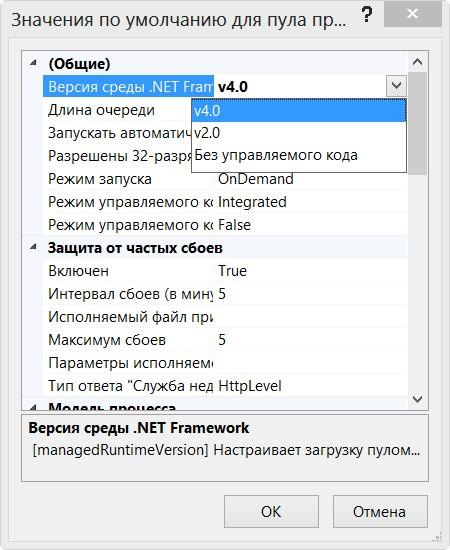
http://professorweb/WebsiteDeployment/FileCopy/

The result will be completely similar to the previous one (professorweb in this case is the server name).

### Configuring the deployment

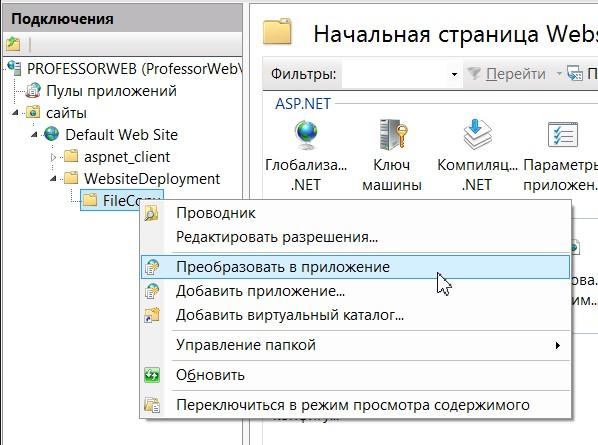
You may have noticed that the version of the .NET Framework reported in the figure above is 4. To change the target version of the .NET Framework, you will need to change the settings for the default application pool. Don't think about what application pools are yet - they will be discussed in detail later.

In the IIS Manager window, expand server, and then click Application Pools. Click the Set Application Pool Defaults link (Define application pool defaults) on the right side of the window. In the Application Pool Defaults (application pool defaults) dialog box that opens, change the .NET Framework Version (.NET Framework version) setting to the required value:



Go back to your browser and reload the page.

IIS will also need to specify that the deployed site is an application. It's not required, but when deploying applications ASP.NET almost always be desirable - aktiviziruyutsya session state and other functional tools ASP.NET. Click the right mouse button on the folder FileCopy in the area of Connections (Connections) and in the context menu select Convert to Application (convert to application), as shown in the figure below:



The Add Application (Add application) dialog box appears. You can change the application pool you are using by clicking the Select button. You can use the Connect as buttons to configure the user account that IIS will use to access site content... (Contd. as...) and Test Settings... (Test settings...). For now, just click on the OK button. You may have to select Refresh from the View menu (or, as is often the case, close and reopen IIS Manager), but now the FileCopy record icon in the tree view should change.

## Deploy with Visual Studio

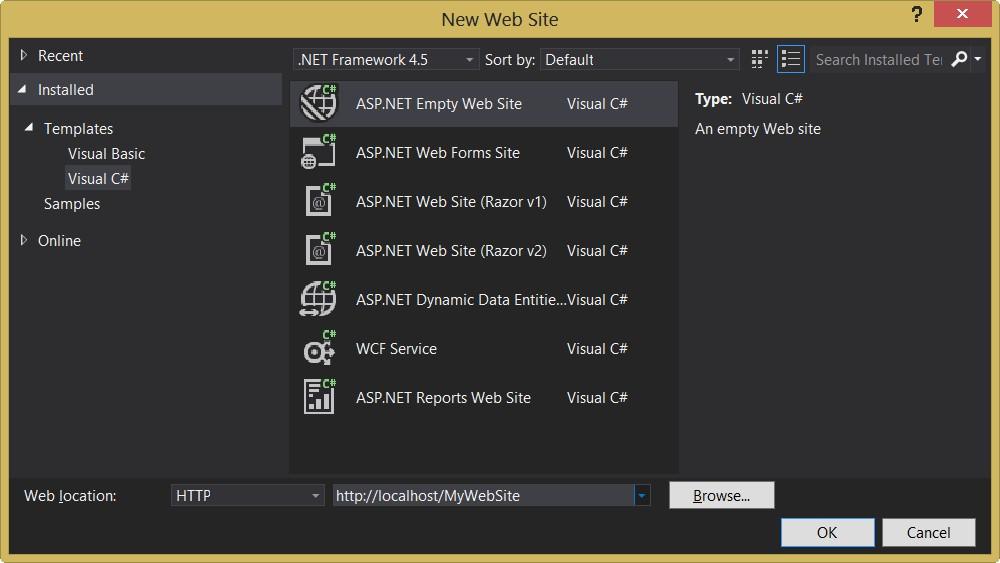
Visual Studio offers built-in capabilities to simplify web application deployment, making it easier to design complex web sites. Visual Studio includes features that integrate with IIS and allow you to create virtual directories from the comfort of your development environment. Visual Studio offers several options for integration with IIS:

* You can create a virtual directory when you create a new project.
* You can use the Copy Web Site (Copy web site) function to transfer an existing web site to a virtual directory.
* You can use the Publish Web Site feature to compile your site and upload it to a new address.

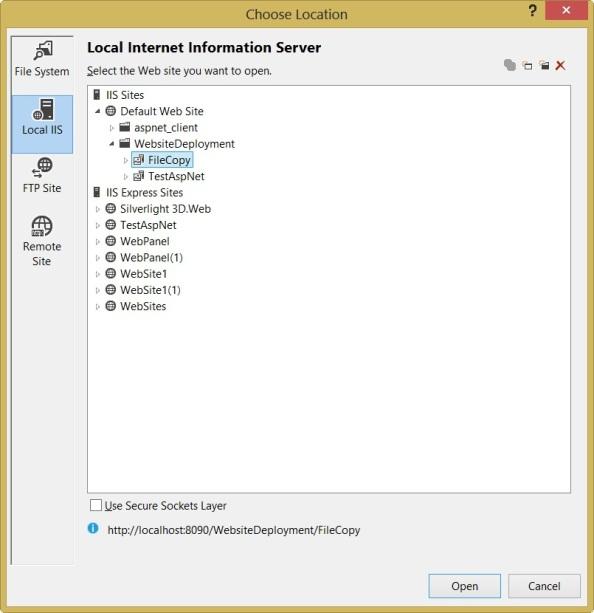
**Create a virtual directory for the new website**

When you create a website in Visual Studio, you can create a virtual directory for that website at the same time. If you do this, Visual Studio will not use the built-in IIS Express test server. Instead, all your requests will be handled by the full version of IIS.

To create an IIS virtual directory, you must first run Visual Studio as an administrator. Then select File --> New --> Web Site and in the new Web Site dialog box that opens, select HTTP for the site location (instead of the file system). You can insert a URL string. For example, if you specify http://localhost/MyWebSite, Visual Studio will create a My WebSite directory in the virtual directory C:\Inetpub\Wwwroot used by default:



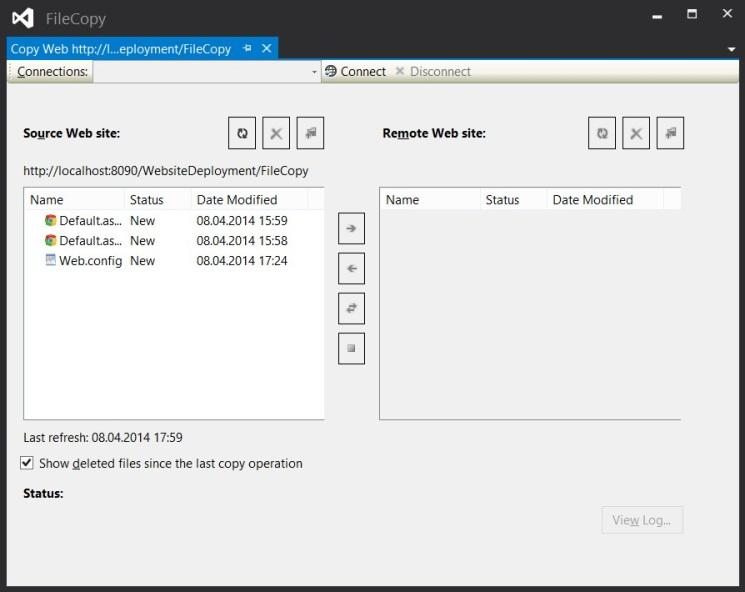
This approach is not the best because you will not be able to configure some settings, such as default pages, custom errors, etc. Instead of creating a new virtual directory using Visual Studio, it would be better to choose an existing directory to IIS, which in turn can be configured using IIS Manager. To do this, click the Browse button and select the desired virtual directory:



In this window, you can add or remove a virtual directory using the buttons at the top right.

**Copying a web site**

Visual Studio also includes a quick and easy way to transfer web application files. You just need to select WebSite --> Copy Web Site from the menu. A new Visual Studio dialog box will open, which will be familiar to anyone who has dealt with various IDE programs to work with sites such as Dreamweaver or Expression Web:



This dialog box consists of two lists of files. On the left are the files in the current project (on the local hard drive). On the right are the files in the target location (remote web server). The first time you open this window, you won't see anything on the right because you didn't specify a target. You must click the Connect button at the top to provide this information. When you do this, Visual Studio opens a familiar dialog box that looks almost the same as what you saw when you created the virtual directory for the new project. In this window, you can select one of the places where you want to copy the website:

***File System***

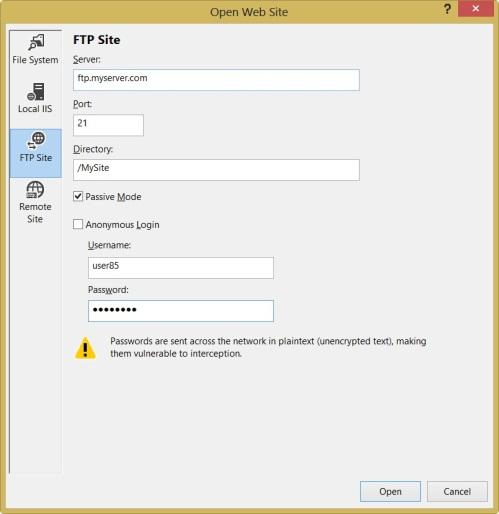
This is the easiest choice - you copy the site files inside the current file system.

***Local IIS***

Allows you to select virtual directories on the local IIS server. This is useful if the site will need to be placed in a different directory.

***FTP Site***

Provides access to a remote site via FTP. Many hosting providers offer access to websites via FTP, so thanks to this setting you can immediately deploy your website to a remote web server without using the services of various FTP programs like TotalCommander:



***Remote Web Server***

This option offers a URL connection to the site using the FrontPage Extensions extension.

Once you have selected the appropriate destination, click Open. Visual Studio will try to connect to the remote site and retrieve a list of files.

**REVIEW QUESTIONS**

1.Deploy your application.

2.You deploy a web site by using IIS.

3.Deploy and update individual files in your application ASP.NET from the command line.

4.Methods Add and Insert

**Task**

1.Create a virtual directory for the new website.

2.Create examples up a web site.

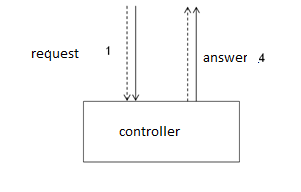
3.Create the Add and Insert methods examples

## II Model-View-Controller (MVC)

## MVC for web

#### Consider the architectural pattern of MVC (Model, View, Controller) in application to web development, "in its pure form", without involving any additional, not related to MVC structures and patterns. We will move from simple to complex, so we will not consider, for example, the further development of the MVC pattern (Hierarchical MVC). Although HMVC is undoubtedly much more interesting for web application development, it does not eliminate the need to understand "normal" MVC.Article in Wikipedia (namely there, apparently, most often get those who are just starting to learn MVC), is replete with inaccuracies and vague formulations, the definition itself, in fact, is incorrect, and the above scheme simply does not correspond to the one that is used in the web in General and in the development of PHP – in particular.The most correct definition of the MVC pattern I found here:

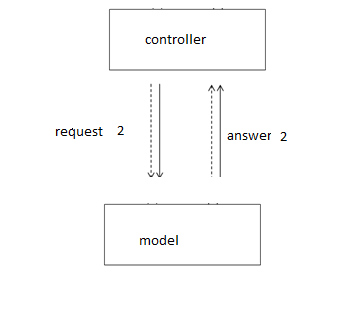
#### The MVC design pattern involves separating application, user interface, and control logic data into three separate components: Model, View, and Controller – so that the modification of each component can be carried out independently.Let us clarify that the term "component" in this case has no connection with the components of some popular CMS or frameworks, and Bitrix components, for example, are built from all three components of MVC.In the above definition, the component should be understood as a separate part of the code, each of which plays one of the roles of the Controller, Model or View, where the Model is used to extract and manipulate application data, the View is responsible for the user's visible display of this data (that is, when applied to the web, it forms the HTML/CSS that the server sends to the user's browser), and the Controller manages all this orchestra. Let's look at the classic scheme of a web application:



In this and subsequent figures, dashed lines show control information (such as, for example, the ID of the requested blog entry or product in the store), and solid – the actual application data (which can be stored in the database, or in the form of files on disk, or even, perhaps, in memory – this issue lies outside the MVC pattern). When applied to the web, the request and response go over HTTP, so we can assume that in this figure the dotted line indicates the headers of the HTTP request and response, and solid lines-their bodies. After receiving the Request 1, the Controller analyzes it, and depending on the results of processing can give the following answers (why the answer is number 4, it becomes clear from the following figures):

1. Immediately give an error response (for example, when requesting a non-existent page to give only HTTP-header " 404 Not found»)

2. If the received Request 1 is found to be correct, then, depending on whether it is a request to view or modify the data, the Controller calls the appropriate Model method, such as Save or Load



Important note: the concept of MVC is not only not tied to any particular programming language, it is also not tied to the programming paradigm used. That is, you can design your application for MVC, without using OOP, and design the Product Model for the online store in this way:

<?

mixed Product\_Load (int $id) { ... }

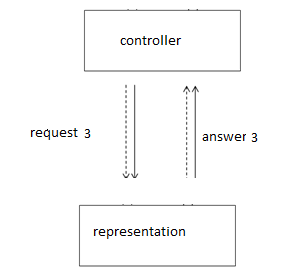
// returns an associative array with Product data or FALSE on failure

bool Product\_Save (array $data) { ... }

// returns TRUE if $data is saved successfully, or FALSE if it fails

?>

So, depending on the Response Model 2, the Controller decides which View to call to form the final response to the initial 1: 3.1 Request. In case of failure – View for error message 3.2. If successful, a View to display the requested data or a message that the data was successfully saved (if Query 1 was to modify the data).



The question of who should check the validity and access rights of the input data (Controller or Model) is a subject of quite a lot of controversy, since the MVC pattern does not describe such details. This means that in this matter the choice is yours (or for you it was made by the authors of your favorite framework or CMS). In our practice, we follow this approach: the controller checks the input data for "common" (ie. regardless of the specific request) correctness, compliance with the Model requirements to the validity of the stored data is checked by the appropriate method of the Model, and access rights – the Access method of a separate class User. To call a View in PHP, sometimes a special class (or even several classes) is designed, for example, View (this is often found in the MVC descriptions in the implementation of a particular framework), but this is not a requirement of MVC. Also, View files are often called templates, and when using so-called templates, the role of the View is played by the template engine itself, and templates (i.e. files containing HTML markup directly) in some frameworks are called layouts. Not quite clear the previous paragraph? Forget it, because we have each View – it's just a separate PHP-file, and PHP itself is designed so that if we use to execute a Query 3 include statement, Answer 3 and Answer 4 (remember that this is the answer to Query 1?) are given to the browser automatically, by means of PHP. Let's look at an example. We have two Presentation options (templates) in which <!-- HTML.header --> will mean the HTML code that represents the main content in the generated web document (i.e. contains the doctype tag, the head container, the code of the page header, etc.), and <!-- HTML.footer -- > - roughly the same, only for the footer pages. Listing 1. Product template.tpl.php displays the data about the Product (which by the time of his call already contains the object $product):

<!-- HTML.header -->

<h1><?=$product->Title;?></h1>

<p>Цена:<b class="price"><?=$product->Price;?></b></p>

<p class="description"><?=$product->Description;?></p>

<!-- HTML.footer -->

#### Listing 2. The error template.tpl.php displays an error message (which is contained in the $error variable):

<!-- HTML.header -->

<h1 class="error">Ошибка: <?=$error;?></h1>

<!-- HTML.footer -->

#### Listing 3. Product controller.php, which is used to display the Product, will look something like this:

<?

include 'product.class.php';

function Error ($error) {

// displays an error message and shuts down the controller, something like this:

header ('correct error status, e.g. 400 or 404'));

$error = 'an Appropriate error message to the user, for example, the Page does not exist';

include ' error.tpl.php'; / / template to display the error

exit;

}

if (!$id = ...) // check" General " validity of Request 1

error(...);

// permissions check

if (!$user->Access(...))

error(403);

if (!$product = Product::Load($id)) // Request 2 and Response analysis 2

error ('there's probably a DB error here'));

include ' product.tpl.php'; / / Query 3 and Answers 3 and 4

?>

Those who love beautiful and optimized code may notice that HTML blocks.header and HTML.footer are duplicated in both templates (aka Views) error.tpl.php and product.tpl.php, and probably want to make them in the product Controller.save.php:

<!-- HTML.header -->

<?

// / / Main controller code from Listing 3

?>

<!-- HTML.footer -->

and to break the basic rule of MVC in this way-separate the Controller, Model, and View.

In this Chapter, we will look at three tools that should be in the Arsenal of each MVC programmer: dependency injection container (DI), a framework for unit testing and a tool for moking (mock-objects).

We have chosen three specific implementations of these tools for this book, but there are many alternatives for each type of tool. If you don't like the ones we use, don't worry. There are so many of them that you will surely find something that is right for you and will facilitate your workflow.

It is simple, elegant and easy to use. There are more complex alternatives, but we like the way that Ninject works, with a minimum of customization. We consider patterns to be the starting point, not the law, and we found it easy to customize our DI with Ninject to different projects and workflows. If you don't like Ninject, we recommend trying Unity, which is one of Microsoft's alternatives.

For unit testing, we'll use what's built into Visual Studio. We're used to using NUnit, which is the most popular .NET framework for unit testing, but Microsoft has made a big leap in improving support for unit testing in Visual Studio (and currently includes it in free editions of Visual Studio). As a result, the unit test frameworks are tightly integrated into the rest of the IDE and they have become quite good.

The third tool chosen by us is Moq, which is a set of moking tools. We use Moq in creating interface implementations for use in our unit tests. Programmers either love or hate Moq; the third is not given. Either you find the syntax elegant and expressive, or you curse it every time you try to use it. If you don't like it, pay attention to Rhino Mocks, which is a good alternative.

We will talk about each of these tools and show their main features. We will not give exhaustive information on these tools, because each of them can be written in a separate book, but we will give you enough information to get you started with them, and, most importantly, follow the examples in the rest of the book.

Note

This Chapter assumes that you will want to take full advantage of MVC, including an architecture that supports many tests, and here we will focus on building applications that are easy to change and maintain. We love these tools and won't build apps without them, but we know that some readers just want to understand the capabilities that MVC offers and not go into the development of philosophy and methodology. We will not try to dissuade you – it is your personal decision and you know what you need to create your projects most effectively. We assume that you will at least run through this Chapter briefly to see the available features, but if you are not a unit testing type of fan, you can skip to the next Chapter and see how to build a real MVC application.

1.Концепция MVC.

2.MVC для веб приложений.

3.Выбор типа MVC 4 проекта.

4.Архитектурный MVC паттерн

5.Доменные модели и хранилища.

1. Создайте простой ASP.NET MVC проекта

## MVC usage experience, conclusions

One of the key innovations of the latest versions of the .NET framework has become asynchronous. Although the framework used to allow the use of asynchronous methods, with the advent of the Task Parallel Library, the work with asynchronous code was extremely simplified, and the format of the work changed. New features have been added to create asynchronous methods using new keywords, such as as async and await.

When creating a new controller, we can already specify in the settings how we need the controller - synchronous or asynchronous. By default, Visual Studio adds standard controllers to the project, whose methods typically return an ActionResult object. But if we choose MVC 5 Controller with views, using Entity Framework, when adding a controller to the Controllers folder, a special field in the configuration window of the new controller will allow us to specify that the new controller will contain asynchronous methods

Why do we need asynchronous methods in controllers? Asynchronous methods allow you to optimize the performance of your application and are primarily designed to handle requests that take or may take quite a long time, such as accessing a database or accessing an external network resource to obtain a large amount of data. Using asynchronous methods allows your application to run other queries in parallel with asynchronous code.

To understand the difference between synchronous and asynchronous methods, consider how IIS handles incoming requests. The web server supports a pool of threads that serve requests. When a user accesses a web resource, IIS allocates a thread from the pool to service the request. And while this thread will not process the request intended for it, it cannot process other requests.

However, assume that the controller method processing the request must complete the request to another resource or database. A request to a network resource or database itself may take some time. In synchronous processing, the thread processing the request is temporarily blocked until the network resource or database returns the data we need.

And if the request is blocked for a very long time, IIS starts to use new threads to serve other incoming requests. However, there are limits to the total number of threads. When the number of threads reaches the limit, incoming requests are placed in the waiting queue. However, there is a limit to the number of requests in the queue. And when that number exceeds the limit, IIS simply rejects all other requests with a 503 status code (Service Unavailable).

In asynchronous processing, the thread does not wait for the database to return data to it, but starts processing a request from another user. But when, finally, from a network resource or database will come the necessary data, the flow returns to the processing of the previously processed request in normal mode.

Let's go directly to the code. To create asynchronous methods, you use the async and await modifiers, which allow you to perform long-running operations without blocking the main thread.

Let's compare the example of calling synchronous and asynchronous method:

|  |  |
| --- | --- |
|  | using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  using System.Web.Mvc;  using AsyncContollers.Models;  using System.Threading.Tasks;  using System.Data.Entity;   namespace AsyncContollers.Controllers  {      public class HomeController : Controller      {          BookContext db = new BookContext();            public ActionResult Index()          {              IEnumerable<Book> books = db.Books;              ViewBag.Books = books;              return View();          }            public async Task<ActionResult> BookList()          {              IEnumerable<Book> books = await db.Books.ToListAsync();              ViewBag.Books = books;              return View("Index");          }      }  } |

Both methods perform the same operation - data extraction from the database and get identical results. But if the first synchronous Index method represents a familiar record for us, the asynchronous BookList method already looks unusual.

This method returns not an ActionResult object, but a Task<ActionResult>object. Task represents an asynchronous operation that runs for a long time.

In addition, to designate a method as asynchronous, the async keyword is placed before the returned type.

The third key point is the use of the await keyword. It is used in an asynchronous method to suspend the execution of this method until the awaited task completes. In our case, this task is to obtain data from the database.

But also keep in mind that await is used with methods that return a Task object. Therefore, the await db.Books.ToListAsync () method is used to get data from the database, which also extracts data from the database, but in asynchronous mode.

When should asynchronous methods be used? First of all, they are preferable to use when querying the database, to external network resources, but in the end, that better synchronicity or asynchrony is solved by the developer on the basis of a specific task.

# 

## Migrating applications ASP.NET MVC in Windows containers

When you run an existing .NET Framework-based application in a Windows container, you do not have to make any changes to the application. To run an application in a Windows container, create a Docker image with the application and run the container. This section shows how to get an existing application ASP.NET MVC and deploy it in a Windows container.

Start with an existing application ASP.NET MVC and build published resources by using Visual Studio. Docker is used to create an image that contains and runs your application. You will go to the site running in the Windows container and check whether the application is running.

Understanding this article requires a basic understanding of Docker. For information about Docker, see understanding Docker.

The app that will run in the container is a simple website that randomly answers questions. This is a basic MVC application without authentication and database storage. It allows you to focus on moving the web layer to the container. The following topics show how to move and manage persistent storage in container applications.

Moving an application consists of the following actions:

1. The creation of publish jobs for Assembly resource image.
2. Create a Docker image in which to run the application.
3. Start the Docker container in which the image will run.
4. Check the application using the browser.

The finished app is on GitHub.

Preliminary requirements

The development computer must have the following software:

Windows 10 anniversary update (or later) or Windows Server 2016 (or later)

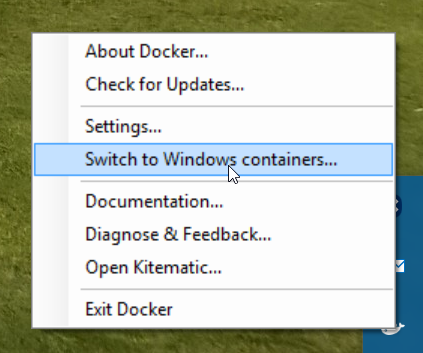
Docker for Windows-Stable version 1.13.0 or 1.12 Beta 26 (or later)

Visual Studio 2017

Important!

If you are using Windows Server 2016, follow the instructions to deploy the container host to Windows Server.

After you install and run Docker, right-click the icon in the taskbar, and then click switch to Windows containers. This is required to run Windows-based Docker images. This command takes a few seconds to complete:

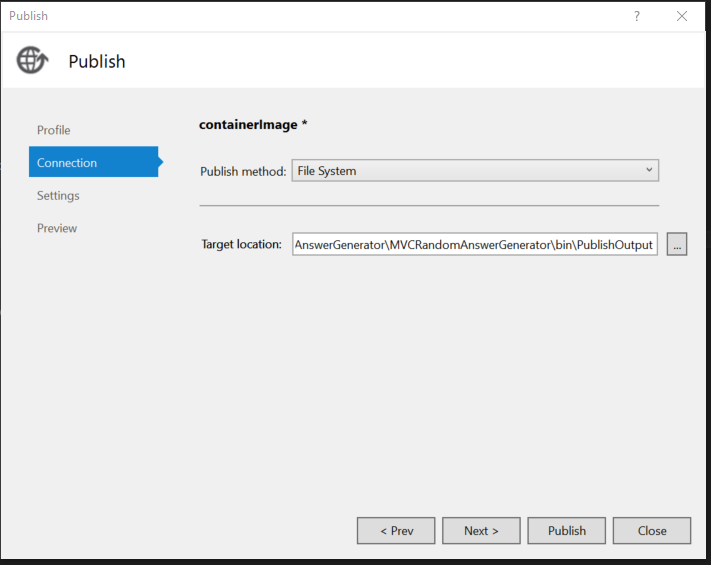


Publish script

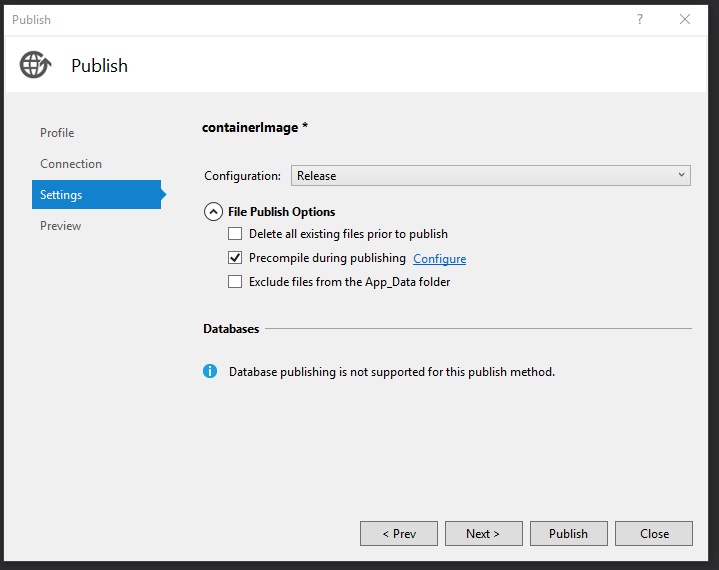
Bring together all the resources that you want to load into the Docker image. There is a Publish command to create an application publishing profile in Visual Studio. In this profile, all resources will be gathered in one directory tree, which you will need to copy to your target image later in this tutorial.

Stages of publication

1. Right-click the web project in Visual Studio, and then click Publish.
2. Click the User profile button, and then select File system as the method.
3. Selected directory. According to the rules, the loaded sample uses bin\Release\Publish Output.



Open the file publish Settings section of the Options tab. Select Precompile during publishing (precompile at publish time). This optimization means that views will not be compiled in the Docker container, but will instead be copied to precompiled views.



Click Publish, and Visual Studio will copy all the required resources to the destination folder.

Building the image

Define a Docker image in Dockerfile. Dockerfile contains instructions for the base image, additional components, the application you want to run, and other configuration images. Dockerfile contains the input parameters for the docker build command that creates the image.

You will create an image based on the microsoft/ASP net image located on Docker Pub.The microsoft/ASP net base image is a Windows Server image. It contains Windows server Core, IIS, and ASP.NET 4.7.2. When you run this image in a container, it automatically uses IIS and installed websites.

The Dockerfile that creates the image is as follows:

console

# The `FROM` instruction specifies the base image. You are

# extending the `microsoft/aspnet` image.

FROM microsoft/aspnet

# The final instruction copies the site you published earlier into the container.

COPY ./bin/Release/PublishOutput/ /inetpub/wwwroot

There is no ENTRYPOINT command in Dockerfile. It is not needed. When you start Windows Server with IIS, the IIS process is an entrypoint that has been configured to run in the aspnet base image.

Run the Docker build command to create an image that runs the application ASP.NET ahhh! To do this, open a PowerShell window in the project directory and type the following command in the solution directory:

console

docker build-t mvcrandomanswers .

This command will create a new image according to the instructions in the Dockerfile naming (-t — tag) image as mvcrandomanswers. This may include getting the base image from Docker Hub. The application is then added to the image.

After you run this command, you can run the docker images command to view information about the new image:

console

REPOSITORY TAG IMAGE ID CREATED SIZE

Mvcrandomanswers latest 86838648aab6 2 minutes ago 10.1 GB

The IMAGE ID on your computer will be different. Now let's run the application.

Running the container

To run the container, run the following docker run command:

console 213

docker run -d --name randomanswers mvcrandomanswers

The-d argument instructs Docker to start the image in disconnected mode. This means that the Docker image runs in isolation from the current shell.

In many examples of docker may appear -p to map the ports of the container and the host. The default aspnet image is already configured to listen on port 80 and grant access to it.

The --name randomanswers argument contains the name of the running container. You can use this name instead of the container ID in most commands.

mvcrandomanswers is the name of the image to run.

Browser check

After starting the container, connect to the running container using http://localhost in the example. Enter this URL in the address bar of your browser and you will see the site working.

 Note

Some VPN programs or proxy servers may be preventing you from going to your site. You can temporarily disable them to make sure that the container is running.

The samples directory on GitHub contains a PowerShell Script that executes these commands for you. Open a PowerShell window, navigate to the solution directory, and then type:

console

./run.ps1

The above command creates an image, displays a list of images on the computer, and starts the container.

To stop the container, run the docker stop command:

console Copy

docker stop randomanswers

To delete a container, run the docker rm command:

console

docker rm randomanswers

# III The web API

# 1. Getting started with ASP.NET Web API

**To use the Web API function, you must add a special controller to your MVC Framework application. This controller is an API Controller, has two distinctive characteristics:**

**1. Action methods return model, rather than a ActionResult object.**

**2. The action methods are selected based on the HTTP method that was used in the request.**

**Model objects created by API controller action methods are converted to JSON format and sent to the client. API controllers are designed to deliver web data to services, so they do not support views, layouts, or any other features that create HTML for display in the browser. The API controller can support requests from any client, but it is most often used to serve Ajax requests from web applications; this is what we will demonstrate in this Chapter.**

**As we showed in Chapter 21, action methods that return JSON data to support Ajax can be created in regular controllers, but the API controller offers an alternative approach. It allows you to separate the actions that return data from the actions that return views, and thus makes it easier and faster to create a General-purpose Web API.**

**Tip**

**You are not required to use API controllers (like almost all other MVC Framework features). In real projects we use techniques from Chapter 21 along with API controllers. Typically, we use API controllers when we need to create a lot of actions that return JSON, or when we are working on a project that does not have an HTML component and only provides data services.**

**To explain how API controllers work, the easiest way is to create an MVC Framework application that uses these controllers; this is what we will do later in this Chapter. However, the Web API is a fairly simple function, and relies on many of the MVC features that we've covered in detail in previous chapters, so we'll spend most of our time building the application, and at the end we'll just add an API controller to it.**

**On the one hand, it's even good. This means that you already know all the functions that the API controller relies on. But the example still turns out to be weird, because we'll have to create a lot of standard MVC application components before we get to the API controller. In addition, we'll have to write JavaScript code that sends Ajax requests. We'll use jQuery to do this, but we won't go through the code in detail, just explain what each section of code does and how it interacts with the web service we'll create.**

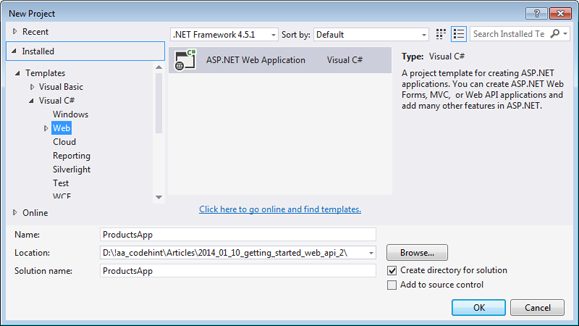
HTTP is not just for transferring web pages. It is also a powerful platform for building an API that can provide services and data. HTTP is simple, flexible and as common as POSSIBLE. Almost all platforms have libraries that work with HTTP, which means that HTTP services can work with many different clients, including browsers, mobile devices, and regular desktop applications.

ASP.NET Web API is an environment for building web API based on .NET Framework. In this article, we'll walk through the process of building a web API that returns a list of products. On the web page (client), we will use jQuery to show the list of products.

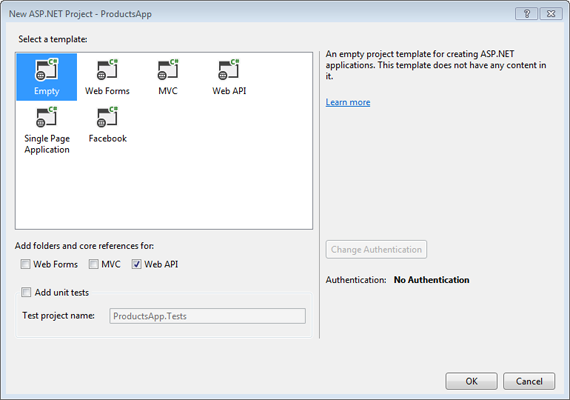
Create a Web API project

Start Visual Studio 2013, select New Project on the start page or from the main menu.

In the window on the left, select Installed->Templates->Visual C# - >Web. Choose from the list of templates ASP.NET Web Application. Let's call the project "ProductsApp" and click OK.



In the new dialog box ASP.NET Project, select the Empty template. Also note the web API checkbox in the section below "Add folders and core references for" (Add folders and main assemblies for).



Of course, you can create a Web API project using a web API template. But this type of project uses ASP.NET MVC to create helper API description pages. We use an empty template intentionally to show the Web API without using MVC. In General, it is not necessary to have knowledge in ASP.NET MVC to build Web API projects.

Add the model

A model is an object that represents application data. ASP.NET Web API automatically serializes the model in JSON, XML or some other format, and then writes the serialized data to the message body of the HTTP response. As soon as the client receives the serialized data, it immediately deserializes it into an object. Most clients recognize JSON and XML. Moreover, the client in the request can specify in what format he wants to receive the data, this is indicated in the Accept header in the body of the HTTP request.

Create a simple model to represent product data.

In the Solution Explorer project view window, right-click on the Models folder. In the context menu that opens, select Add - > Class.



Let's call the class "Product". Add the following properties to the new class:

namespace ProductsApp.Models

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public string Category { get; set; }

public decimal Price { get; set; }

}

}

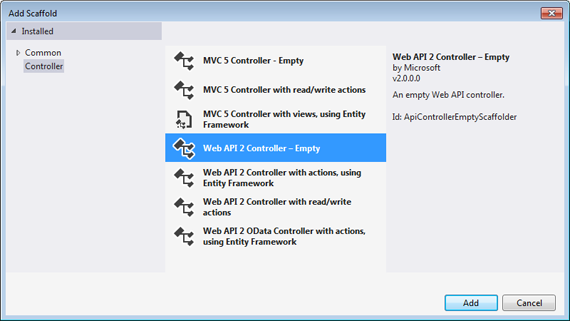
You are adding a controller

In the Web API, a controller is an object that manages HTTP requests. We will create a controller that will return both a list of products and each product individually by a given ID.

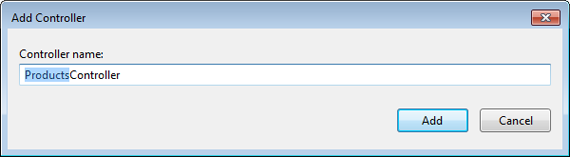
In Solution Explorer, right-click to open the context menu above the Controllers folder. Choose Add - >Controller.



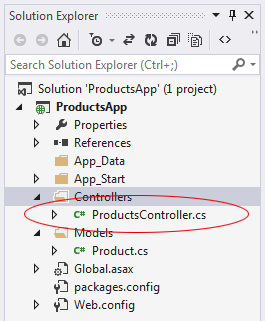
Next, in the Add Scaffold dialog box, select Web API 2 Controller-Empty. Click Add.



In the add controller dialog box, enter the controller name "ProductsController". Click Add.



ProductsController.cs file is generated in the Controllers folder.



Open this file and enter the following code:

namespace ProductsApp.Controllers

{

public class ProductsController : ApiController

{

Product[] products = new Product[]

{

new Product { Id = 1, Name = "Помидор", Category = "Продукты", Price = 1 },

new Product { Id = 2, Name = "Умка", Category = "Игрушки", Price = 3.75M },

new Product { Id = 3, Name = "Молоток", Category = "Инструменты", Price = 16.99M }

};

public IEnumerable<Product> GetAllProducts()

{

return products;

}

public IHttpActionResult GetProduct(int id)

{

var product = products.FirstOrDefault((p) => p.Id == id);

if (product == null)

{

return NotFound();

}

return Ok(product);

}

}

}

For simplicity of the example, the products are stored in an array inside the controller. Of course, in a real application data is requested from a database or from some other source.

The controller defines two methods for returning products:

* The GetAllProducts method returns the entire list of products as IEnumerable<Product>type.
* The GetProductById method searches for one product by the given ID.

That's it. We have created a working web API. Each controller method corresponds to one or more URIs:

|  |  |
| --- | --- |
| Controller method | URI |
| GetAllProducts | /api/products |
| GetProductById | /api/products/{id} |

For the GetProductByID method, id is a field for substitution. For example, to get a product with ID=5, the URL must be /api/products/5.

**There are many different ways to deploy MVC Framework applications, and there are many target servers. You can deploy an application to a Windows Server-based machine running IIS that you run and configure locally; you can deploy to a remote hosting that will configure services for you; and you are increasingly deploying to a cloud infrastructure that will provide all the resources your application needs.**

**We had a hard time deciding how to create a useful deployment example for this Chapter. We ruled out deploying directly to IIS because the server configuration - the process is long and complicated, and the majority of MVC developers who work with the local server, leave the configuration and deployment of the IT group. We also excluded deployment to managed hosting because each provider defines its own deployment process and no one sets standards for hosting.**

So all we have left is deployment to Windows Azure, Microsoft's cloud platform and with good support for MVC applications. We're not saying that Azure is suitable for all deployment options, but we like the way it works; using it in this Chapter, we'll focus on the deployment process rather than being distracted by configuration issues. At the time of writing this Chapter, a free 90-day trial of Azure was available (some MSDN subscriptions also include Azure); therefore, you can repeat the example in this Chapter even if you are not going to use Azure to deploy your applications. First, we'll show you how to prepare your application for deployment, and then we'll move on to the deployment itself.

Attention!

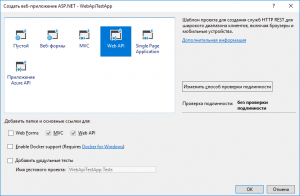
We recommend that you practice with a test application and server before deploying a real application in a production environment. Like any other aspect of the software development lifecycle, the deployment process will benefit from testing. We know eerie stories about development teams that have destroyed production applications due to an overly hasty or poorly tested deployment procedure. You cannot say that the deployment features ASP.NET especially dangerous-no, but any interaction that involves running an application with real user data must be carefully thought out and planned.

Deploying an application has been a tedious and error-prone process, but Microsoft has spared no effort to improve the deployment tools in Visual Studio, so even if you need to deploy your application to a different infrastructure, you can make sure that Visual Studio does a lot of the heavy lifting for you.

# 2. One of the ASP.NET Web API

The first project ASP.NET Web API

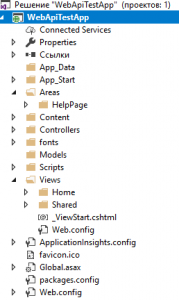
Create a project ASP.NET Web API using the standard Visual Studio dialog box.



Note that both the Web API and the project template are already connected by default ASP.NET MVC. It's not really a coincidence.

The fact is that the web API platform is based on the MVC pattern. Therefore, the implementation of even a template project is based on the appropriate platform.

Below is a screenshot with the structure of the standard web API project template. At first glance, this is the most common ASP.NET MVC project, but the differences are not in the structure (although it includes by default the so-called help page (folder Areas\HelpAge), which can be safely removed), but in the internal content.



One of the main differences is that API requests are handled by controllers that inherit from the ApiController base class. This class allows the controller to handle not only GET and POST, but also other types of HTTP requests. That allows you to develop web API applications in the style of RESTful.

The default controller template offered by Visual Studio already contains a set of methods to support GET, POST, PUT, and DELETE requests, and the following is an example of such a controller.

C#



|  |  |
| --- | --- |
|  | public class ValuesController : ApiController  {      public IEnumerable<string> Get()      {          return new string[] { "value1", "value2" };      }      public string Get(int id)      {          return "value";      }      public void Post([FromBody]string value)      {      }      public void Put(int id, [FromBody]string value)      {      }      public void Delete(int id)      {      }  } |

The names of the controller's public methods correspond to the kind of HTTP requests THEY handle. The from Body parameter attribute means that the value of this parameter should be obtained from the query body.

We will not consider the work of the Web API on the example of a template controller, because even with all the abundance of methods, its work is too simple and is based on the exchange of data of primitive and string types.

Instead, we'll create a complex object to communicate with and a controller to work with.

Object for data exchange

Create the following class in the Models folder.

C#



|  |  |
| --- | --- |
|  | public class DataTransferModel  {      public int id { get; set; }      public double Rate { get; set; }      public String Title { get; set; }      public String[] Content { get; set; }  } |

Its instances will be the objects that we will receive and give in the controller.

The use of relatively simple types and strings in this class, as well as the autorealizable properties, is due not only to the demonstration nature of the task. In almost any API, data exchange is based on the JSON format. Therefore, it is highly desirable to design classes to communicate through the API for ease of JSON serialization and deserialization.

Otherwise, it will be extremely difficult or impossible to process such a class using standard .NET tools. And then you have to write your own conversion algorithm.

An example implementation of the exchange

Let's create a web API controller that will create and return a new DataTransferModel object in case of a GET request, and in case of a POST request, receive a similar object from its body.

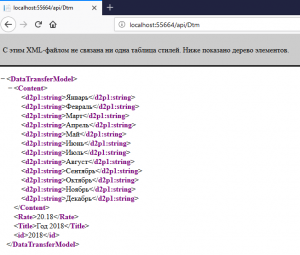
C#



|  |  |
| --- | --- |
|  | public class DtmController : ApiController  {      public DataTransferModel Get()      {          DataTransferModel model = new DataTransferModel          {              id = 2018,              Rate = 20.18,              Title = "Год 2018",              Content = new String [] {"January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", " December" }          };          return model;      }      public void Post([FromBody]DataTransferModel value)      {          DataTransferModel model = value;      }  } |

Now let's try to run the application and execute the GET request.

Note that by default, addresses that belong to the Web API contain an "api/" insert between the address and the controller name (see the screenshot below).



The screenshot shows the result of the query. But, he is represented in XML format, despite the fact that all of the APIs typically use JSON.

This is because the default web API returns data in XML. Therefore, to get an object as JSON, you must serialize it accordingly.

To do this, we use the generalized jsonresult class (the system namespace.Web.Http) and the standard Json function, which actually performs the process of serialization.

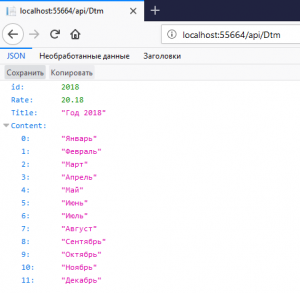
After completion, the method for processing the GET request will look like this:

C#



|  |  |
| --- | --- |
|  | public JsonResult<DataTransferModel> Get()  {      DataTransferModel model = new DataTransferModel      {          id = 2018,          Rate = 20.18,          Title = "Год 2018",  Content = new String [] {"January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "November", " December" }  };      return Json(model);  } |

The result of the request is now passed in JSON format.



As for sending data to the server, everything is much easier there. Simply send a JSON object of the corresponding structure with the HTTP header "Content TYpe: text/json"in the POST request. As a result, the resulting JSON will be automatically deserialized to the required object on the server side.

C#



|  |  |
| --- | --- |
|  | public void Post([FromBody]DataTransferModel value)  {      DataTransferModel model = value;  } |

However, this approach has two very significant drawbacks:

* Given method if it is successful, will return a response code of 204 (no content). Ideally, it is desirable to return something more informative (for example, code 200 (Ok));
* What happens if the resulting JSON does not match the required structure and a critical error occurs as a result?

In the first case, the issue can be easily resolved using the HttpResponseMessage object.

In the second case, the Web API will try to detect data in the incoming JSON, which in its structure corresponds to the desired object and, if successful, will form such an object. Otherwise, the object will also be created. Only the properties of reference types will be null, but primitive types to 0 (bool false). But how correct is this behavior of the program?

Therefore, the received data must be checked.

The following is a modified method, which in case of successful receipt of the DataTransfer Model object returns the code 200 and there is the simplest check of the received data.

C#



|  |  |
| --- | --- |
|  | public void Post([FromBody]DataTransferModel value)  {      if ((value!=null)&&(value.id!=0)&&(value.Title != null)&&(value.Content!=null))      {           DataTransferModel model = value;           return new HttpResponseMessage(HttpStatusCode.OK);      }      else      {          HttpResponseMessage message = new HttpResponseMessage(HttpStatusCode.BadRequest);          message.Content = new StringContent("{\"Message\": \" Invalid request format \"");          return message;      }  } |

The value of the Rate property is not checked because it is a real type, and the values of real types cannot be checked for strict equality or inequality.

Conclusion

We got acquainted with the basic principles of sharing complex data structures with ASP.NET Web API.

The questions related to the "client" part of the application in this article have been specifically omitted, as the client for the server API can be not only a Web application, as it was in the case of AJAX. The API server responds to HTTP requests from any clients that support this Protocol. What makes this approach truly universal.

In particular, screenshots of the results of GET requests were made using an ordinary browser.

Using the Web API allows you to create with ASP.NET web applications of a completely different level, which was almost unavailable earlier in Web Forms and even "clean" ASP.NET MVC.

# 3. Availability management in projects ASP.NET Web API

For security reasons, browsers do not allow AJAX requests to resources that are located on a different domain. This means that using the default settings, you cannot use the Web API service, which is located in a different domain. This is the case when the cross Origin Resource Sharing (CORS) specification comes to the rescue.

While there are other ways to use CORS at the web application level, ASP.NET the web API manages CORS through attributes. Using these attributes, you can configure GPRS at the global level, controller level, and specific action methods. Next, we'll look at how CORS support can be used in applications ASP.NET Web API.

What is CORS?

CORS is Cross Origin Resource Sharing, which can be translated as cross-domain provisioning. CORS is a W3C standard and is related to the mechanism of sending AJAX requests through different domains. A domain consists of a combination of schema, host, and port. For example, consider two addresses:

1. http://www.website1.com
2. http://www.website2.com

In these addresses, http is a schema, www.website1.com I www.website2.com -hosts respectively. And ports are not specified explicitly. In this case, the hosts are different, so the domains are different. And for addresses http://www.website1.com I https://www.website1.com domains are also different, as the schemes are different (http and https).

Suppose that the application ASP.NET Web API available at http://www.website1.com. And there is another application at http://www.website2.com. The second application should use the service offered the first app ASP.NET Web API. You can use to call service functions the XMLHttpRequest object or the jQuery function $.ajax (). But unfortunately this connection will not work, because the browser does not allow to make requests to resources located on other domains.

Although the browser blocks cross-domain requests for security reasons, these requests may still be necessary, and sometimes the developer needs this interaction to work. The CORS specification provides a standard way to establish communication between domains. That is, using CORS you can make AJAX requests to the Web API service located on another domain.

To fully understand this issue, try to run a cross-domain query between two different applications. First, create a project ASP.NET Web API and change the code in the file so:

public class ValuesController : ApiController

{

// GET api/values

public IEnumerable<string> Get()

{

return new string [] {"Red", "Green", " Blue" };

}

\*\*\*\*

That is, the Get method will simply return an array of strings with three values.

Then we add a new TestController and a view for it, Index.cshtml, in which we will place the following code (it is still in the same Web API project):

@{

ViewBag.Title = "Проверка вызова метода /api/values";

}

<h2>@ViewBag.Title</h2>

<script src="Scripts/jquery-2.0.2.min.js"></script>

<script type="text/javascript">

$(document).ready(function () {

var options = {};

options.url = "http://localhost:13667/api/values";

options.type = "GET";

options.contentType = "application/json";

options.success = function (results) {

alert(results.join());

};

options.error = function (evt) {

alert(evt.status + " - " + evt.statusText);

};

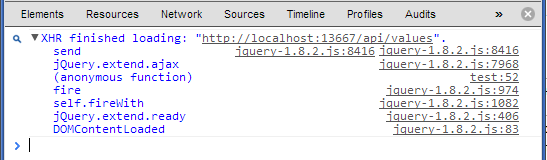
$.ajax(options);

});

</script>

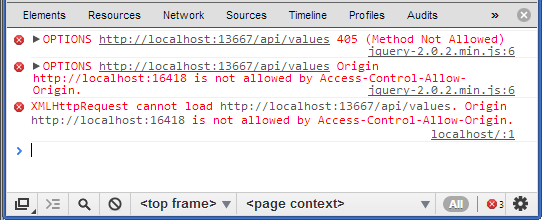
This code performs an AJAX request to the Web API method written earlier. If the request is successful, a window with the array elements received from the service appears. This is the case when the AJAX request and the Web API service are in the same domain (in the same project) and so everything works well. Let's check this by running the application with reference to the /test page (this is the TestController and the Index method).

Now open the Chrome Developer Tools console to track the request. If it is empty, call the right-click context menu, check the item Log Xmlhttprequest and reload the page. We will see the following:



That is, we see the message "XHR finished loading". This is the jQuery method $.ajax () used the XMLHttpRequest object and the object successfully executed the request.

Now create another project ASP.NET MVC 4, choose an empty template (Empty). Add the HomeController controller, method, and Index view. Copy the code from the Index view from the previous project and place it in the new Index view of the new project. In this case, the AJAX request to the Web API will be sent from another application. Such a request will not pass.



As you can see in the picture, the XMLHttpRequest call failed and the error description clearly indicates that the request is not allowed due to domain access restrictions (Access-Control-Allow-Origin).

3. Basic namespaces and ASP.NET Web API configuration

In the process of working with the Web API, we will encounter a similar query processing structure: controllers, request context, model validation, filters, etc. For all these processes, the Web API provides us with similar classes, as in MVC. At the same time, these will be completely different classes, even if they are called the same and fulfill the same role in the system.

For example, you can map some types of MVC and Web API as follows:

|  |  |
| --- | --- |
| Web API | MVC |
| System.Web.Http.Controllers.IHttpController | System.Web.Mvc.IController |
| System.Web.Http.ApiController | System.Web.Mvc.Controller |
| System.Web.Http.Controllers.HttpRequestContext | System.Web.HttpContext |
| System.Net.Http.HttpRequestMessage | System.Web.HttpRequest |
| System.Net.Http.HttpResponseMessage | System.Web.HttpResponse |

At the same time, these types are not complete analogues. And for more convenience, we can also use the classes from the System.Web namespace in Web API 2.

The basic Web API functionality is concentrated in the following namespaces:

* System.Net.Http: contains types representing HTTP requests and responses.
* System.Web.Http: defines most of the types used in the Web API, in particular, the ApiController controller class
* System.Net.Http.Formatting: defines the classes that control the serialization of data when sending a response to the user
* System.Web.Http.Controllers: contains a number of types used by the ApiController class
* System.Web.Http.Dependencies: contains classes that manage dependency injection
* System.Web.Http.Dispatcher: includes types that control the retrieval of request data from the hosting platform and the transfer of the controller from the request processing
* System.Web.Http.Filters: contains filter classes
* System.Web.Http.Metadata: contains classes used to describe model classes using metadata
* System.Web.Http.ModelBinding: classes from this namespace are responsible for associating data from the request with method parameters
* System.Web.Http.Results: contains classes that describe the result of the methods sent to the client
* System.Web.Http.Routing: defines the classes that control the routing in the Web API
* System.Web.Http.Validation: contains classes that control the validation of client request data
* System.Web.Http.ValueProviders: contains a number of auxiliary classes, for example, value providers used to extract data from a query

Web API configuration

The configuration of the WebApi application is defined in the WebApiConfig.cs file, which is located in the project in the App\_Start folder. By default, it contains only the description of the Web API routes, but it may contain other configuration settings:

|  |  |
| --- | --- |
|  | public static class WebApiConfig  {      public static void Register(HttpConfiguration config)      {          // Web API configuration and services            // Web API routes          config.MapHttpAttributeRoutes();            config.Routes.MapHttpRoute(              name: "DefaultApi",              routeTemplate: "api/{controller}/{id}",              defaults: new { id = RouteParameter.Optional }          );      }  } |

The Register method sets the Web API configuration settings.

Whether the application is deployed on IIS, locally or in Windows Azure, the entry point to the application and its configuration is the application class defined in the Global.asax.cs file:

|  |  |
| --- | --- |
|  | public class WebApiApplication : System.Web.HttpApplication  {      protected void Application\_Start()      {          GlobalConfiguration.Configure(WebApiConfig.Register);          //......................................      }  } |

The System.Web.Http.GlobalConfiguration class provides an entry point for configuring the Web API application and defines the following static methods:

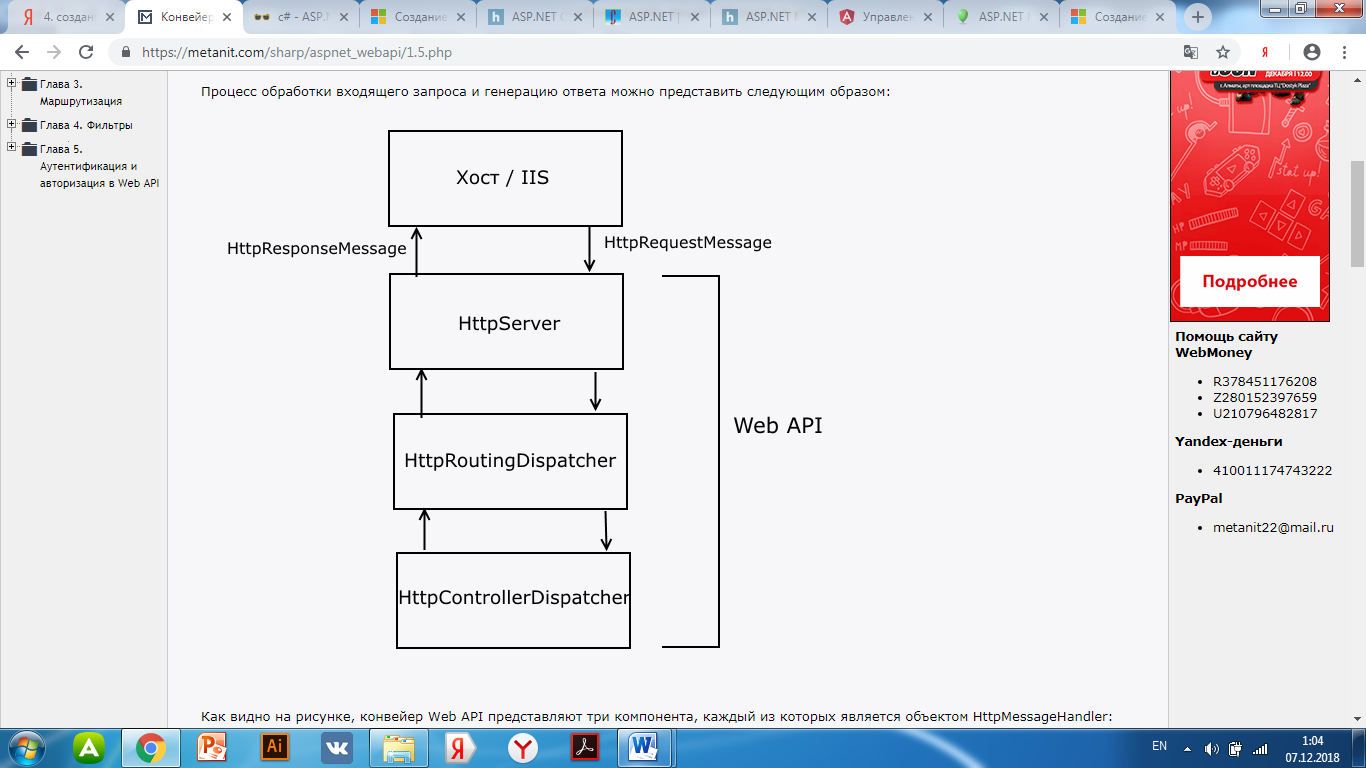
* Configuration: stores an HttpConfiguration object that represents the Web API configuration
* DefaultHandler: provides an HttpMessageHandler object, which is used by default to process requests
* Configure (callback): registers a callback method that is called to configure an application. By default (GlobalConfiguration.Configure (WebApiConfig.Register)) the Register method of the WebApiConfig class is called
* The HttpConfiguration class allows you to customize the configuration using the following properties:
* DependencyResolver: used for dependency injection
* Filters: get or set query filters
* Formatters: stores media type formatters that control the serialization of the response to the client.
* IncludeErrorDetailPolicy: Specifies whether to include detailed information in error messages.
* MessageHandlers: stores message handlers
* ParameterBindingRules: returns a set of rules that establish a binding between the request data and parameters
* Routes: returns a set of application routes.
* Services: returns all used web api services

All these properties form the Web API infrastructure used to process incoming requests.

## 4. ASP.NET Web API Conveyor

The request processing pipeline in the Web API involves several steps. Each stage is controlled by a special component called a message handler or message handler. All these components are inherited from the abstract class HttpMessageHandler from the System.Net.Http namespace.

The processing of an incoming request and the generation of a response can be represented as follows:



As you can see in the figure, the Web API pipeline is represented by three components, each of which is an HttpMessageHandler object:

* HttpServer: receives the HttpRequestMessage request object from the hosting environment
* HttpRoutingDispatcher: set the routing information for the current request
* HttpControllerDispatcher: selects a controller and calls its method to process the request and generate the HttpResponseMessage

### HttpServer

During the configuration phase of the application's life cycle, the GlobalConfiguration configuration class creates an HttpServer object and sets the next MessageHandler object to which HttpServer is to transmit data (by default, this is HttpRoutingDispatcher).

Actually, the entire task of HttpServer is to receive from the server a request in the form of an HttpRequestMessage object and transfer it to the routing system.

Httproutingdispatcher

HttpRoutingDispatcher manages the routing system in the Web API. At the stage of routing, the data is extracted from the request, on the basis of which the data for the route is generated. Route data represents a collection of key-value pairs that are used to match a specific route defined in the application. Typically, this data includes the controller name and method for processing the request, as well as additional request parameters.

In this case, the routing system only establishes data such as a controller or method, but does not select or call it.

HttpControllerDispatcher

The HttpControllerDispatcher receives data from the HttpRoutingDispatcher and, based on this, selects the controller using the IHttpControllerSelector object:

|  |  |
| --- | --- |
|  | public interface IHttpControllerSelector  {      IDictionary<string, HttpControllerDescriptor> GetControllerMapping();      HttpControllerDescriptor SelectController(HttpRequestMessage request);  } |

The GetControllerMapping method returns a collection of all controllers that are in the application. To select controllers in an application, the method activates an IHttpControllerTypeResolver object.

And the SelectController method sets the controller that should process the request. It returns an HttpControllerDescriptor object containing information about the controller.

After this, the HttpControllerDispatcher calls the CreateController () method on the received HttpControllerDescriptor object, which returns the IHttpControllerActivator interface object:

|  |  |
| --- | --- |
|  | public interface IHttpControllerActivator  {      IHttpController Create(HttpRequestMessage request,          HttpControllerDescriptor controllerDescriptor, Type controllerType);  } |

The Create method already creates the controller object itself as an IHttpController interface and, via parameters, passes the HttpRequestMessage request object, the HttpControllerDescriptor object describing the controller and controller type to it.

The Web API provides an embedded implementation of the IHttpControllerActivator interface — the DefaultHttpControllerActivator class from the System.Web.Http.Dispatcher namespace

After creating the controller, the ExecuteAsync method is called:

|  |  |
| --- | --- |
|  | public interface IHttpController  {      Task<HttpResponseMessage> ExecuteAsync(HttpControllerContext controllerContext,              CancellationToken cancellationToken);  } |

# IV Web pages

# Programming ASP.NET Web Pages (Razor) with Visual Studio

Razor is a presentation engine that Microsoft introduced in MVC 3 and that has been slightly redesigned in MVC 4 (although the changes are relatively minor). The presentation engine processes ASP.NET content and searches for instructions, usually to insert dynamic content into the output sent to the browser. Microsoft supports two types of engines: ASPX engine works with <% and % > tags, which were the basis of development ASP.NET for years. And the Razor engine, which works with individual content areas, is indicated by the @symbol.

By and large, if you're familiar with the syntax of<%%>, Razor won't cause too much trouble, though there are a few new rules. In this section, we will give you a brief overview of the Razor syntax so that you can recognize new elements when you meet them. We are not going to study Razor in depth in this Chapter, rather it is an accelerated syntax course. We will explore Razor in more detail later in this book.

Council

Razor is closely related to MVC, but with the advent of ASP.NET 4.5 the Razor view engine also supports ASP.NET Web Pages.

How can I use Visual Studio or Visual Web Developer Express with ASP.NET Web Pages (Razor) web site software?

You will learn how

* What is needed for installation (if available) to work with ASP.NET web pages in your version of Visual Studio.
* How to add support for ASP.NET web pages in Visual Web Developer 2010 Express.
* How to use functions in Visual Studio to work with ASP.NET Razor pages, including IntelliSense and a debugger.
* Software Versions Used in This Manual
* ASP.NET Web Pages (Razor) 3
* Visual Studio 2013
* WebMatrix 3

This tutorial also works with ASP.NET 2, Visual Studio 2012, Visual Studio 2010, and WebMatrix 2 web pages.

You can program ASP.NET web pages with Razor syntax using WebMatrix or other code editors. You can also use Microsoft Visual Studio, which is a full-featured integrated development environment (IDE), provides a powerful set of tools for creating different types of applications (not just websites). You can use Visual Studio 2017 to work with ASP.NET Razor pages.

Below are two particularly useful features provided by Visual Studio for programming using ASP.NET Razor web pages.

* IntelliSense. The IntelliSense feature built into Visual Studio is implemented in more than IntelliSense in WebMatrix.
* Debugger The debugger allows you to eliminate the code, stop the program during its execution, examine the state of variables, and step through the code one by one.

Using Visual Studio with Different Versions of ASP.NET Web Pages

To develop an ASP.NET web application in Visual Studio 2017, you must install ASP.NET and web development workload.

Visual Studio 2012 and Visual Studio 2013 include support for ASP.NET web pages. (When you install Visual Studio, the packages that are required to support ASP.NET web pages will be installed.)

Visual Studio 2010 does not support ASP.NET default web pages. To use ASP.NET web pages using Visual Studio 2010, you must install the ASP.NET MVC package. To get ASP.NET Web Pages 2, you must install ASP.NET MVC 4.

The following table lists the support for ASP.NET web pages in different versions of Visual Studio.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Visual Studio 2010 |  | Visual Studio 2012 | Visual Studio 2013 |
| ASP.NET  Web Pages 2 | Installation ASP.NET MVC 4 |  | (Included) | (Included) |
|  | Веб-  страницы  ASP.NET 3 |  |  | Обновление ASP.NET Web Pages 3 с помощью NuGet | ((Included) |

To work with Visual Studio 2010, see Install Support for ASP.NET Web Pages in Visual Studio 2010.

Start Visual Studio from WebMatrix

If you run a project in WebMatrix and switch to Visual Studio, WebMatrix provides a button to easily open the project in Visual Studio. You must install Visual Studio installed on the computer for this button, enable. The following picture shows the buttons in WebMatrix.

When you click the button, the project is open in Visual Studio. You can switch back and forth WebMatrix and Visual Studio without problems. You will be notified if the files have been changed in a different environment, you must restart to get the latest changes.

Creating an ASP.NET Razor site in Visual Studio

To create an ASP.NET Razor website in Visual Studio:

1. Launch Visual Studio.
2. In the file menu, click the new website.
3. In the new website dialog box, select the language to use (Visual C # or Visual Basic).
4. Choose an ASP.NET (Razor) website template.
5. Click OK.

The project exists and contains some default web pages to help you get started.

Using IntelliSense

Now that you have created a node, you will see a message about how IntelliSense technology works in Visual Studio.

1. Open the website you just created, the Default.cshtml page.
2. After <h3>, enter the tags in the @ServerInfo page. (including point). The IntelliSense function will show the available methods for the ServerInfo helper function in the drop-down list.
3. Select the GetHtml method in the list and press enter. IntelliSense automatically fills the method. (Like using any method in C #, you need to add () characters after the method.) The full code for the GetHtml method looks like the following example:
4. CSHTMLCopy

@ ServerInfo.GetHtml ()

1. Press Ctrl + F5 to launch this page. This looks like a page when displayed in a browser:
2. Close a browser.

### Using the debugger

1. At the top of the Default.cshtml page, after the line that starts with Page.Title, add the following line of code:

C # Copy

var myTime = DateTime.Now.TimeOfDay;

1. In the gray editor box to the left of the code, click next to this new line to add breakpoints. A breakpoint is a marker that tells the debugger to stop the program at this stage so that you can see what is happening.
2. Remove the call to the ServerInfo.GetHtml method and add the @myTime call to the variable in its place. This call displays the current time value, which is returned by a new line of code.
3. Press the F5 key to launch this page in the debugger. The page stops at a given breakpoint. Below is the result of a page in the editor with a breakpoint (in yellow).
4. On the debug toolbar, click the step-by-step button (or F11 key) to launch the next line of code. Each time you press this run button, you go to the next line of code.
5. Check the value of myTime variables by holding the mouse pointer over it or by checking the values ​​displayed in Local and the windows call stack. Visual Studio displays the value of a variable.
6. After completing variable checking and step-by-step code execution, press the F5 key to continue the page execution without stopping in each line. After all the code is completed, the browser displays the page step by step.
7. Send feedback

Using Razor in ASP.NET MVC Projects Using Visual Studio

Razor syntax is also widely used in ASP.NET MVC projects. MVC is an efficient, template-based way to create dynamic websites. If the ASP.NET Web Pages site becomes difficult to maintain, you may need to convert it into an ASP.NET MVC application.

Installing ASP.NET Web Page Support in Visual Studio 2010

This section shows you how to install Visual Web Developer Express 2010 and ASP.NET Web Page Tools for Visual Studio.

1. If you do not already have a web platform installer, download it from the following URL:
2. https://www.microsoft.com/web/downloads/platform.aspx
3. Run the web platform installer.
4. Click the products tab.
5. Search ASP.NET MVC 4 (for ASP.NET Web Pages 2) and click the add button. These products include Visual Studio tools for building ASP.NET Razor websites.
6. Click the install button to complete the installation.

# Redirecting and creating links in Razor Pages

The RedirectToPage () method is used to redirect to the Razor page. The relative path to the page is passed to this method. In this case, the transmitted path is combined with the path to the current page, from which this method is called.

Consider some possible mappings of Razor transmitted paths and pages. For example, the current page is located on the Pages / Products / Create.cshtml path, and the RedirectToPage method is called from it.

|  |  |
| --- | --- |
| RedirectToPage("/Index") | *Pages/Index* |
| RedirectToPage("./Index") | *Pages/Products/Index* |
| RedirectToPage("../Index") | *Pages/Index* |
| RedirectToPage("Index") | *Pages/Products/Index* |
| RedirectToPage("/Edit") | *Pages/Edit* |
| RedirectToPage("Edit") | *Pages/Products/Edit* |

Forward example:

|  |  |
| --- | --- |
|  | public IActionResult OnGet()  {      return RedirectToPage("Index");  } |

Generate address per page

To generate an address on the Razor page, use the Url.Page () method. This method has a number of overloads, allowing you to set various parameters. Some of the versions are:

|  |  |
| --- | --- |
| 1  2  3  4 | Page(string pageName, string pageHandler, object values);  Page(string pageName, object values);  Page(string pageName, string pageHandler);  Page(string pageName); |

The pageName parameter points to the page path. The pageHandler parameter represents the handler defined for the page. And the values parameter represents the query string parameters that are passed to the page.

For example, the About page takes two parameters:

|  |  |
| --- | --- |
|  | public class AboutModel : PageModel  {      public string Message { get; set; }        public void OnGet(string name, int age)      {          Message = "Your application description page.";      }  } |

Creating URLs on this page followed by redirection in C # code:

|  |  |
| --- | --- |
|  | public IActionResult OnGet()  {      string url = Url.Page("About", new {name="Tom", age=34});      return Redirect(url);  } |

Creating an address on a page in a view:

|  |  |
| --- | --- |
|  | <a href='@Url.Page("/About", new {name="Bob", age=32 })'>About</a> |

Creating links

To create a link in the AnchorTagHelper tag-helper, use the asp-page parameter.

For example, a link to the same About page:

|  |  |
| --- | --- |
|  | <a asp-page="About" asp-route-name="Bob" asp-route-age="32">About</a> |

# 3. HTML helpers

To display content in a view, we can use standard html elements that allow you to create blocks, lists, tables, etc. But besides the actual html elements for creating markup, we can use special methods - html-helpers. In general, helper can be translated from English as an "auxiliary method". And actually html-helpers are auxiliary methods, the purpose of which is to generate html-markup.

To create the simplest html helper, take an ASP.NET Core 2.0 project like Web Application (Model-View-Controller) and add the App\_Code folder to it. Then add a new class ListHelper to this folder:

|  |  |
| --- | --- |
|  | using Microsoft.AspNetCore.Html;  using Microsoft.AspNetCore.Mvc.Rendering;    namespace HtmlHelpersApp.App\_Code  {      public static class ListHelper      {          public static HtmlString CreateList(this IHtmlHelper html, string[] items)          {              string result = "<ul>";              foreach (string item in items)              {                  result += $"<li>{item}</li>";              }              result += "</ul>";              return new HtmlString(result);          }      }  } |

In the new helper class, one static CreateList method is defined, which takes as its first parameter the object for which the method is being created. Since this method extends the functionality of html-helpers that the Microsoft.AspNetCore.Mvc.Rendering.IHtmlHelper interface represents, it is an object of this type that is passed in this case as the first parameter. The second parameter of the CreateList method is an array of string-values, which will then be displayed in the list.

In the method itself, we simply run through the array of strings and form the html markup from them as a string. The result of the method is an HtmlString object, which in the constructor receives the html markup as a string.

This very simple method can already simplify the work with markup. Consider using it. Suppose we need to display in the view an array of strings in the list:

|  |  |
| --- | --- |
|  | @{      ViewData["Title"] = "Home Page";  }    @{      string [] cities = new string [] {"London", "Paris", " Berlin" };  string [] countries = new string [] {"United Kingdom", "France", " Germany" };  }  @using HtmlHelpersApp.App\_Code    <h3> Cities </h3>  @Html.CreateList(cities)  <br />  <h3> Countries </h3>  <!-- or you could call -- >  @ListHelper.CreateList(Html, countries) |

Since the html helper represents an extension method for an IHtmlHelper object, to use it, we just need to write Html.CreateList and pass the required parameters to the method. Or we can call it as a method of the class in which it is defined: ListHelper.CreateList

And now, if we want to create a list <ul>, we only need to write one line with the helper call, passing it an array:

In the absence of such a helper, we would essentially have to duplicate the same html-code to create the list. However, this helper is still quite simple, and if we have to create a hundred times more complicated, but the same type of html markup, then the helpers will be even more useful.

Send feedback

### TagBuilder

To create html tags in the helper we can use the class Microsoft.AspNetCore.Mvc.Rendering.TagBuilder. So, rewrite the helper code as follows:

|  |  |
| --- | --- |
|  | using Microsoft.AspNetCore.Html;  using Microsoft.AspNetCore.Mvc.Rendering;  using System.Text.Encodings.Web;    namespace HtmlHelpersApp.App\_Code  {      public static class ListHelper      {          public static HtmlString CreateList(this IHtmlHelper html, string[] items)          {              TagBuilder ul = new TagBuilder("ul");              foreach (string item in items)              {                  TagBuilder li = new TagBuilder("li");                  li.InnerHtml.Append(item);                  ul.InnerHtml.AppendHtml(li);              }              ul.Attributes.Add("class", "itemsList");              var writer = new System.IO.StringWriter();              ul.WriteTo(writer, HtmlEncoder.Default);              return new HtmlString(writer.ToString());          }      }  } |

The element for which the tag is created is passed to the TagBuilder constructor. TagBuilder has a number of properties and methods that can be used:

* The InnerHtml property allows you to set or get the contents of the tag as a string. To manipulate this property, you can call one of the methods:
  + Append (string text): append test string inside element
  + AppendHtml (IHtmlContent html): appending an html code to an element as an IHtmlContent object — this could be another TagBuilder object
  + Clear (): clears the item
  + SetContent (string text): set element text
  + SetHtmlContent (IHtmlContent html): setting the inner html code as an IHtmlContent object
* The Attributes property allows you to control the attributes of an element
* The MergeAttribute () method allows you to add one attribute to an element.
* The AddCssClass () method allows you to add the css class to an element
* The WriteTo () method allows you to create a string from an element and its internal contents using TextWriter and HtmlEncoder objects.

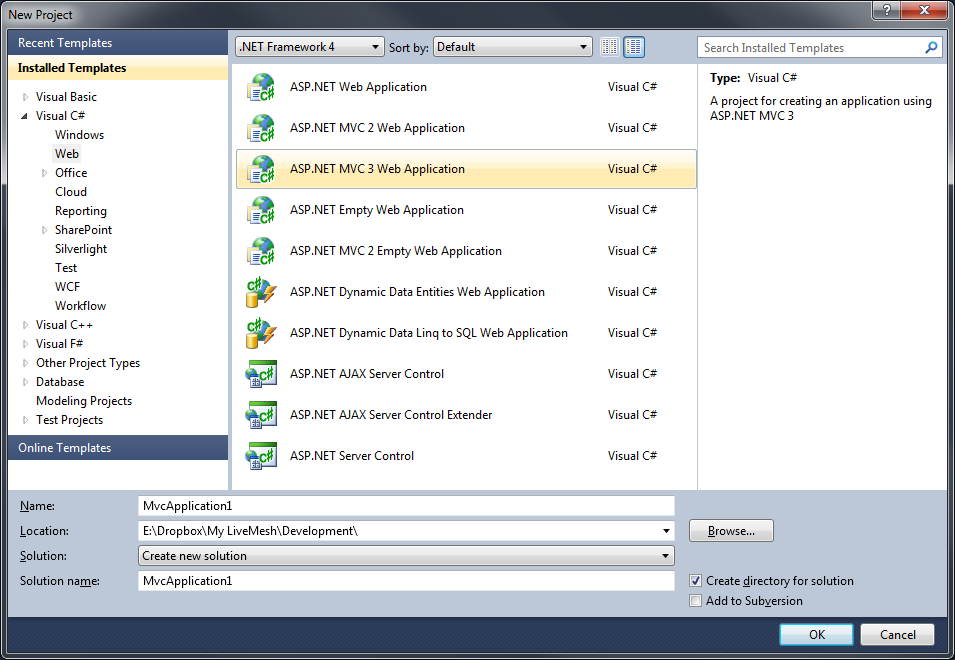
As a result, we get the same list as before

# 

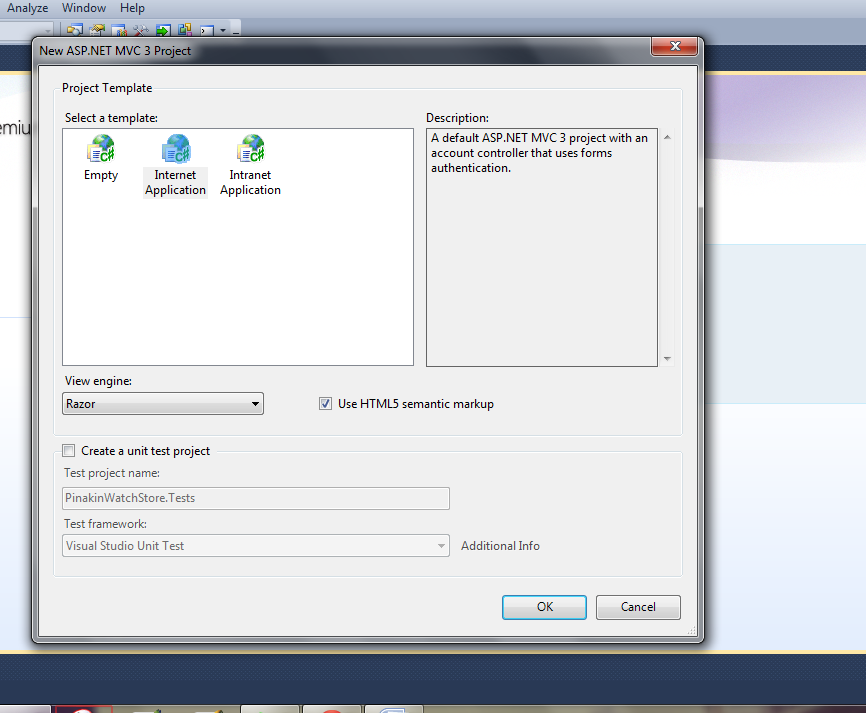
**Practical task №1 . Create ASP.NET MVC project**

Start Visual Studio, choose File->New->Project from the menu..."

In the open window, select "Web", from the list Of installed Templates. Then choose "ASP.NET MVC 3 Web Application" . Write the name of the project and the path to the directory where it will be located



Click OK. In the next window, leave the selected "Internet Application" and click OK



We have to create the basic project asp.net mvc. After pressing F5 (Start Debugging), we will see our website (start virtual IIS and an icon will appear on the start panel that displays its operation). Every site running with Visaul Studio is running on some port (for example, localhost: 29663) so don't worry if the numbers will be different from mine.

What we've built the Studio, and how it works asp.net the mvc app.

First you need to understand the simple logic of all sites and how they differ from the desktop/windows applications.

When we open a website (for example www.aspnet.com.ua/Category/category-3.aspx), then a request is sent to the server (this is equivalent to the event of pressing a button in the application desktop), and tell the server to give us some information (in our example to give the info about the "category 3" of the site aspnet.com.ua).

The server, in turn, either knows this command (this url) and gives us the necessary information, or returns an error (for example, page 404). After the server has executed the command, it forgets about us request.

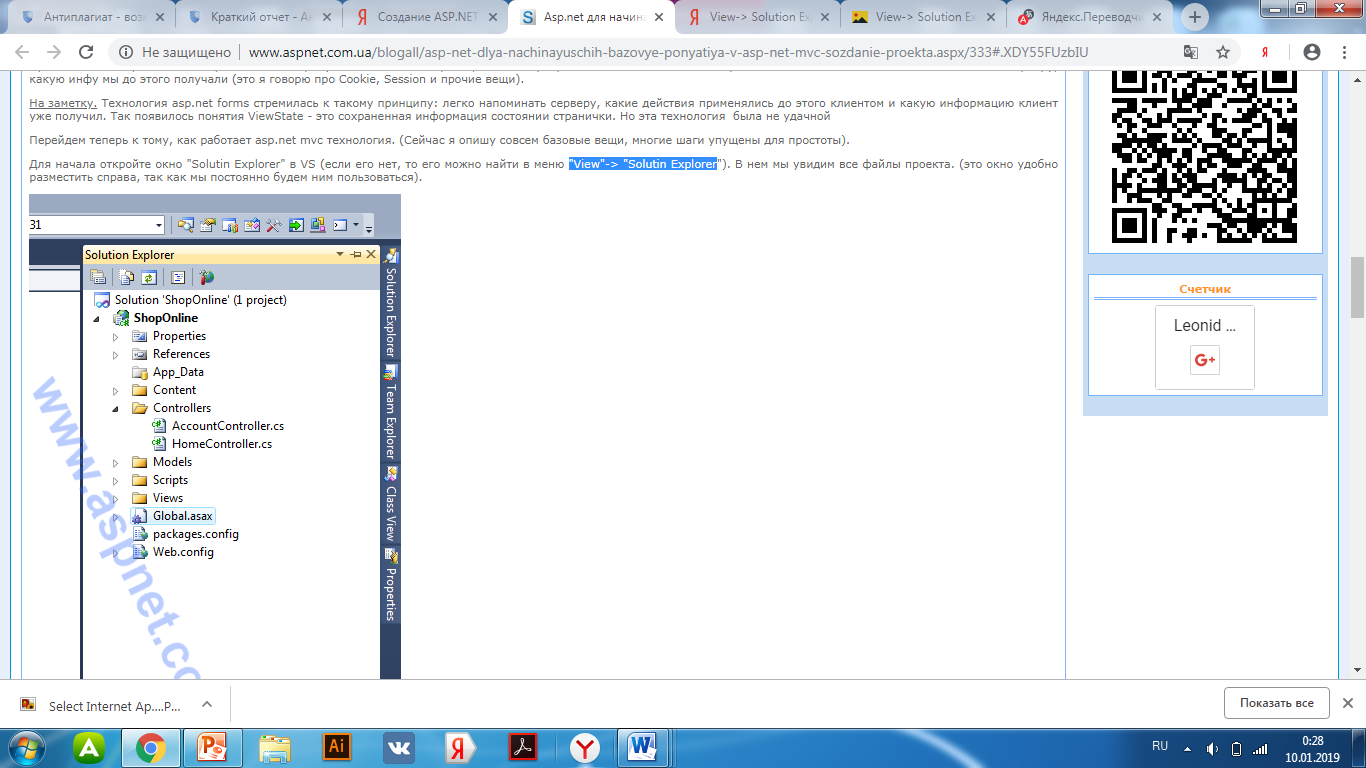
When we click a button on the site, it is generally equivalent to opening a new page of the site by a new user. The server is made does not remember what he told us before that we gave and who we are. Ie, the server works according to the principle: 'tis forgot.

With the further development of the Internet began to come up with all sorts of tricks that the server would somehow remember with whom it works and that it would be easier for us to remind the server what info we had received (I'm talking about Cookies, Session and other things).

On a note. Technology asp.net forms was committed to this principle: it is easy to remind the server what actions were used before by the client and what information the client has already received. So emerged the concept of ViewState is stored information about the state of the page. But this technology was not successful

We now turn to how to operate asp.net mvc technology. (Now I will describe very basic things, many steps are missed for simplicity).

To start, open the window "Solutin Explorer" in VS (if it is not, it can be found in the menu "View"-> "Solutin Explorer"). In it we will see all the files of the project. (this window is convenient to place on the right, as we will always use it).



We are now interested in the Controllers folder-in it we create files that will process our actions (process our URLs). We are also interested in the Global file.asax, in it we will specify which file from the Controllers folder which url will process. Open the Global file.asax and find this code:

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute ("{resource}.axd / {\*pathInfo}");

routes.MapRoute(

"Default", / / Route name

"{controller} / {action}/{id}", / / URL WITH parameters

new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

}

This is the url-to-Controllers binding rule. Delete the line with " routes.MapRoute..." on." ....UrlParameter.Optional }); " Instead, we will write our three rules:

routes.MapRoute(

"Root", / / rule name

"", / / what URL

new { controller = "Home", action = "Index"} / / which controller file

);

routes.MapRoute(

"Home-About", / / rule name

"About.aspx", / / what URL

new { controller = "Home", action = "About"} / / which controller file

);

routes.MapRoute(

"Account-LogOn", / / rule name

"Account / LogOn.aspx", / / what URL

new { controller = "Account", action = "LogOn"} / / which controller file

);

Each rule has its own name, which should not be repeated ("Root "Home-About" "Account-LogOn"). Also, each rule must specify a URL AND a controller that will handle this action.

Now in my site there are three pages/three rules:

- Account / LogOn.aspx-it will be handled by The accountcontroller controller and the method of this LogOn controller

- About.aspx-this page will be handled by the HomeController controller and The about controller method

- root page-it will handle the HomeController controller and the method of this Index controller

Now open the HomeController file (use "Solutin Explorer" to open the files). in This file you will see the HomeController class, which is inherited from the Controller class and two methods of this class Index and About. These methods will process our URLs.

The basic logic of the application is written in these methods (for example, reading or entering information into the database). Let's count the server time in the Index method, but in the About method we will calculate how much 345 times 23.

Look at the code:

public class HomeController : Controller

{

public ActionResult Index()

{

var date = DateTime.Now;

return View();

}

public ActionResult About()

{

var result = 345 \* 23;

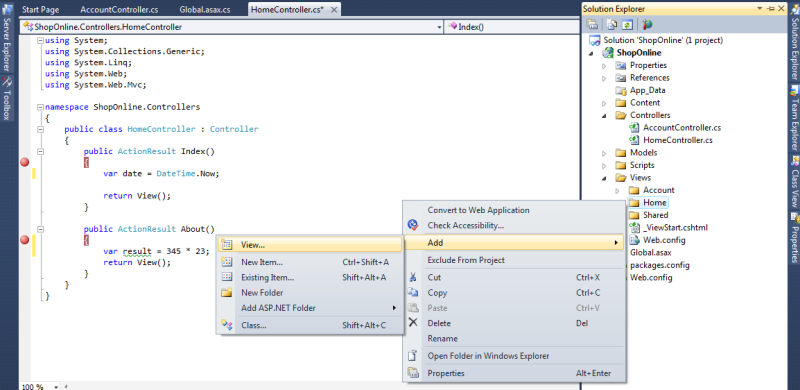
return View();

}

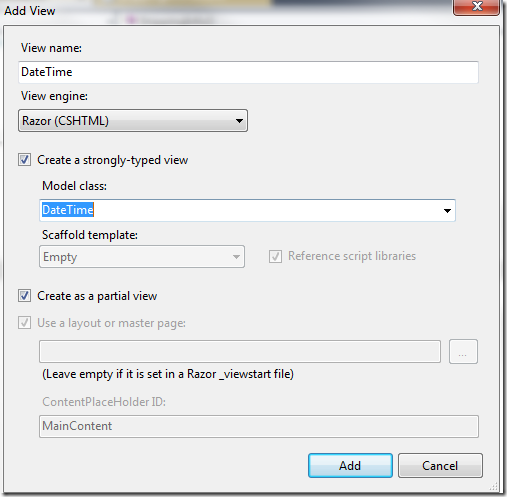
}

Now we need to display the result of our actions to the user. In web applications, this is done using html pages. These pages are usually located in the View folder (use "Solutin Explorer"). In the View folder for each controller creates its own folder with the name of the controller (so it is easier to navigate). Let's create some pages.

Right-click on the folder " Home "in the context menu, select"Add"-> " View..."



Before we open a window in which we give the name our page "ViewDateTime". Just need to remove the tick "Use layout or master page", we'll talk about it later. So, the window should look like:



Click Add. Studio will create the file ViewDateTime.cshtml and create a basic html structure in it. In the same way, add another ViewResult file

Let us now return to our methods of controller. Change the string " return View ();" in the Index method to "return View("~/Views/Home/ViewDateTime.cshtml", date);" and in the About method on "return View("~/Views/Home/ViewResult.cshtml", result);"

This means that we will display the result of our actions on the views (View) of ViewDateTime and ViewResult, respectively, as we passed date and result to these views.

To simplify the life of programmers and not always write a long way ("~/Views/Home/.....cshtml) files display made use of such rules:

\* return View (); / / means that the display file is located in the folder with the controller name, and the file has the same name as the method. For example, for the index method of the HomeController, the view will be " ~/Views/Home / Index.cshtml"

\* return View ("MyView"); / / means that the display file is located in a folder named controller, and the file is named MyView. For example, for the index method of the HomeController, the view will be ~/Views/Home / MyView.cshtml

Based on the above, let's change the lines of code once again: the Index method will return the return View ("ViewDateTime", date); and the About method will return the return View ("ViewResult", result);

Now note that in addition to specifying the View file, we also pass the data for display (date and result). Now we need to adjust their correct display.

Open the ViewDateTime file.cshtml and first add the code "@model DateTime". It means that the ViewDateTime file will display a variable of DateTime type. If we did not specify what type of variable will display this view, the code would be working, but then on this page we did not have a clue. Add the @Model code between the <body> tags.ToShortDateString ().

Using the code @Model-we refer to the object that was passed to the View. The complete code of the ViewDateTime file.cshtml

@model DateTime

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<title>ViewDateTime< / title>

< / head>

<body>

<div>

@Model.ToShortDateString()

< / div>

< / body>

< / html>

Now open the ViewResult file.cshtml. At the beginning of the file, add " @model int "

Between <body> tags write the code "345 \* 23 = @Model" the Full text looks like this:

@model int

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<title>ViewResult< / title>

< / head>

<body>

<div>

345 \* 23 = @Model

< / div>

< / body>

< / html>

Now start your application (F5 key). You will see the current date

If you go to the url http://localhost:29663/About ahhh!aspx you will see the result of operation 345 \* 23

So, what you should master: asp.net mvc application has a folder Controllers - in which we write all the actions/URLs of the site; there is also a folder Views-in which we write a view of our actions (html page); there is also a file Global.asax-which links the url of the site from the methods of the controller.

**Practical task № 2. Create and process a view. site architecture**

Site architecture all companies have their own and all consider it ideal, and get used to someone else's site construction is very difficult. I'll tell you how I arrange the project files. My architectural solution is not the best, so all suggestions and criticism for improvement are accepted.

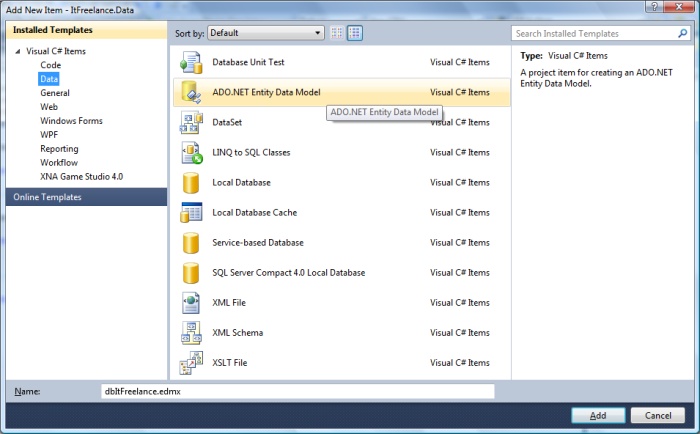
Create Asp.Net MVC 3 web application with unit test project. After that we will have one solution and two projects in it (website and unit tests )

The next step is to create a project to communicate with the database. Framework I'm using ORM faces. It is better to make a separate project in the ORM, so that it can be easily changed or use other methods of communication with the database or even use another storage. The project is generally referred to as I {my project}.Its Data type will be " class library".

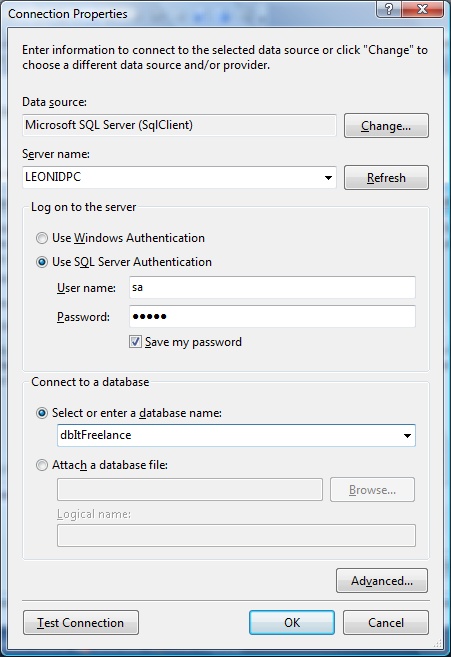


In the creation of the separate project to delete the file Class1.cs, he has to go. And add ORM.

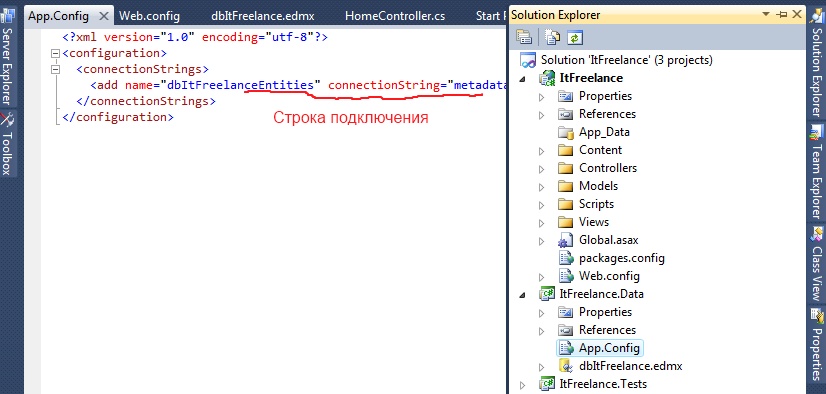
To do this, click on the project name and select "Add"->"new item in the context menu...". Select "model entities ADO.NET data" Write on name of our ORM



After that, we will open the wizard to create a connection to the database. In the first step, select "Create from database". Then click the button " new connection..."and specify the connection parameters and the desired database.



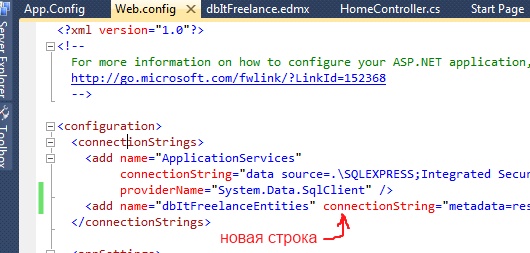
Click "OK". Select the option "Yes, include sensitive data in the connection string" and click"next>". In the next step, you must select the tables that will be involved in the project (if they are not present, you will have to create, for example, a table. Persons and blogs). One of the features of this ORM is that all tables had a primary key. Click Finish. After that, two files will be added to our project (\*.edmx file and applications.Configuration.) In the application file.Config is the database connection string.



You need to copy the line:

<add name= "dbItFreelanceEntities"connectionString property=" metadata=RES://\*/.................. system provider name=".Data.EntityClient provider>/ "

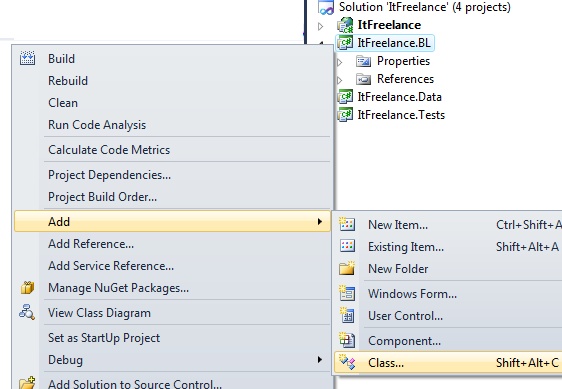
and paste it into a web file.site config.

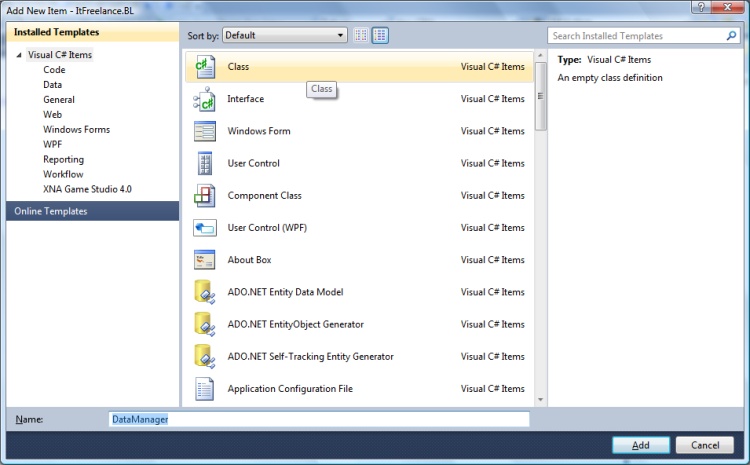


We will now have two connection strings there. The second we will use to configure user authorization (membership).

Now create a project that will contain all the business logic of the system. Let's call the project {My project}.BL its type will also be "class library". Class 1.cs should be removed.

We give the DataManager class to the project. It will manage (or rather store) business objects of the system (repository)





In this class, we write a constructor in which we will initialize the ORM object. With this object we will access the database:

Open DataManager class

{

private dbItFreelanceEntities the DataContext class;

public DataManager()

{

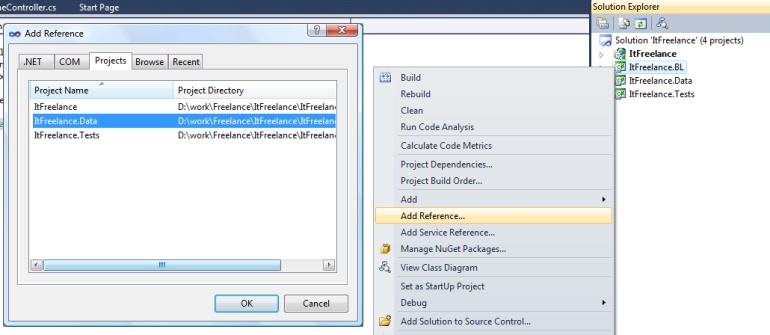
class DataContext = new dbItFreelanceEntities();

}

}

Now let's make sure that the projects know about each other. To do this, we give the project "{My project}.Data" in " {MyProject}.BARREL."A Reference" {My Project}.BL " to the project of our website. Therefore, data->connection turns out BL->Website. Please note that the Site does not know anything about the data, and will communicate with her through the BL.

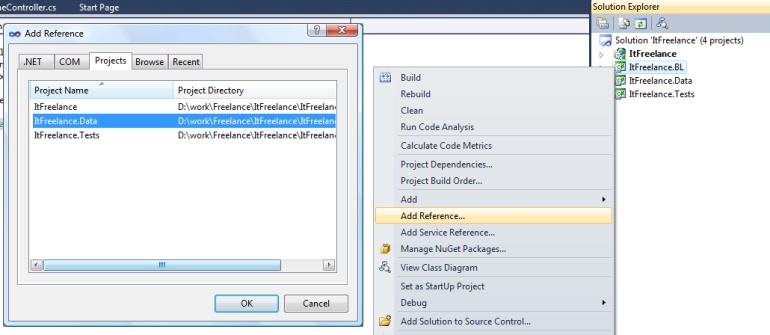
To add links, click on the project name and select "Add link" in the context menu...". In the window that appears on the projects tab, select the desired project (for {My project}.Doda BL {my project}.Data, and for site {My project}.BARREL)



Solution solution (F5) see that everything works. (Likely in project {My project}.BL you will need to add a link " system.Data.The essence of "bookmarks" .NET")

Create now the main kotnroler, it will do common actions for all controllers (for example, to find out information about the user authorized, set the desired location, or keep some statistics). This controller will be inherited by all other controllers.

Click the right mouse button on the controllers folder and add the controller with the name Controller Main.



Index the index of the action and override the initialization method. This method will vyzyvaetsya each time upon initialization of the controller, similar to the constructor. Also in this controller there will be a DataManager property, with the help of it we will get access to the database. Here's the code

public class ControllerMain: controller

{

private DataManager \_dm;

public DataManager DM { get { return \_dm; } }

secure override void Initialize (System.Network.Routing.RequestContext RequestContext-)

{

base.Initialize(requestContext);

\_dm = new DataManager();

}

}

Let's start working with the database.

We will use the database to retrieve the first and last name of the Vygdm user.Men.GetInfoUser (user ID)) to implement this, we will add the Persons\_Repository {my project} class to the project.BARREL. For each logical object of the system will create a similar class, the main thing that this class would have methods that apply only to the desired object.

In this class, we implement a constructor that will take dbItFreelanceEntities class DataContext and DataManager DM, it is necessary for the class to have access to the database:

public class Persons\_Repository

{

personal data.dbItFreelanceEntities the DataContext class;

private DataManager dm;

public Persons\_Repository (Data.dbItFreelanceEntities DataContext, and DataManager dataManager)

{

this.the DataContext = DataContext class in;

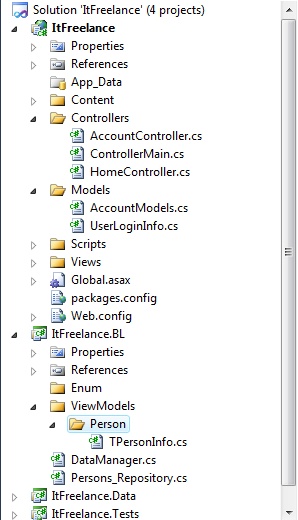
this.dm = dataManager;

}

}

Now we will add methods to work with the human object to this class. The main criterion to be supported is not to take the classes that ORM created outside the {My project} project.BARREL. This means that the project in the MVC-site should not use the classes created by ORM, if you want to pass some information for it to create a separate class.

For example, we implement the GetInfoUser method. In the project, first create a folder {My project}.View models BL. It will store our classes for the exchange of information between projects and supervisors and views. In this folder create a new folder of a person's need for a convenient grouping of classes. And already in this folder we give the class TPersonInfo.



Now let's write a method that will return this class, as well as add the necessary properties to the class:

GetInfoUser public TPersonInfo(GUID? user ID)

{

var data = datacontext.Men.Where (pr => pr.PersonID == Username &&!PR.IsDelete)

.Select (l => create TPersonInfo()

{

PersonID = l.PersonID,

RoleID = l.Users.Roles.Select (p => p. RoleId).Method firstordefault(),

Name = L. Name,

LastName = l.Surname,

IsDelete = l.IsDelete,

}).Method firstordefault();

return data;

}

The next step is to open the DataManager class and add the human property there

Open DataManager class

{

private dbItFreelanceEntities the DataContext class;

public DataManager()

{

class DataContext = new dbItFreelanceEntities();

}

private Persons\_Repository \_Persons;

public Persons\_Repository persons

{

get

{

if (\_Persons == null)

\_Persons = new Persons\_Repository(DataContext, this);

return of \_Persons;

}

}

}

So, that we have. First, we have a DataManager class that has references to classes for working with objects (an example of such a class is Persons\_Repository). These classes have methods with which we can add/delete/modify/retrieve data from the database (example of the GetInfoUser method in which we get information about the user). Secondly, we have a main controller (ControllerMain) from which all system controllers will inherit, and in itself it will have a reference to an object of type DataManager.

Now open the controller HomeController. Let's make it inherit from the ControllerMain class. In the index action, we will receive information about the user with ID {90A793FD-5391-41AE-8A5F-8A5C92091136} (such a user is not present, anything terrible ):

public class HomeController: ControllerMain

{

public ActionResult index()

{

ViewBag.Message = " Welcome ASP.NET MVC!";

users var = dm.Men.GetInfoUser(new GUID ("{90A793FD-5391-41AE-8A5F-8A5C92091136}"));

return view();

}

}

Now we get information about the registered user. As it is necessary to know on all pages, we will receive this information in the method of initialization of our controller.

Create UserLoginInfo class (right-click on the folder and select "Add" - >"class" in the context menu..."write the name UserLoginInfo) (in order to understand where which files are located, see picture above)

This class will have the following properties (this is General information about the user):

state system.Network.Security.MembershipUser user { get; private set; }

public Guid? UserID { get; private set; }

public string UserName { get; private set; }

public Guid RoleID { get; Private set; }

public bool IsAdmin { get { return false; } }

public bool IsAutoriaze { get { return user ID != GUID.Empty; } }

public string FirstName { get; private set; }

public string LastName { get; private set; }

Now we implement a static method in this class with the help of which we will get this class

public static Userlogininfo GetAccountInfo (DataManager dm)

{

if (dm = = null)

dm = new DataManager();

return a new UserLoginInfo (dm);

}

This method uses a constructor, let's implement.

public UserLoginInfo(DataManager DM)

{

this.User = System.Network.Security.Membership.GetUser functions();

if (this.User = = null)

return;

this.Login = User.user name;

this.User ID = (ID)Of The User.Realization of action;

var InfoUset = dm.Men.GetInfoUser(Userid);

if (InfoUset == null)

return;

**Practical task № 3 How to build a Single page Application on ASP.NET?**

Single page Application - abbreviated SPA, translated into Russian means “one page Application”. In other words, SPA is a web application hosted on one web page, which loads all the necessary code together with the page itself to ensure operation. An application of this type appeared relatively recently, with the beginning of the era of HTML5 and SPA is a typical representative of applications on HTML5.

As we know, HTML5 is something other than HTML + CSS3 + JavaScript + [several new tags]. Thus, SPA is an application written in JavaScript. And, therefore, slightly paraphrasing the previous definition we obtain:

"SPA is a web application hosted on a single page that loads all javascript files (modules, widgets, controls, etc.), as well as CSS files along with the page itself , to ensure operation.”

If the application is quite complex and contains a rich functionality, such as an electronic document management system, the number of files with scripts can reach several hundred or even thousands. And "... loading all scripts... " in no way means that when you load the site will be loaded at once all the hundreds and thousands of files with scripts. To solve the problem of loading a large number of scripts in the SPA API called AMD. AMD implements the ability to load scripts on demand. That is, if the “main page” of a single-page portal required 3 scripts, they will be loaded before the start of the program. And if the user clicks on another page of the single-page portal, for example,” About", the AMD principle will load the module (script + markup) only before going to this page.

It turns out a little crumpled: "one page.. the other page, the third page ... one-page portal." Let's dot the “E”s. The page of the site, which contains all the links to all CSS, and links to the scripts necessary for the SPA, we will call “Web-page”. A file with such a page is usually called " index.html” (in ASP.NET MVC can be an index.cshtml or index.vbhtml or even index.aspx) and the pages that the user switches within a single-page portal are called "modules".

Let's look at the pros and minutes of this approach. Why all this is necessary and why SPA is so popular?

SPA: Advantages

The first advantage is the fact that SPA applications work perfectly on both stationary and mobile devices. “Big " computers, tablets, smartphones, and, in the end, simple phones (some) can easily work with sites built on the principle of SPA. So, the first "plus" - work on a large number of devices, which means that by creating one application, you get a much larger audience of users than using the standard approach.

Next, the second "plus" - a rich user interface, the so-called User Experience. Since there is only one web page, building a rich, rich user interface is much easier. It is easier to store session information, manage view States, and manage animations (in some cases).

The third” plus " – SPA significantly (at times) reduces the so-called “walking in a circle”, that is, downloading the same content again and again. If your portal (site) uses a template, then along with the main content of any page, the visitor of the site necessarily loads the layout of the template. Yes, data caching at this stage of WWW development has achieved the highest results, but if there is nothing to cache, then both time and resources are not spent on it.

SPA: Cons

If you program in C#, the only downside to SPA is the need to learn JavaScript. In any case, I could not find out other global problems.

SPA components

The principles of any framework (we'll talk about them later) that implements the SPA paradigm should adhere to the following concepts and definitions:

\* SPA supports client navigation. All” walking “of the user through the modules-pages are unambiguously recorded in the navigation history, and the navigation is” deep", that is, if the user copies and opens a link to the internal module-page in another browser or window, he will get to the corresponding page.

\* SPA is located on one web page, so all the scripts and styles necessary for the operation of the site (portal) must be defined in one place of the project-on a single web page.

\* SPA permanently stores the state (important variables) of the client (client script) in the browser cache or Web Storage.

\* SPA loads all scripts required to start the application when initializing a web page.

\* SPA gradually loads modules on demand.

Templates SPA (SPA templates)

As you may have guessed, SPA is an abstract concept. This is the principle of the application architecture. Let's talk about where to start when developing a website on the principles of SPA.

There are a large number of basic libraries (framework – from the English word framework – “basis, structure, framework”) that implement the principle of Single page Application. What these frameworks give:

\* provide basic principles for SPA development, minimizing the effort to solve universal problems (see section " SPA Components);

\* frameworks created by the community of developers, and therefore use the experience of creating websites of many programmers;

\* frameworks are the starting point for building a structure based on a Single page Application.

Since I have been working on the NET platform FOR many years, I will consider single page Application templates based on ASP.NET ahhh! Let's look at the following comparison table.

Comparison of SPA template features

The table shows the most common templates for how to build a Single page Application application. Note that the blue background highlights the basic building blocks for a complete framework, such as DurandalJS and HotTowel, which are highlighted in green.

So, following the data provided in the table you can create a Single page Application application using “naked” ASP.NET and KnockoutJS. However, the implementation of the work with data (DAL) you have to write yourself, however, as the management of navigation and navigation history including.

On the other hand, you have the right to choose Ember or BreezeJS, or even Google's AngularJS as the basis of your website or even as the basis of your own framework, the fact remains that the missing principles that make up the concept of SPA you will have to implement yourself.

An alternative to the previous solution can be the choice of a ready-made full-fledged framework (marked in the table with a green background). Each option has its own "pros”and "cons".

**Conclusion**

There are many examples of applications built on the principles of Single page Application. One of the most powerful and well – known is GMail-mail service of Google.

Basic rules of single page application

1. All web application entities are based on models and objects (working with page DOM elements is encapsulated inside objects).

2. HTML templates are stored in scripts (as far as possible).

3. Any changes to the page dynamically change the url.

4. Direct download of any url should display the corresponding data page.

5. History back (back button in the browser) should be processed correctly and return the page to the previous state.

6. Caching of data models on the client side.

From my point of view, the above points are the main ones. Of course, to optimize the work, or to avoid the complexity of the system something will have to sacrifice.

Sequence of work with web application

1. The index page is loaded (the template is fully displayed and filled with data).

All necessary objects are Initialized and all the event listeners are installed.

2. The user clicks on the link/button / any interactive element.

3. The application intercepts the click event.

4. If clicking on the object involves changing the state of the web application - >

5. Create a new URI for the new page state.

6. Change the current uri using javascript (change uri without reload page).

7. The URI change event is intercepted.

8. Parse our new address, get all the keys-values.

9. Check what has changed in the keys.

10. We send a request to the server for new data.

11. We accept the answer and call the callback function of successful data loading.

12. Redraw the necessary parts of the page.

In this sequence, the question appears on the account of points 5-10 (javascript url change and data query), why not make a data query immediately when changing the address? The answer is simple: we create one entry point to work with changing the uri, and one entry point to handle the new address and request data. If a dozen methods do this every time, it will be bad code, as there will be a lot of copy-paste. The above mentioned way will be two entry points and, as a consequence, extension points of these sections of the web application.

Implementing a single-page application

In the final, using the sequence / click on the active element - > change uri - > process new uri - > query data - > draw new page elements / you can create full-featured single-page applications. In my work, I used the jQuery framework, and distributed almost everything into classes, each of which managed its own area.

The main JavaScript initialization file is created, which starts the application. It also creates a main class (for example, singleApplication), which controls the state of the application, initializes the necessary events, works with the history object, processes and changes the url, and other functions. The URL formed with the support of SEO (/category/tech/page/2) on the concept of /key/value/. In my application, I also used a javascript observer, which allowed me to reduce the number of errors, minimize the connectivity of classes and make it easier to work with the functions-callback, on which I built a single page application.

**Practical task№ 4.ASP.NET. Authentication**

Let us now turn to the description of the authentication process directly to the environment ASP.NET where your choice of 3 kinds of authentication:

\* Windows authentication

\* Form

\* Passport

Windows authentication:

As the name implies, this method is based on the use of Windows accounts. This method is appropriate if you are creating an application for a local network, and all valid accounts and groups are stored on a predefined domain. You should be very careful when assigning access rights to users, because you also set permissions for Windows at the same time. To configure ASP.NET to work in Windows authentication mode, you must change the web project configuration file.config or, if necessary, the entire server configuration file located at WINDOWS\_FOLDERMicrosoft.NET

Framework.NET versionCONFIGMachine.config. In our example, we will work exclusively with the project file – Web.config where you need to find the authentication section and set its mode attribute to Windows:

<AUTHENTICATION mode="Windows"></AUTHENTICATION>

You can now proceed directly to programming and implementing Windows-based authentication. To help you, the WindowsIdentity class is specifically designed to work with Windows authentication. In General, for Windows-based authentication, there are two main classes provided by the .NET Framework:

\* GenericIdentity-only implements the IIdentity interface and does not apply to any particular authentication type

\* WindowsIdentity is also an implementation of the IIdentity interface, but it also includes methods specific only to Windows-based authentication

The user and group names are stored in the WindowsIdentity object in the following format: DOMAINUserName and DOMAINGroup, respectively. The only exceptions are built-in groups, for example, Administrators group, you can use the connection string through WindowsIdentity: BUILTINAdministrators to access it. Alternatively, you can specify a built-in group from the System enumeration.Security.Principal.WindowsBuiltInRole.

The WindowsIdentity object allows you to get a user name; determine the type of authentication; determine whether authentication was performed anonymously; also, you can find out whether the user was authenticated or not, whether he or she is a guest or a system user.

Because in applications ASP.NET to access the WindowsIdentity object, you will need to build the following chain:

HttpContext.Current.User.Identity, you can also determine which role the current user belongs to. This can be achieved because the User property in this chain implements the IPrincipal interface, which allows you to determine whether a user belongs to a specific role by calling the IsInRole function, which has the following syntax:

Public Overridable Function IsInRole(ByVal role As String) As Boolean

Member of: System.Security.Principal.Iprincipal

But let's briefly move away from the bare theory and try to implement a practical example. To do this, create a new project ASP.NET Web Application and enter the following code:

Default.aspx:

<%@ Page Language=" vb"AutoEventWireup=" false"Codebehind=" default.aspx.vb "Inherits=" AuthSample.WebForm1"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>

<head>

<title>Authentication Sample< / title>

<meta name= "GENERATOR" content= "Microsoft Visual Studio .NET 7.1">

<meta name= "CODE\_LANGUAGE" content= "Visual Basic .NET 7.1">

<meta name=vs\_defaultClientScript content= "JavaScript">

<meta name=vs\_targetSchema content="http://schemas.microsoft.com/intellisense/ie5">

< / head>

<body MS\_POSITIONING="GridLayout">

<form id= "Form1" method= "post" runat= "server">

< / form>

< / body>

< / html>

Default.aspx.vb:

Public Class WebForm1

Inherits System.Web.UI.Page

#Region “ Web Form Designer Generated Code “

‘This call is required by the Web Form Designer.

<System.Diagnostics.DebuggerStepThrough ()> Private Sub InitializeComponent()

End Sub

‘NOTE: the following placeholder declaration is required by the Web Form Designer.

‘Do not delete or move it.

Private designerPlaceholderDeclaration as System.Object

Private Sub Page\_Init (ByVal sender as System.Object, ByVal e as System.EventArgs) Handles MyBase.Init

'CODEGEN: This method call is required by the Web Form Designer

‘Do not modify it using the code editor.

InitializeComponent()

End Sub

#End Region

Private Sub Page\_Load (ByVal sender as System.Object, ByVal e as System.EventArgs) Handles MyBase.Load

Dim s As String

s = " <p><b>Name:</b> “ & HttpContext.Current.User.Identity.Name & " < / p>” & \_

"<p><b> Authentication type:</b > “ & HttpContext.Current.User.Identity.AuthenticationType.ToString & " < / p>” & \_

"<p><b> Is authenticated:</b > “ & HttpContext.Current.User.Identity.IsAuthenticated.ToString & " < / p>” & \_

"<p><b> Is admin:</b > “ & HttpContext.Current.User.IsInRole (”Administrator").ToString & " < / p>”

Response.Write(s)

End Sub

End Class

If Windows authentication mode has been selected and IIS settings have not caused any conflicts, you will receive the appropriate information about your current user. If the username and authentication type fields are empty, you just need to configure IIS by following these steps:

1. Open IIS and locate the virtual directory with this application

2. Open the properties window for this directory and click the directory Security tab. In the Anonymous access and authentication box, click Edit…

This concludes the consideration of the authentication based on Windows and move on to the authentication form.

**Practical task№ 5. Project creation**

To demonstrate the language capabilities in this part of the book, we created a new Visual Studio project ASP.NET MVC 4 Web Application called LanguageFeatures and selected the Empty template option. The features are not specific to MVC, but Visual Studio Express 2012 for web does not support creating projects that can output to the console, so you will have to create an MVC application if you want to follow the examples.

We'll need a simple controller to demonstrate these language features, so we've created a HomeController file.cs in the Controllers folder using techniques.

The initial contents of the home controller.

using LanguageFeatures.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace LanguageFeatures.Controllers

{

public class HomeController : Controller

{

public string Index()

{

return "Navigate to a URL to show an example";

}

}

}

For each example, we will create action methods, so the result of the Index action method is a simple message, so as not to complicate the project. To display the results of our action methods, we added a view named Result.cshtml to the Views/Home folder.

The contents of the file represent the Result

@model String

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title> Result</title>

</head>

<body>

<div>

@Model

</div>

</body>

</html>

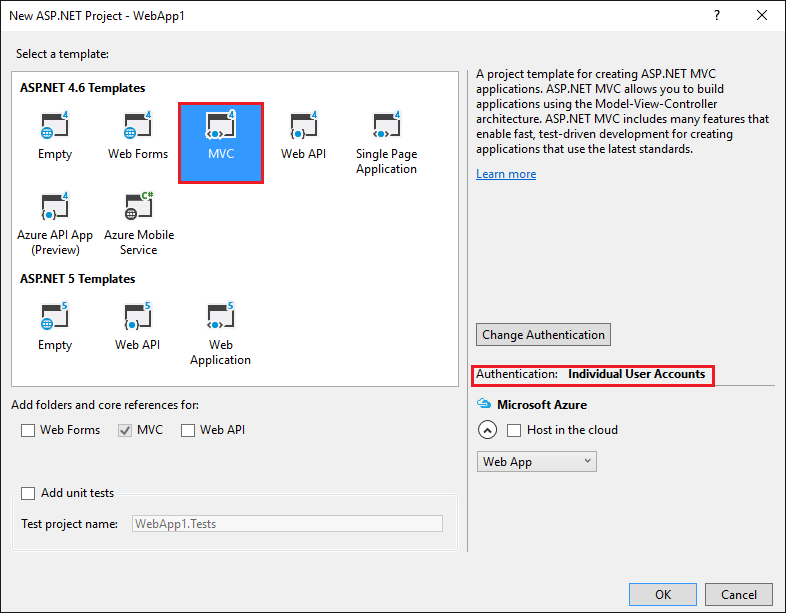
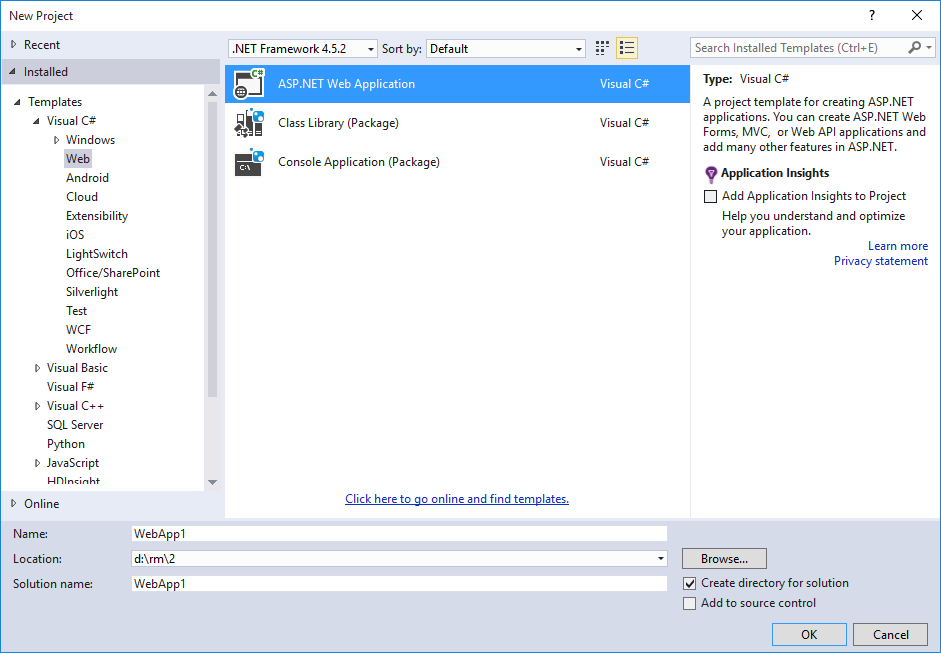
# You can see that this is a strongly typed representation where the model type is String: these are not complex examples, and we can easily present the results as simple strings.

## 2

## Practical task11.The Transition from ASP.NET in MVC 5 to MVC 6

How to start migrating an ASP.NET MVC 5 project to an ASP.NET MVC 6. We will talk about many things that distinguish MVC 5 from MVC 6. The transition from MVC 5 to MVC 6 is a multi-step process, and this article covers initial setup, basic controllers and views, static content, and client-side dependencies. From other articles you will learn about the migration of ASP.NET Identity models, as well as the launch and configuration of many MVC 5 projects. Create MVC 5 project

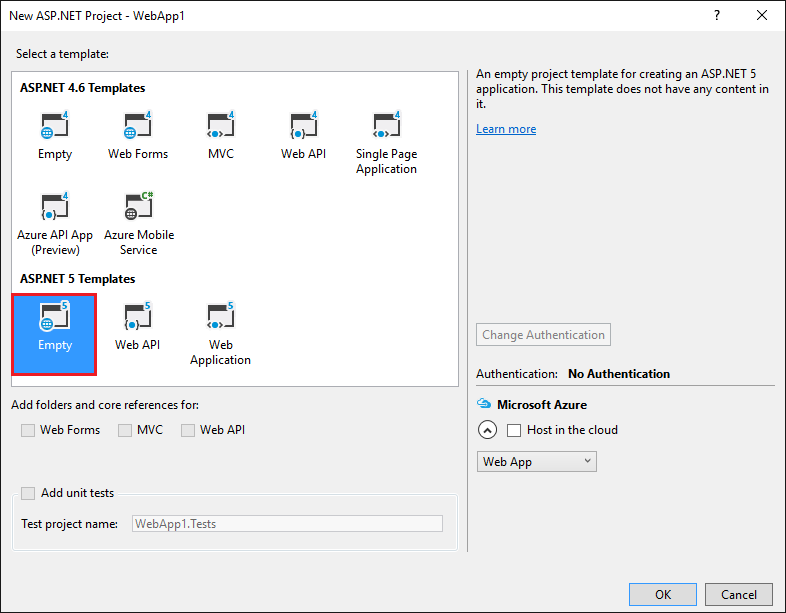
To show the update, we will create a new ASP.NET MVC 5 application. Let's call it WebApp1, so the namespace will correspond to the MVC 6 project that we will create in the next step.



Optional: Change the solution name from WebApp1 to Mvc5. Visual Studio displays the name of the new solution (Mvc5), which makes it easier to call this project from the following. You may need to exit Visual Studio and reload the project to see the new name.

Creating MVC 6 project

We create a new empty MVC 6 web application with the same name as the previous project (WebApp1), so that the namespaces of the two projects match each other. Having two identical namespaces makes it easy to copy code between two projects. In order for a project to use the same name, you need to create it in a different directory than the previous project.



• Optional: Create a new MVC 6 application named WebApp1 with Individual User Accounts authentication. Rename this app to FullMVC6. Creating such a project will save your time during conversion. You can take a look at the code generated by the template to see the final result, or copy the code into the conversion project. This is also useful if you are stuck in the conversion phase when comparing with a project generated by a template.

Setting up a site to use MVC

• Open the project.json file and add Microsoft.AspNet.Mvc and Microsoft.AspNet.StaticFiles to the dependencies property and the scripts section, as shown below:

{

    "version": "1.0.0- \*",

    "compilationOptions": {

        "emitEntryPoint": true

    },

    "dependencies": {

        "Microsoft.AspNet.StaticFiles": "1.0.0-rc1-final",

        "Microsoft.AspNet.Mvc": "6.0.0-rc1-final",

        "Microsoft.AspNet.IISPlatformHandler": "1.0.0-rc1-final",

        "Microsoft.AspNet.Server.Kestrel": "1.0.0-rc1-final"

    },

    "scripts": {

        "prepublish": ["npm install", "bower install", "gulp clean", "gulp min"]

    }

}

Microsoft.AspNet.StaticFiles is a static file handler. ASP.NET runtime is modular, and you must make a direct choice to serve static files

The scripts section is used to indicate when certain automatic scripts should be run. Visual Studio now has built-in support for running scripts before and after certain events. The scripts section above defines the NPM, Bower and Gulp scripts that should be run at the pre-posting stage. Later we will talk about NPM, Bower and Gulp. Notice the "," at the end of the publishExclude section.

• Open the Startup.cs file and change the code to the following:

 public class Startup

{

    // This method gets called by the runtime. Use this method to add services to the container.

    // For your information, visit http://go.microsoft.com/fwlink/?LinkID=398940

    public void ConfigureServices (IServiceCollection services)

    {

    services.AddMvc ();

    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.

    public void Configure (IApplicationBuilder app)

    {

        app.UseIISPlatformHandler ();

   app.UseStaticFiles ();

   app.UseMvc (routes =>

        {

            routes.MapRoute (

                name: "default",

                template: "{controller = Home} / {action = Index} / {id?}");

        });

    }

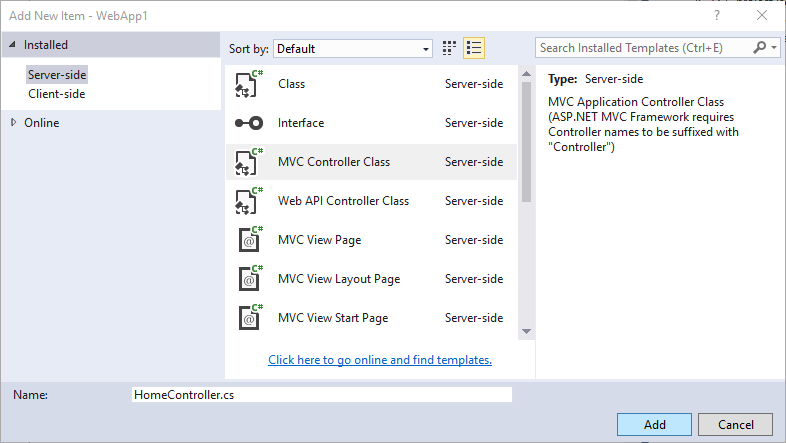
UseStaticFiles adds a static file handler. As mentioned earlier, ASP.NET runtime is modular, and you must make a direct choice to process static files.

Add controller and view

In this section, we will add a controller and a view that will serve as substitutes for the MVC 5 controller and the views that you transfer later.

• Add the Controllers folder.

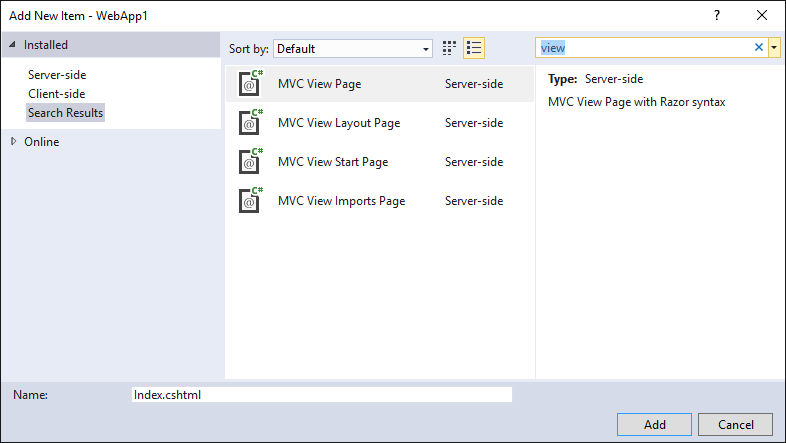
• Add an MVC controller class named HomeController.cs to the Controllers folder.



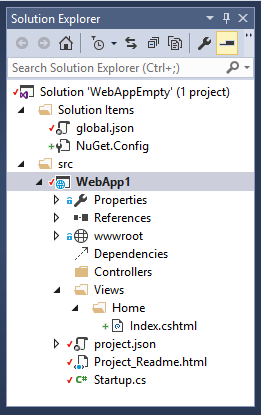
• Add a Views folder.

• Add the Views / Home folder.

• Add the MVC Index.cshtml view to the Views / Home folder.



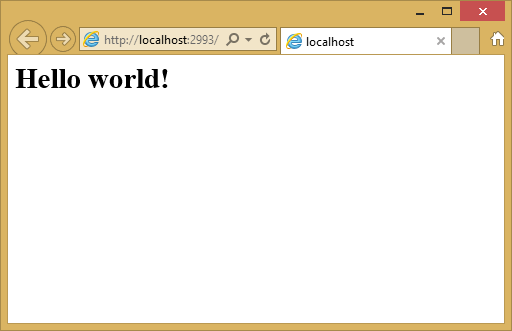
The project structure is shown below:



Replace the context of the Views / Home / Index.cshtml file with the following:

<h1> Hello world! </ h1>

Run the application.



Now we have a small working MVC 6 project, and we can begin to transfer functionality from the MVC 5 project. We need to transfer the following:

• client-side content (CSS, fonts and scripts)

• controllers

•representation

• models

• binding mechanism

• filters

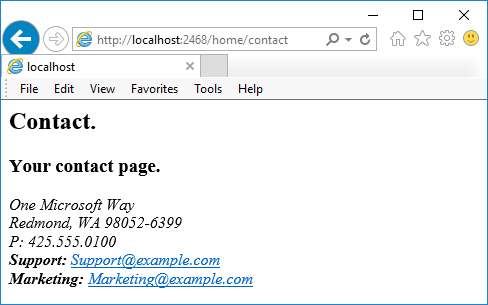
Controllers and Views

• Copy each method from MVC 5 HomeController to MVC 6 HomeController. Note that in MVC 5, the action method of the embedded template controller returns the type ActionResult; in MVC 6, templates generate IActionResult methods. ActionResult is the only implementation of IActionResult, so you don’t need to change the return type of action methods.

• Remove Views / Home / Index.cshtml view from MVC 6 project.

• Copy the Razor files of About.cshtml, Contact.cshtml and Index.cshtml views from the MVC 5 project to the MVC 6 project.

• Run the MVC 6 application and test each method. We have not yet transferred the file with the layout and styles, so that the displayed views will contain only the context of the view files. You will not have generated links with layout for the About and Contact views, so you need to call them from the browser (replace 2468 with the port number that is used in the important project).



Note the lack of styles and menu items. Soon we will fix it.

Static content

In previous versions of MVC (including MVC 5), static content was hosted from the web project's root directory and was mixed with server-side files. In MVC 6, static content is hosted in the wwwroot folder. You copy static content from your MVC 5 application to the wwwroot folder of your MVC 6 project. Conversion for this example:

• Copy the favicon.ico file from the MVC 5 project to the wwwroot MVC 6 folder of the project.

In the MVC 5 project, Bootstrap is used for styling, and Bootstrap files are saved in the Content and Scripts folders. The project generated from the MVC 5 template refers to Bootstrap in the layout file (Views / Shared / \_Layout.cshtml). You can copy the bootstrap.js and bootstrap.css files from the MVC 5 project to the wwwroot folder of the new project, but this approach does not use the improved client-side dependency management in ASP.NET 5.

In the new project, we will add support for Bootstrap (and other client side libraries) using Bower:

• Add a Bower configuration file named bower.json to the project root directory (right-click on the project and then click Add> New Item> Bower Configuration File). Add Bootstrap and jquery to the file (see the code below).

{

    "name": "ASP.NET",

    "private": true

    "dependencies": {

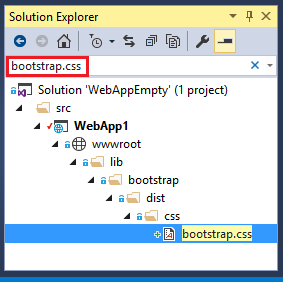
        "bootstrap": "3.3.5",

        "jquery": "2.1.4"

    }

}

When saving a file, Bower automatically downloads the dependencies to the wwwroot / lib folder.

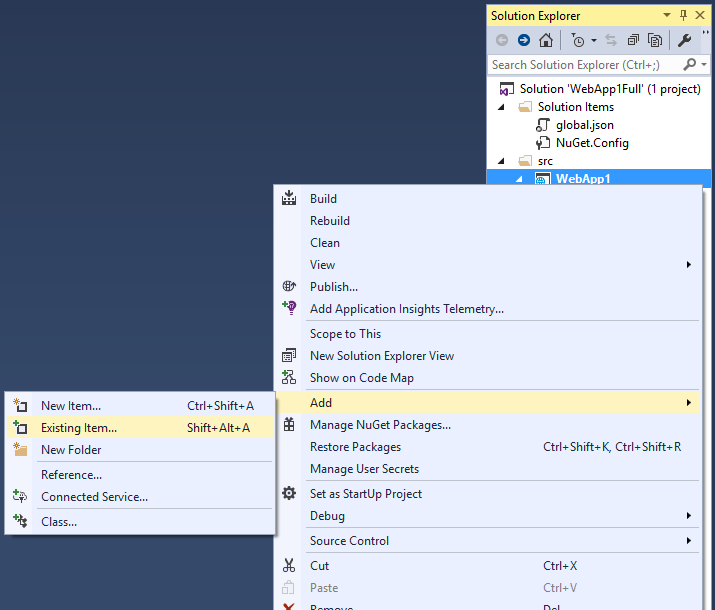


You can use Search Solution Explorer to find the path to assets.

Gulp

When you create a new application using an ASP.NET 5 Web Application template, the project is configured to use Gulp. Gulp is a streaming build system for client side code (HTML, LESS, SASS, etc.). The included gulpfile.js contains JavaScript that defines a set of gulp tasks that can be automatically run for events, or you can run them manually using Task Runner Explorer in Visual Studio. In this section, we will show you how to use the MVC 6 gulpfile.js file to link and reduce JavaScript and CSS files in a project.

If you created an additional FullMVC6 project (a new ASP.NET MVC 6 web application with Individual User Accounts), add gulpfile.js from this project to the project we are updating. In Solution Explorer, right-click on the project and select Add> Existing Item.



Transferring configuration settings from web.config

Our ASP.NET MVC 5 project includes the required database connection string in web.config in the <connectionStrings> element. In our MVC 6 project, we will store this information in the appsettings.json file. Open appsettings.json and you will see the following in it:

{

"Data": {

"DefaultConnection": {

"ConnectionString": "Server = (localdb) \\ MSSQLLocalDB; Database = \_CHANGE\_ME; Trusted\_Connection = True;"

}

}

}

In the line selected above, change the database name \_CHANGE\_ME. The new database will be called NewMvc6Project, which corresponds to the name of our portable project.

4 Counter of visits for a site on ASP.NET MVC

The principle of building a counter for ASP.NET MVC has much in common with the method discussed earlier for Web Forms. The Request property is also used to obtain information about the visitor. But, unlike Web Forms, in ASP.NET MVC Request is a property of the request context object (HttpContext). Therefore, in order to get the same information about the visitor in ASP.NET MVC, you must access the HttpContext property of the controller.

C #

public ActionResult Index ()

{

    string userHostAddress = this.HttpContext.Request.UserHostAddress;

    string userAgent = this.HttpContext.Request.UserAgent;

    string url = this.HttpContext.Request.Url.ToString ();

    return View ();

}

 Thus, the procedure for obtaining information about the visitor is not much different from Web Forms. The only difference is that information is collected in the controller methods and the code itself becomes more voluminous.

After collecting information about the visitor, it must be saved in a database (DB). To do this, you need to refer to the model, which is designed to ensure the operation of the counter, or the model method created specifically for this purpose for which this controller was written. The first method for architectural reasons is preferable.

Create a model for the counter. The model class will implement the “Repository” pattern in conjunction with LINQ to SQL. In this article, only a constructor and a single method that adds data to the database will be quite sufficient for the model class.

public class StatisticRepository

{

    private dbDataContext \_dc;

    public StatisticRepository (dbDataContext dc)

    {

        \_dc = dc;

    }

    public Statistic CreateStatisticRecord (string ip, string userAgent, string url)

    {

        Statistic s = new Statistic ();

        s.IP = ip;

        s.UserAgent = userAgent;

        s.URL = url;

        s.VisitDate = DateTime.Now;

        \_dc.Statistic.InsertOnSubmit (s);

        \_dc.SubmitChanges ();

        return s;

    }

}

 If you refer to the created model in the corresponding controller methods, then the attendance data will be saved in the database.

C #

public ActionResult Index ()

{

    statisticRepository. CreateStatisticRecord (this.HttpContext.Request.UserHostAddress, this.HttpContext.Request.UserAgent, this.HttpContext.Request.Url.ToString ());

    return View ();

}

 It is best to contact the counter model at the very beginning of the method in order to immediately save the visitor information in the database.

From the functional point of view, the visitor counter that was developed in this article is a complete analog of the counter developed earlier for Web Forms. The possibilities of this counter, if necessary, can also be expanded. However, due to the use of the MVC pattern, the implementation of this counter is much more complicated and cumbersome.

The Web API represents a different way to build an ASP.NET application that is slightly different from ASP.NET MVC. A web API is a web service that can interact with various applications. In this case, the application can be an ASP.NET web application, or it can be a mobile or a regular desktop application.

It should also be noted that the Web API 2 platform is not part of the ASP.NET MVC framework and can be used in conjunction with MVC, and in conjunction with Web Forms. Therefore, the Web API has its own versioning system. So, the first version appeared with .net 4.5. And along with .NET 4.5.1 and MVC 5, Web API 2.0 was released.

To understand the work of the Web API Web services, create a simple template application. When creating a new project, specify as a Web API template:

A standard Web API project after creation will look like a regular mvc 5 project with a few modifications:

By default, there will already be two controllers in the Contollers folder: the standard HomeController controller and the -ValuesController controller, which implements the Web API functionality:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Http;

using System.Web.Http;

namespace WebApiApp.Controllers

{

    public class ValuesController: ApiController

    {

        // GET api / values

        public IEnumerable <string> Get ()

        {

            return new string [] {"value1", "value2"};

        }

        // DELETE api / values ​​/ 5

        public void Delete (int id)

        {}

    }

}

The definition of a Web API controller is different from a regular MVC controller. First, the main functionality is concentrated mainly in the namespace System.Web.Http, so it connects at the beginning of the file.

Secondly, it is the successor of the ApiController class, which is in no way connected with the base controller class for MVC - Controller

Third, the Web API controllers use the REST (Representation State Transfer) style or "view state transfer".

The REST architecture assumes the use of the following methods or types of HTTP requests to interact with the server:

• GET

• POST

• PUT

• DELETE

• PATCH

In the ValuesContoller controller, there are no standard action methods that return an ActionResult, as in regular controllers. The methods defined in the Web API controller are mapped to HTTP methods of the same name.

Web API Routing

Another feature of the Web API project is the presence of the WebApiConfig.cs file (in the App\_Start folder), which contains the definitions of the Web API routes:

public static class WebApiConfig

{

    public static void Register (HttpConfiguration config)

    {

        // Web API configuration and services

        // routes web API

        config.MapHttpAttributeRoutes ();

        config.Routes.MapHttpRoute (

            name: "DefaultApi",

            routeTemplate: "api / {controller} / {id}",

            defaults: new {id = RouteParameter.Optional}

        );

    }

}

In this case, one route is defined, where the controller is the second parameter, and the third optional parameter represents some identifier. Thus, unlike the routes of conventional controllers, we have no action here, only the controller and the optional optional parameter

As a result, the api / values ​​call will correspond to the call to the ValuesCotroller controller, and almost to all actions at once (except for Get (int id) - since in this case an identifier is also needed, for example, api / values ​​/ 2)

But, as mentioned above, depending on the HTTP method used, the framework will distinguish which action the current request belongs to.

For example, the GET api / values ​​request will be matched with the IEnumerable <string> Get () method and will return to the browser a collection of string elements. At the same time, if the api / values ​​/ 7 parameter is passed in the GET request, then the string Get (int id) method will correspond to this request, since it takes a parameter.

If the server receives a PUT request with the address api / values, then such a request will be associated with the Put method.

To enable Web API routing, the following line is added to the Application\_Start () method in the Global.asax file: GlobalConfiguration.Configure (WebApiConfig.Register);

Convention on naming methods

When creating the methods of the Web API controller, some conventions apply. So, the default method names must begin with the name of their intended HTTP method. In the case of the default controller, everything is simple: all action methods are called HTTP methods.

But we do not have to strictly adhere to these conventions, and we can use any other names without prefixes. True, in this case we will need to explicitly specify the HTTP method as an attribute, for example:

public class ValuesController: ApiController

{

    public IEnumerable <string> GetAllItems ()

    {

        return new string [] {"value1", "value2"};

    }

    public string GetItem (int id)

    {

        return "value";

    }

    [HttpPost]

    public void CreateItem ([FromBody] string value)

    {}

    [HttpPut]

    public void EditItem (int id, [FromBody] string value)

    {}

    [HttpDelete]

    public void RemoveItem (int id)

    {}

}

Thus, if the first two actions correspond to the naming conventions (they contain the name of the HTTP method at the beginning), then we apply attributes to the rest so that the system knows which method to match the request with.

## Practical task2 Transferring configuration from ASP.NET in MVC 5 with MVC 6

By default, our project already has an Index.cshtml view with some standard markup. Change it to interact with the server.

Our visual interface will be very simple: there will be three blocks on the page. The first block (tableBlock) will be designed to display information about all books, the second block (editBlock) will contain the book edit form, and the third block (createBlock) will include the form for adding a new book. In this case, the second and third blocks will overlap each other, and depending on what we need - to create a new book or edit an existing book, we will make one of the blocks visible, controlling the css-style display.

So, change the Index.cshtml file as follows:

<div id = "tableBlock"> </ div>

<div id = "editBlock"> <p> <b> Editing a book </ b> </ p>

    <table>

    <tr> <td> <input type = "hidden" id = "editId" /> </ td> <td> </ td> </ tr>

    <tr> <td> <label> Name: </ label> </ td> <td> <input type = "text" id = "editName" /> </ td> </ tr>

    <tr> <td> <label> Author: </ label> </ td> <td> <input type = "text" id = "editAuthor" /> </ td> </ tr>

    <tr> <td> <label> Year: </ label> </ td> <td> <input type = "number" id = "editYear" /> </ td> </ tr>

        </ table>

    <button id = "editBook"> Save </ button>

</ div>

<script type = "text / javascript">

    $ (document) .ready (function () {

        GetAllBooks ();

        $ ("# editBook"). click (function (event) {

            event.preventDefault ();

            EditBook ();

        });

        $ ("# addBook"). click (function (event) {

            event.preventDefault ();

            AddBook ();

        });

     }

    // book request for editing

    function GetBook (id) {

        $ .ajax ({

            url: '/ api / values ​​/' + id,

            type: 'GET',

            dataType: 'json',

            success: function (data) {

                ShowBook ​​(data);

            },

            error: function (x, y, z) {

                alert (x + '\ n' + y + '\ n' + z);

            }

        });

    }

</ script>

}

Since jquery code is used to organize requests to the server, you need to make sure that the jquery library is already connected on the master page, by default it is connected at the bottom of the page. The rest of the javascript code that uses jquery is added to the scripts section, which is defined by default on the master page after the jquery library is enabled.

Also add to the stylesheet the style definitions for the blocks:

div {

    display: inline-block;

    margin: 20px;

}

#tableBlock {

    width: auto;

}

#editBlock {

    display: none;

    float: right;

}

#createBlock {

    float: right;

}

As a result, after loading the page in a web browser, we get all our books from the database:

We can edit an existing book by clicking on the 'Edit' link in the corresponding book. And then we will have a visible form for editing, the fields of which will be filled with the values ​​of this book:

Now we will sort the presentation code. When loading the page in the browser in javascript code, we simultaneously retrieve all the books from the database using the GetAllBooks function:

function GetAllBooks () {

    $ ("# createBlock"). css ('display', 'inline-block');

    $ ("# editBlock"). css ('display', 'none');

    $ .ajax ({

        url: '/ api / values ​​/',

        type: 'GET',

        dataType: 'json',

        success: function (data) {

            WriteResponse (data);

        },

        error: function (x, y, z) {

            alert (x + '\ n' + y + '\ n' + z);

        }

    });

}

The function switches the visibility of createBlock and editBlock blocks and sends a request to the server. Since this is a GET request without a parameter, it will be mapped to the GetBooks controller method, and the data obtained from this method will be displayed on the page via the WriteResponse function.

In the WriteResponse function, for each 'Edit' link we hang the EditItem click handler (this), and the DeleteItem handler (this) on the 'Delete' link. Both of these handlers take the <a ...> element as a parameter and then get it through the data-item id attribute of the pressed book.

The DeleteItem handler calls the DeleteBook method, passing the id of the book to be deleted to it:

function DeleteBook (id) {

    $ .ajax ({

        url: '/ api / values ​​/' + id,

        type: 'DELETE',

        contentType: "application / json; charset = utf-8",

        success: function (data) {

            GetAllBooks ();

        },

        error: function (x, y, z) {

            alert (x + '\ n' + y + '\ n' + z);

        }

    });

}

Since this is a Delete request, it calls the DeleteBook controller method.

Editing is divided into two stages: loading data into the editing form and sending new values ​​to the server. To download books from the data server, use the GetBook function:

function GetBook (id) {

    $ .ajax ({

        url: '/ api / values ​​/' + id,

        type: 'GET',

        dataType: 'json',

        success: function (data) {

            ShowBook ​​(data);

        },

        error: function (x, y, z) {

            alert (x + '\ n' + y + '\ n' + z);

        }

    });

}

Since this is a GET request with a parameter, it calls the public Book GetBook (int id) method on the controller, and sends the book received as a response for output to the ShowBook ​​function.

The second stage of editing is sending updated data by pressing a button on the form. As we in the jquery function hung EditBook handler on it. then it will send new data in a PUT request (for which the EditBook method responds to the controller):

function EditBook () {

    var id = $ ('# editId'). val ()

    // get new values ​​for editable book

    var book = {

        Id: $ ('# editId'). Val (),

        Name: $ ('# editName'). Val (),

        Author: $ ('# editAuthor'). Val (),

        Year: $ ('# editYear'). Val ()

    };

    $ .ajax ({

        url: '/ api / values ​​/' + id,

        type: 'PUT',

        data: JSON.stringify (book),

        contentType: "application / json; charset = utf-8",

        success: function (data) {

            GetAllBooks ();

        },

        error: function (x, y, z) {

            alert (x + '\ n' + y + '\ n' + z);

        }

    });

}

Similar to loading in a PUT request, it will be possible to create a new book and send data to the server in a POST request in the AddBook function.

## Practical task№3. Introduction to Web API

Application programming interface (Application Programming Interfaces, APIs) - is a ready design of a programming language that allows the developer to build complex functionality with less effort. They "hide" more complex code from the programmer, providing ease of use.

For a better understanding, consider the analogy with home power grids. When you want to use some electrical appliance, you just plug it into an outlet and everything works. You are not trying to connect the wires directly to the current source — it is useless and if you are not an electrician, it is difficult and dangerous.

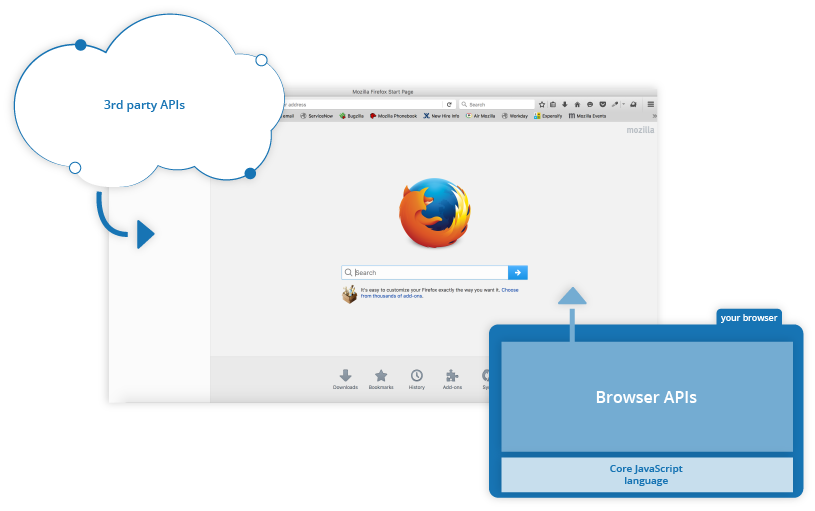
Similarly, if we want to program 3D graphics, for example, it is much easier to do so using APIs written in high-level languages such as JavaScript or Python.

Client-side API JavaScript Section

For client-side JavaScript, in particular,there are many APIs. They are not part of the language, but are built with built-in JavaScript functions to increase your ability to write code. They can be divided into two categories:

Browser APIs are built into the web browser and are able to use browser and computer environment data to perform more complex actions with this data. For example, the Geolocation API (Geolocation API) provides easy-to-use JavaScript constructs to work with location data so that you can, for example, mark your location on a Google Map. In fact, the browser runs complex low-level code (like in C++) to connect to a GPS device (or any other geolocation device), retrieve data, and send it to the browser for processing by your program, but as mentioned above, these details are hidden thanks to the API.

Third-party APIs are not built into the browser by default. Such APIs and information about them usually need to be searched on the Internet. For example, the Twitter API allows you to post the latest tweets on your website. This API defines a set of constructs that query Twitter services and return specific data.



The interaction of JavaScript, APIs, and other tools JavaScript

So, above we talked about what The JavaScript API of the client part is and how they are related to the JavaScript language. Let's now write down the basic concepts and define the purpose of other JavaScript tools:

JavaScript is a high-level scripting Language built into the browser that allows you to create the functionality of web pages/applications. Note that JavaScript is also available on other software platforms, such as Node. But we will not stop there yet.

Browser API (Browser APIs) — constructions built into the browser, built on the basis of JavaScript language, designed to facilitate the development of functionality.

Third — party APIs-designs embedded in third-party platforms (such as Twitter, Facebook) that allow you to use some of the functionality of these platforms in your own web pages/applications (for example, to show the latest Tweets on your page).

JavaScript libraries — Usually one or more files containing user defined (custom) functions . Such files can be attached to a web page to speed up or provide tools for writing General functionality. Examples include jQuery, Mootools, and React.

JavaScript frameworks — frameworks) - the Next step in development after libraries. JavaScript frameworks (such as Angular and Ember) aim to be a set of HTML, CSS, JavaScript and other technologies, after which you can "write" a web application from scratch. The main difference between frameworks and libraries is "inversion of Control". The method is called from the library at the request of the developer. When using the framework - on the contrary, the framework makes a call to the developer code.

That is capable of API?Section

A wide variety of API in modern browsers allows you to give your application more features. Just look at the list on the MDN APIs index page.

Common API browsermaster

In particular, the most commonly used API categories (which we'll look at later in this module) are :

API for working with documents loaded into the browser. A clear example is the DOM (Document Object Model) API, which allows you to work with HTML and CSS — create, delete and modify HTML, dynamically change the page view, etc.Any pop - up window on the page or content that appears "on the go" - all thanks to DOM. Learn more about this API category on the working with documents page.

APIs that receive data from the server are often used to refresh small parts of a web page. This seemingly small detail has a huge impact on the performance and behavior of the sites, as there is no need to reload the entire page if you just need to update the list of products or new stories available. It will also make the app or website more responsive and "alive". The list of APIs that make this possible includes: XMLHttpRequest and Fetch API. You might also come across the term Ajax describing this technology.

To learn more about this API, see get data from the server.

The graphics APIs are widely supported by browsers, the most popular being Canvas and WebGL, which allow you to programmatically modify the pixel data contained in the HTML <canvas> element to create 2D and 3D images. For example, you can draw shapes such as rectangles or circles, import an image into a canvas and apply filters such as Sepia or grayscale using the Canvas API, or create a complex 3D image with lighting and text using WebGL. Such APIs are often used in conjunction with the animation looping API (such as window.requestAnimationFrame () and others to create an ever-changing image on the screen, like in cartoons or games .

Audio and Video APIs like HTMLMediaElement, Web Audio API, and WebRTC allow you to do really interesting things with multimedia. For example, create your own user interface (User Interface, UI) to play audio/video, display subtitles, record video from a webcam for processing in canvas (see above) or to transfer to another computer in a video conference, apply sound effects to audio files (such as gain, distortion, panning, etc.).

Device APIs - mainly API to parse and read data from modern devices user-friendly web applications. We've already talked about the Geolocation API, which allows you to read the location data of the device. Other examples include notifying the user when an update is available for a web application using system notifications (see Notifications API) or vibration (see Vibration API).

User-side storage APIs are becoming more common in web browsers — the ability to store client-side information is very useful when you need to create an application that will save its state between page reloads, or even work when the device is offline. Many of these APIs are available at this point. For example, a simple data store in the name/value (name/value) format of the Web Storage API or a data store in the IndexedDB API table format.

Common third-party Section

There are many third-party APIs; some of the most popular you'll use sooner or later include:

Twitter API to add functionality such as showing the latest tweets on the site.

Google Maps API to work with maps on a web page (interestingly, Google Maps also uses this API). Now it's a whole set of APIs that can handle a wide range of tasks, as evidenced by the Google Maps API Picker.

The Facebook API Suite allows you to use different parts of the Facebook platform in your app, such as the ability to log in with your Facebook login, pay for in-app purchases, display targeted ads, etc.

YouTube API that allows you to embed YouTube videos on your website, search, create playlists, etc.

Twilio API is a framework for embedding voice and video communication functionality in your application, sending SMS/MMS from the application, etc.

How does the API?Section

The work of different JavaScript APIs is a bit different, but mostly they have similar functions and working principle.

They are based on the objects of a partition

Interaction with the API in the code occurs through one or more JavaScript objects that serve as containers for the information that the API works with (contained in the object properties) and implement the functionality that the API provides (contained in the object methods).

Let's go back to the example with the Geolocation API — a very simple API consisting of several simple objects:

* Geolocation, contains three methods for monitoring and retrieving GEODATA.
* Position, provides data about the location of the device at a given time-contains Coordinates-object that stores the coordinates and the current time stamp.
* Coordinates, contains a lot of useful information about the location of the device, including latitude and longitude, altitude, speed and direction of movement, etc.

## Practical task №4. Routing System in the web API

For routing requests to specific routes, as in MVC, the routing system is responsible. The key class for the routing system is the HttpRoutingDispatcher class, which processes the request to retrieve route data and adds this data to the HttpRequestContext.RouteData collection.

Another key type is the IHttpRoute interface, which describes the route. The Web API provides an embedded implementation of this interface in the form of the HttpRoute class. To manage the route, IHttpRoute provides the following properties:

• RouteTemplate: URL pattern that is used to match with the request.

• Defaults: returns an IDictionary <string, object> object containing parameter sets and their default values ​​as key-value pairs.

• Constraints: contains a set of route constraints in the form of an IDictionary <string, object>

• DataTokens: returns data of parameters and their route values ​​in the form of an IDictionary <string, object>

• Handler: returns the route handler - HttpMessageHandler object

The HttpRequest.RouteData object storing the route parameters extracted from the request represents the implementation of the IHttpRouteData interface. Web Api also introduces the embedded implementation of this interface - the HttpRouteData class. The IHttpRouteData interface defines two properties:

• Route: IHttpRoute object representing the current route

• Values: An IDictionary <string, object> containing route data.

Although each request is matched with only one route, several routes can be defined in an application. They all fall into the HttpRouteCollection collection. The following methods and properties are defined for managing routes in a class:

• CreateRoute (template, defaults, constraints): creates an IHttpRoute object that accepts the specified URL pattern, default values ​​and constraints

• CreateRoute (template, defaults, constraints, tokens): the same, plus the route accepts route value tokens

• CreateRoute (template, defaults, constraints, tokens, handler): the same, plus the route accepts route handler

• Add (name, route): adds a new route.

• Clear (): deletes all routes.

• Contains (route): returns true if the collection contains a route.

• Insert (index, name, route): adds a route route named name at a specific index to the collection

• Remove (name): removes a route with a specific name.

• this [int]: gets the route from the collection at a specific index.

• this [name]: gets a route with a specific name

• IgnoreRoute (name, template): registers a route with a specific name and a template that will not be processed by the Web API

• IgnoreRoute (name, template, constraints): registers a route with a specific name and a pattern and constraints constraints that will not be processed by the Web API

• MapHttpRoute (name, template): creates and registers a route with a specific name and pattern

• MapHttpRoute (name, template, defaults): creates and registers a route with a specific name, pattern, and default values

• MapHttpRoute (name, template, defaults, constraints): creates and registers a route with a specific name, pattern, default values ​​and constraints

• MapHttpRoute (name, template, defaults, constraints, handler): creates and registers a route with a specific name, pattern, default values, constraints, and handler

Defining default routes is done in the WebApiConfig.cs file. If we open this file, we will already see the registration of one route:

|  |  |
| --- | --- |
|  | public static class WebApiConfig  {      public static void Register(HttpConfiguration config)      {          config.MapHttpAttributeRoutes();            config.Routes.MapHttpRoute(              name: "DefaultApi",              routeTemplate: "api/{controller}/{id}",              defaults: new { id = RouteParameter.Optional }          );      }  } |

We can access the route collection using the Routes property defined in the Web configuration class Api HttpConfiguration

Route Patterns

The template allows you to match the request with a specific route. In the Web API, templates are defined as in MVC. In particular, the “api / {controller} / {id}” template is defined here, where the id parameter is optional. Therefore, this route will correspond, for example, to the query api / values ​​/ 5, where the "values" will be matched with the controller name, and the number 5 with the id parameter. Since the Web API uses the HTTP request type (POST / PUT / GET) to match the controller method, we can omit the segment for the method in the template.

Note the api prefix that starts the template. Although it is used by default and is often used in real Web Api applications, its presence is optional. The meaning of its use is to distinguish between Web API and MVC routes, since by default ASP.NET MVC + Web API application registers both mvc routes and Web Api routes. If they are identical, then a mapping problem may arise, especially if the MVC route and the Web Api route use the same pattern.

At the same time, this is not a mandatory form of route determination. We can use a template that includes the controller method as well. For example, change the class in the WebApiConfig.cs file as follows:

public static class WebApiConfig

{

    public static void Register (HttpConfiguration config)

    {

        config.MapHttpAttributeRoutes ();

        config.Routes.MapHttpRoute (

            name: "BookRoute",

            routeTemplate: "api / {controller} / {action}"

        );

                   }

}

And if we have defined in the controller, for example, the method GetValue:

public class ValuesController: ApiController

{

    public string GetValue ()

    {

        return "getvalue";

    }

}

then we can access this method using the query: api / values ​​/ getvalue

Default settings

If we provide default values ​​for a certain segment, this segment may be omitted when matching the route with the request. For example,

routes.MapHttpRoute (

    name: "DefaultApi",

    routeTemplate: "api / {controller} / {id}",

    defaults: new {id = "5"}

);

Even if we do not use a segment for id in the request: http: // localhost / api / values, the request will still correspond to the route, so the default value is used, that is, the id will be 5 in this case.

Attribute NonAction

If we do not want the method to be associated with a specific route, then we can mark it with the NonAction attribute:

[NonAction]

public string GetValue ()

{

    return "getvalue";

}

Now it will not process requests, even if it is the only method suitable for this.

**Practical task№ 5.** Web API design in 7 steps

Step 1: List all components

The first step is to list all the types of data that the client application may want to receive or transmit using the service. We call it a semantic description. Semantic-because it displays the value of the data in the application, and the description – because it contains a description of what is happening in the application. Note that you should be working from a client application perspective, not from a service perspective. It is important to develop a user-friendly API for use by the client.

For example, a simple application like "to-do List", you can find the following semantic descriptions:

id-a unique identifier for each record in the system

title-title for each " case»

dateDue – the date by which the "case" must be completed

complete - "Yes/no" checkbox that shows whether the "case is completed»

In this application, there can be a lot of items to display concepts such as the category of "Affairs" (work, family, garden, etc.), user information and other things. Now it is better to stop at such a simple list, so that you can focus on the process itself.

Step 2: Draw a state diagram

The next step is to draw a state diagram for the intended API. Each block in the diagram represents a possible state-a document that includes one or more semantic descriptors found in the first step. You can use the arrows to indicate transitions from one block to another, from one state to the next. These transitions are triggered by requests.

There is no need to worry about determining which method is used for each transition. Just indicate whether the transition is safe (HTTP GET), unsafe/non idempotent (HTTP POST), or unsafe/idempotent (PUT)

Notes in the margin: Idempotent actions are those that can be repeated without unexpected side effects. For example HTTP PUT is idempotent because the specification says that the server should use the status values received from the client, to replace any values in the target resource. While HTTP POST is not idempotent, since the HTTP specification specifies that values passed by POST should be added to existing resources rather than replaced.

In our case, the client application for the simplest "to-do List" may require access to the list items, the ability to filter, view individual items and mark them as completed. Many of these actions use state values to transfer data between the client and the server. For example, the add item action allows the client to pass title and dueDate state values. Below is a diagram that illustrates the basic steps

The actions shown in the diagram and listed below are also semantic descriptors-they describe the semantics of actions for the service.



read-list

filter-list

read-item

create-item

mark-complete

While working on the chart, you may find that you have missed some actions or data that the client requires. In this case, it is a great opportunity to go back a step and enter the missing data and improve the chart in step 2.

Once you go through these two steps a couple of times, you will have a good understanding of all the data and actions that the customer needs to interact with your service.

Step 3: Match magic strings

The next step is to reconcile all the "magic strings" in the interface of your service. The "magic strings" in our case are all descriptor names – they don't make sense outside of the task, they just represent the actions or data items that clients will access when communicating with the service. Name matching means bringing them to commonly used terms using, for example, the following resources:

schema.org

microformats.org

Dublin Core

IANA Link Relation Values

These are all repositories for well-defined, widely used names. When you use the names of these services, make sure the developers understand them as well as you do. This process can greatly improve the usability of your API.

Field notes: using common names for descriptors may be a good idea for the frontend, however no one is forcing you to use them for your internal needs. We mean, for example, names for databases. The service itself can use the map of correspondences between external and internal names without any problems.

For the To-Do service from the example, I managed to find acceptable existing names for all descriptors except one – "create-item". In this case, I turned to creating a unique URI based on rules from Web-Linking RFC 5988. During the selection of the Convention names for the interface, you will always be pursued by compromises. It is rare to find the perfect hit to the internal names and this is normal.

That's what I got:

id - >identifier from Dublin Core

title ->name from Schema.org

dueDate ->scheduledTime from Schema.org

complete ->status from Schema.org

read-list ->collection from IANA Link Relation Values

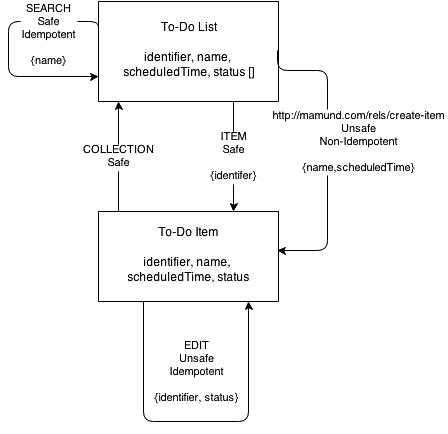
filter-list ->search from IANA Link Relation Values

read-item ->item from IANA Link Relation Values

create-item -> http://mamund.com/rels/create-item using RFC5988

mark-complete - >edit from IANA Link Relation Values

So here's what the chart looks like after using name matching:



Step 4: choose the type of hypermedia

The next step in designing your API is to choose the type of data that will be used to send messages between the server and the client. One of the distinguishing features of the network is that data is transmitted by standardized documents through a common interface. It is very important to choose a type that supports the same data descriptors ("identifier", "status") and actions ("search", "edit"). Such formats are quite small.

Here are some of the hypermedia formats from the top of the list (the order doesn't matter in this list):

HyperText Markup Language (HTML)

Hypertext Application Language (HAL)

Collection+JSON (Cj)

Siren

JSON API

Uniform Basis for Exchanging Representations (UBER)

The choice should also be influenced by how well the chosen hypermedia format works with the data transfer Protocol. Most developers prefer the HTTP Protocol for the service interface. However, WebSockets, XMPP, MQTT and CoAP are also used – especially for high-speed implementations, with short messages, point-to-point connections.

For our example, I will use HTML as the message Protocol and HTTP as the communication Protocol. HTML already has support for the necessary descriptors (<UL> for lists, <LI> for elements, and <SPAN> for data). It also provides adequate support for action descriptors (<a> for secure links, <FORM method="get"> for secure transitions, and <FORM method="post"> for unsafe transitions).

Field notes: the state Chart now shows "editing" as idempotent (HTTP PUT), but HTML still does not have built-in support for PUT. However, I can add an additional field to emulate a non-HTML POST.

Great, now I can "try out" the interface based on the States from the diagram. For our example, we need to describe only two things: "To-Do List" and " To-Do Item»:

Collection" To-Do List " in HTML view

<html>

<head>

<!-- for test display only -->

<title>To Do List< / title>

<style>

.name .scheduledTime,.status,.item {display:block}

< / style>

< / head>

<body>

<!-- for test display only -->

<h1>To-Do List</h1>

<!-- to-do list collection -->

<ul>

<li>

<a href= "/list/1 "rel=" item " class="item">

<span class= "identifier" >1< / span>

</a>

<span class= "name" >First item in the list</span>

<span class= "scheduledTime">2014-12-01< / span>

<span class= "status" >pending< / span>

< / li>

<li>

<a href= "/list/2 "rel=" item " class="item">

<span class= "identifier" >2< / span>

</a>

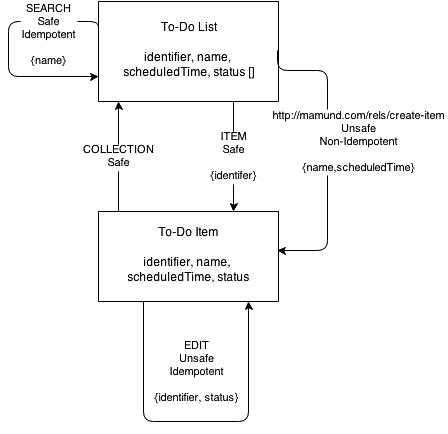
<span class= "name" >Second item in the list</span>

<span class= "scheduledTime">2014-12-01< / span>

<span class= "status" >pending< / span>

< / li>

<li>



<a href= "/list/3 "rel=" item " class="item">

<span class= "identifier" >3< / span>

</a>

<span class= "name" >Third item in the list</span>

<span class= "scheduledTime">2014-12-01< / span>

<span class= "status" >complete< / span>

< / li>

< / ul>

<!-- search transition -->

<form method= " get "action=" /list/ "class= "search">

<legend>Search< / legend>

<input name= "name" class= "identifier" />

<input type= "submit" value= "Name Search" />

< / form>

<!-- create-item transition -->

<form method= "post" action= "/list/ " class=">

<legend>Create Item< / legend>

<input name= "name" class= "name" />

<input name= "scheduledTime" class= "scheduledTime" />

<input type= "submit" value= "Create Item" />

< / form>

< / body>

< / html>

Collection" To-Do Item " in HTML view

…..

<!-- edit transition -->

<form method= "post" action= "/list/1 "class= "edit">

<legend>Update Status< / legend>

<input type= "hidden" name= "etag" value= "q1w2e3r4t5y6" class= "etag" />

<input type= "text" name= "status" value= "pending" class= "status" />

<input type= "submit" value= "Update" />

< / form>

< / body>

< / html>

Keep in mind that when working with an approximate implementation of your state diagram, you may find that something is missing and you need to go back a step or two. It is normal and it is not necessary to be frightened of it. Now is the time to try it all out in test cases – before you start implementing everything in the code.

When you are satisfied with how everything is presented, the last step before coding is to create a semantic profile.

Step 5: create a semantic profile

A semantic profile is a document that lists all the descriptors in your solution and describes the details of each one to help developers create both a client and server implementation.

It should be clearly understood that this is an implementation guide, not an implementation guide.

The formats of the description of the service

Formats for describing the service exist for a long time and are very useful when you want to generate code or document an existing implementation of the service.

The main formats that fight for the hearts of developers:

Web Service Definition Language (WSDL)

Atom Service Description (AtomSvc)

Web Application Description Language (WADL)

Blueprint

Swagger

RESTful Application Modeling Language (RAML)

Profile description formats

There are only a few formats to describe the profile at the moment. And here are the ones I recommend:

Application-Level Semantic Profiles (ALPS)

JSON-LD+ Hydra

Both formats are quite new. JSON-LD specification has received the status of W3C Recommendation in early 2014. Hydra is still in a state of unofficial draft (at the time of writing), but has an active developer community. ALPS is also in the early stages of the draft.

Since the idea of the document is to describe the real aspects of the problem area (rather than a particular solution), the format is quite different from the typical descriptive formats:

<html>

<alps version= "1.0">

<doc>

ALPS profile for InfoQ article on " API Design Methodology"

< / doc>

<!-- data descriptors -->

<descriptor id= "identifier" type= "semantic" ref=" />

<descriptor id= "name" type= "semantic" ref=" />

<descriptor id= "scheduledTime" type= "semantic" ref=" />

<descriptor id= "status" type= "semantic" ref=" />

<!-- action descriptors -->

<descriptor id= "collection" type= "safe" ref=" />

<descriptor id= "item" type= "safe" ref=">

<descriptor href= "#identifier" />

< / descriptor>

<descriptor id= "search" type= "safe" ref=">

<descriptor href= "#name" />

< / descriptor>

<descriptor id= "create-item" type= "unsafe" ref=">

<descriptor href= "#name" />

<descriptor href= "scheduledTime" />

< / descriptor>

<descriptor id= "edit" type= "idempotent" ref=">

<descriptor href= "#identifier" />

<descriptor href= "#status" />

< / descriptor>

</alps>

You may have noticed that the document looks like a General dictionary of all possible values for fields and actions from the to list service interface – and this is the essence of the idea. Services that agree to adhere to this profile can make their own decisions about the Protocol, message format, and even the URL. Clients who agree to accept this profile will be created in such a way as to understand and, if possible, activate the descriptors.

It's also a great format for:

generate documentation in an easy-to-read format for people,

analysis of similar formats,

tracking which profiles are most commonly used,

even to generate a state diagram.

But this is a topic for another article.

Now that you have a complete list of descriptors with consistent names, brief notes for the state diagram, and a semantic profile, you're ready to start writing code for the service and the client.

Step 6: Write some code

From this point on, you can pass your developments (state diagram and semantic profile) to the server and client developers to execute a specific implementation.

The HTTP server must have access to a diagram of the second step and the query client needs to start the state transitions of the service. Each view submitted from the service must be in the format selected in step 3 and must include a link to the profile created in step 4. Responses should include appropriate hypermedia controls that implement actions from the state diagram and are described in the document profile. Client and server side developers can write relatively independent code from now on, but don't forget to run tests to see if the state diagram and profile match.

Once you've written and stabilized your code, the last step in the list remains: Publish.

Step 7: Publish your API

The Web API must publish at least one URL that will always respond to customer requests – even in the distant future. I call it the" billboard URL " –the ONE everyone knows. It is also a good idea to publish a profile document so that new implementations of the service can reference it in the responses. You can also publish at this address readable documentation, tutorials and other information that can help developers understand and use your service.

In the end, you should have a well-designed, stable, affordable and working service that is ready to use.

In conclusion

This article covers a set of steps to build an API for your network. The main emphasis is on getting the correct data descriptors and actions. Documenting them for machine reading to make it easier for people to write client and server parts even if they don't communicate directly.

7 steps again:

List all parts

Specify all types of data that the client needs to communicate with the service.

Draw a state diagram

Document all actions (state transitions) that are available to the service

Agree on magic variables

Bring the names in your public interface to accepted standards

Select the type of hypermedia

Select the message format that most fully displays transitions in the service based on the selected Protocol.

Creating a semantic profile

Write a document that will contain and define all the descriptors used in the service

Write the implementation

Distribute the semantic profile and state diagram to server and client developers so that they can write code as recommended and edit the profile/diagram as needed.

Publish your API

Publish the "billboard URL" and semantic profile so others can use it to create new services and/or client applications.

You will likely need to go back to some of the steps and go through them again to fill in the missing data or to display the trade-offs found in the design process. The sooner this happens, the better. It is also possible that you will be able to use these API developments at some point in the future to create new implementations with new formats and protocols that developers will need.

After all, this methodology is just one way to create a reliable, repeatable, consistent, and seamless Web API design process. In the process of considering this example, you may decide that in your case some steps need to be added, some shortened and, of course, the decisions on the choice of data exchange format and Protocol can vary greatly from project to project.

I hope this article has given you food for thought on how to build an optimal methodology to build an API in your organization or team.

**3**

**Practical task№ 1.An Example web site to ASP.NET WebForms**

While many are beginning to use ASP.NET in their projects, and Microsoft is releasing new frameworks, more and more often the question arises of which model to use - "classic ASP.NET" (called WebForms), that is, use the model offered by default or newly created ASP.NET MVC. Also, many people think that Microsoft is making a mess, saying that they will fully support and develop both models. And since every project can be done using WebForms and MVC, the question often arises — "what is better for my project?"One of the good ways to answer this question is to conduct a SWOT analysis of both approaches.

Strengths (Strong sides)

Offers full control over the generated HTML code;

Generates pure HTML;

Better separation between UI and code (application logic and presentation logic);

Easier for unit testing;

Supports many different view engines (View Engines);

By default uses restful approach to URLS — which is also good for SEO;

No ViewState (this can also be a disadvantage in certain cases);

The normal size of the loaded page is small;

Easy integration with jQuery-type frameworks.

Weaknesses (Weaknesses)

Not based on (server-side) events, so it can be difficult to master for those who think in terms of model ASP.NET WebForms;

The number of third-party control libraries and components is small;

No ViewState (also an advantage).

Opportunities (Features)

Allows you to use the approach of Test Driven Development (TDD) — the framework is built for the most part with an emphasis on TDD, so that MVC is much easier to write unit tests, create mock objects and intercept the execution of the application, controlling it at all stages;

Allows you to reuse the application model with other UI interfaces and applications.

Threats (Threats)

It requires more time to start productive work with it and more time to learn by beginners in web development.

Strengths (Strong sides)

Provides excellent opportunities for RAD( Rapid Application Development, rapid application development);

Excellent support for" designer " in Visual Studio;

Easy to develop business applications that work with large amounts of data and are data-driven (data-heavy);

A huge number of third-party companies that provide ready-made library controls, as well as supporting projects on WebForms;

Familiar to Windows Forms developers concept of events (server events for UI) that allows you to quickly start working with this approach.

Weaknesses (Weaknesses)

UI logic is closely related to code and thus difficult to separate;

It is difficult to perform unit-testing, so it is difficult to use the TDD approach;

A large (and often huge) page size due to ViewState.

Opportunities (Features)

Great for quickly prototyping business applications. This works great when you need to agree with a potential customer a preliminary concept.

Threats (Threats)

It is difficult to apply different UI to the same application, even though there are enough frameworks and technologies available (Master pages, themes, etc.).

Based on all these arguments, it is possible to present the decision-making process in favor of an approach in this way:

Scheme decision-making approach ASP.NET

Summarizing this scheme, we can draw two key conclusions that determine the choice of approach:

Test Driven Development (development with tests) – life will be much easier if you use MVC when using TDD practice.

Data Driven Application-life will be MUCH easier if you use WebForms while the application works with a huge variety of data.

The website is based on Web Forms software technology ASP.NET, C # programming language. Website design based on open-source templates. The site consists of several pages for greater clarity. All web pages are located in the root directory.

To load web pages, natural addresses are used: both the page is called physically, and it is indicated in the address bar of the browser. The advantage of this approach is the simplicity and accuracy of the file names of web pages, disadvantages-it is difficult to maintain a site with a large number of pages.

The content of the site's web pages is stored in the database that the file plays .xml. By changing the text content of the database file, you can change the text that is displayed on site pages. A speciccal class unifies the loading of content for each web page separately. Instead of a file .xml can be easily used any of the binary database.

The website is built using master pages and subsidiary pages with content. This is done to ensure the uniformity of the appearance of the website pages. The master page contains the main design, in the subsidiary pages only the content necessary for display. When loading, the master page is automatically merged with the requested subsidiary page.

The master page is essentially an ordinary ASPX page, but with a special extension .master. Just like any aspx page, it consists of two pages, one with html code (plus embed code), the other with separated code in .NET languages: C# or Visual Basic.

Listing № 1 view of the master page

<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site1.master.cs" Inherits="WebApp1.Site1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>An example of a site on ASP.NET. Web Forms </ title>

<asp:ContentPlaceHolder ID="head" runat="server">

<% /\* Here you can find meta tags used to store information

designed for browsers and search engines. Each subsidiary page can

post your personal information between tags with ID

ID="head" \*/

%>

</asp:ContentPlaceHolder>

</head>

<body>

<form id="form1" runat="server">

<div>

In contrast to the master page, the subsidiary page is greatly simplified, allowing you to easily work with the content of only this web page of the site. In its pure form, the subsidiary page has very little service code.

Listing No. 2 subsidiary page view

<%@ Page Title="Преимущества технологии ASP.NET" Language="C#"

MasterPageFile="~/Site1.Master" AutoEventWireup="true" CodeBehind="gallery.aspx.cs" Inherits="WebApp1.gallery" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">

<% /\* Location of headers, metadata, scripts, styles, etc. \*/ %>

</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="Content" runat="server">

<% /\* Here should be the main content of the web page \*/ %>

</asp:Content>

The site was created in Visual Studio 2013. You can open and edit the website source in Visual Studio 2010, Visual Studio 2012, Visual Studio 2013, any version, and in SharpDevelop 5.x.

**Practical task№ 2 Example of a website on the Razor engine**

The sample website is based on pages with an efficient Razor engine. You can create a website on the Razor engine quickly and comfortably. A website consists of physical pages that are manually created and have an extension .cshtml. To simplify the source code of a website, dynamic behavior is used only to merge a single layout page with multiple content pages. In the source code of the website on the Razor engine there is an example of the implementation of the C# language code in a web page with html content. The program code itself is located in a separate folder for better structuring of the site.

To maintain a consistent design style and speed up the development process, the site structure is based on layout pages and content pages. The layout page contains the overall design and content for all pages of the site. The final result is achieved by merging the layout and content, as a result, a full html code of the synthesized web page is sent to the browser. I must say that the ability to automatically merge the source code of several pages is not available technology on pure html, but only web technologies with embedded code: ASP.NET, PHP, Python, etc.

A layout page is typically a shared content placement field for subsidiary content pages. The layout pages are level and can be nested. Belonging of a web page to the layout is conditional and is determined only by the presence of calls to some methods. So on the page layout should be:

call @RenderBody (...),

and not necessarily one or more sections @RenderSection (...).

Listing №1 layout Page on the engine Razor

@\*

As you can see from the file extension of the level page \_Layout.cshtml

it is no different from other pages .html. Belonging to

the layout page is determined only by the presence of a mandatory query on it

the body of the page content:

@RenderBody(),

on the layout page, you can declare one or more sections

embed code from content pages:

@RenderSection([название секции], required: [mandatory section on the content page])

\*@

\*@

@Page.Title - dynamically generated variable

where the subsidiary pages of content

the content of the Title of the web page will be determined

\*@

<!DOCTYPE html>

<html>

<head>

<title>\*@

Page.Title</title>

<meta charset="utf-8" />

@\*

The announcement section for

keywords and meta-descriptions,

"required: false" - determination of non-requirement of presence of this section on the pages content.

You can create as many sections as you want on a level page,

for greater dynamism and interactivity of the site.

\*@

@RenderSection("meta", required: false)

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

<link href="css/main.css" rel="stylesheet" type="text/css" />

<script src="js/main.js" type="text/javascript"></script>

</head>

<body>

@\*

Space for the overall design of the top of the web pages: header,

menu, advertising content, etc.

\*@

@\* Space to insert the content of the page content\*@

@RenderBody()

@\*

Space for the overall design of the bottom of web pages:

additional menu, basement, advertising content, etc.

\*@

</body>

</html>

Page content on the Razor engine must have the addresses of the layout between the symbols @{...}, for example:

@{Layout = " ~/\_LayoutMain.cshtml";}

And if the layout was declared mandatory section:

@RenderSection ("meta", required: true), the subsidiary pages must also strictly define the specified sections. If the presence of sections is not necessary:

@RenderSection ("meta", required: false), you can ignore the definition. A subsidiary page, in turn, can itself be a layout for the other pages of the contents when placed in the body of the code @RenderBody() and the possibility of declaring sections of the @RenderSection().

Listing No. 2 subsidiary content page on the Razor engine

@\*

The content page must have the address

layout pages between @{characters...} :

Layout = " [layout page name].cshtml.

In addition, if the layout declared mandatory

sections (for example, @RenderSection ("meta", required: true)),

the page content must be

definition of this section.

The body of the page content,

inserted into the layout in place of the web site originator created in MS Visual Studio 2013 and can be opened and edited in Visual Studio 2012, 2013 of any versions, WebMatrix 3 and above, SharpDevelop 4 and above

**Practical task№ 3. Writing html code for the page ASP.NET WebForms**

Web forms ASP.NET allow you to create dynamic websites by adding drag-and-drop controls and an event-based programming model that is familiar to all windows programmers . Hundreds of controls and components allow you to quickly create complex sites with an interactive interface and access to databases. Web Forms ASP.NET simplify the process of creating interactive web pages provide a rich user interface and contribute to the rapid adaptation of novice programmers.

The name Web Forms is associated with Windows Forms for a reason. There are also forms, and these are user interface (UI) elements, or server controls, called web forms. The web forms code is processed by the server running the .NET Framework, and when the web page is loaded into the browser, the server controls are converted to the appropriate html tags. Web forms form the face of the page.

The main tool for creating and editing web form pages ASP.NET is Microsoft Visual Studio. When working in Visual Studio, web controls can be added to a page manually or by dragging and dropping from the Toolbar. An html form tag is automatically added to the web page, usually in a single instance, allowing users of the site to interact with the web page interface. Working with WebForms, you can create html markup for controls in a variety of ways. All methods give visually the same, but have their own characteristics.

Classic, html method of creating elements and processing POST request. In this case, you can simply manually add html tags and form a response to the POST request before reloading the web page in the Load event. The advantage of this method is the possibility of full independent processing of requests. In html elements of the form must put down the attribute name, because it builds a mechanism for decrypting requests based on the key-value. Programmatic communication is supported through properties and methods that return a string type. It should be added that with this method, all responsibility for the security of the site falls on you and you will also check the input data. Listing 1 using html tags and embedded code

<form id="form1" runat="server">

<div class="content">

<h4 style="text-align: center; color: #ff0000;">Тригонометрический калькулятор</h4>

Градусы

<input name="Degree" type="text" value="<% =degree %>" style="width: 180px;" />

Косинус: <span style="color:blue;"><% =cosinus %></span>

<br /><br />

Синус: <span style="color:blue;"><% =sinus %></span>

<input name="Calculate" type="submit" style="width: 180px;" value="Вычисление" /><br />

</div>

</form>

. . . . . . . . . . . . . . . . . . . . .

public string degree = null;

public string cosinus = null;

public string sinus = null;

protected void Page\_Load(object sender, EventArgs e)

{

if(IsPostBack == true)

{

if (Request.Params["Calculate"] != null)

{

double radian = Math.PI \* double.Parse(Request.Params["Degree"]) / 180.0;

cosinus = Math.Cos(radian).ToString();

sinus = Math.Sin(radian).ToString();

degree = Request.Params["Degree"];

}

}

}

The second method is similar to the first one with one small but magic code detail: the runat="server"attribute. Now we trust the html elements of the form to be processed by the server and the .NET Framework, for this purpose it is necessary to assign an identifier to the element together with the runat="server" attribute. Now, it seems to be purely html elements are transformed into objects of classes of the namespace System.Web.UI.HtmlControls, becoming HTML server controls. You can now programmatically manipulate html elements and access their properties through the value of the id attribute. In the source code of the web page, the name attribute is borrowed from the id attribute. Note that the name attribute of The element with the length identifier is different from the id, but after loading the page in the browser name will be equal to the ID: this server tried. Listing # 2 Processing html tags server

<form id="form1" runat="server">

<div class="content">

<h4 style = "text-align: center; color: # ff0000;"> Calculate the area of a rectangle </ h4>

<% // For html tags processed by the server the id attribute is important, the name is inserted automatically equal to id%>

Length

. . . . . . . . . . . . . . . . . . . . .

protected void Page\_Load(object sender, EventArgs e)

{

if(IsPostBack == true)

{

if (Calculate.Value != null)

{

double width = double.Parse(Width.Value);

double length = double.Parse(Length.Value);

Area.InnerText = (width \* length).ToString();

}

}

}

Of course two ways to create html code page Web Forms described above will be more like those in html as a fish in the water. But these methods are easy to write only small in size interfaces of web pages and tedious to write complex, multi-part html groups. For such purposes, and serves as a third method, justifying the characteristic ASP.NET WebForms as a technology for rapid development of web applications with writing a small amount of code.

The third method and the main ASP.NET WebForms is the use of built-in controls. Hundreds of built-in elements and the ability to create their own greatly increases the speed of website development. Listing No. 3 in the Built-in controls with events

<form id="form1" runat="server">

<div class="content">

<h4 style="text-align: center; color: #ff0000;"

runat="server" id="HeaderLogin"> Set the current date and sign in </h4>

<asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>

<asp:Login ID="Login1" runat="server" BackColor="#EFF3FB" BorderColor="#B5C7DE"

BorderPadding="4" BorderStyle="Solid" BorderWidth="1px" OnLoggingIn="Login1\_LoggingIn" >

</asp:Login>

<span id="Help" runat="server"> Login: planet, password: earth </span>

</div>

</form>

. . . . . . . . . . . . . . . . . . . . .

<form id="form1" runat="server">

<div class="content">

<h4 style="text-align: center; color: #ff0000;"

runat="server" id="HeaderLogin"> Set the current date and sign in </h4>

<asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>

<asp:Login ID="Login1" runat="server" BackColor="#EFF3FB" BorderColor="#B5C7DE"

BorderPadding="4" BorderStyle="Solid" BorderWidth="1px"

OnLoggingIn="Login1\_LoggingIn" OnAuthenticate="Login1\_Authenticate" >

</asp:Login>

<span id="Help" runat="server"> Login: planet, password: earth </span>

</div>

</form>

The appearance of element code is somewhat unusual for web programmers and initially slows down the speed of development, but thanks to the wonderful IntelliSense technology, the development of built-in elements moves in leaps and bounds.

This method is friendly to windows programmers, they can easily and quickly switch to the web platform without spending a lot of time learning html tags. In this case, you only form the interface, and the web Forms framework turns user actions into relevant events. Web Forms provides the ability to create multiple tags to build a complex interface with a single class object, practically creating new html elements. In addition to html code, web components can create the necessary javascript code. For frequent use of html groups, you can create your own components.

In Web Forms, usually pure html tags are used only for decorative elements of a web page, interactive controls are the same, for security purposes, it is better to trust the server and the .NET Framework framework. Using built-in or self-created web elements can significantly improve the speed of development. It is for the rapid creation of websites and web Forms framework is designed.

All that is not expressed in words can be seen clearly in the source code. You can test the source code of web form pages in WebMatrix or in MS Visual Studio 2012 or later. In the two source code: web application and web site. The web application will open in Visual Studio 2012 and above, the website in Visual Studio 2008 and in Web Matrix 3 and above.

**Practical task№ 4. HTML form element helpers**

To create forms, we can use standard html elements, for example:

|  |  |
| --- | --- |
|  | <form method="post" action="/Home/Create">      <p>          <label>Модель</label><br />          <input type="text" name="Name" />      </p>      <p>          <label>Цена</label><br />          <input type="number" name="Price" />      </p>      <p>          <input type="submit" value="send" />      </p>  </form> |

This is a common html-form that sends all the data entered by a POST request to the address /Home/Create by pressing the button. The built-in BeginForm/EndForm helper allows you to create the same shape:

|  |  |
| --- | --- |
|  | @using(Html.BeginForm("Create", "Home", FormMethod.Post))  {      <p>          <label>Модель</label><br />          <input type="text" name="Name" />      </p>      <p>          <label>Цена</label><br />          <input type="number" name="Price" />      </p>      <p>          <input type="submit" value="send" />      </p>  } |

The BeginForm method takes the action method name, controller name, and query type as parameters. This helper creates both an opening <form> tag and a closing </form>tag. Therefore, when rendering a view to the output stream, we get the same html code as using the form tag. Therefore, both methods are identical.

There is one point here. If we have two versions of the same method defined in the controller - for POST and GET methods, for example:

|  |  |
| --- | --- |
|  | [HttpGet]  public IActionResult Create()  {      return View();  }    [HttpPost]  public IActionResult Create(Phone myPhone)  {      //..............  } |

That is, in fact, the page with the form is called and the form is sent by the same action . In this case, you can omit the Helper html.BeginForm parameters:

|  |  |
| --- | --- |
|  | @using(Html.BeginForm())  {      .............  } |

Information input

In the previous example, together with the Helper html.BeginForm used standard html elements. However, the html helper set also contains helpers for user input. MVC defines a wide range of input helpers for virtually every html element. What to choose - a helper or standard html input elements, is already decided by the developer.

Regardless of the type, all basic html helpers use at least two parameters: the first parameter is used to set values for the ID and name attributes, and the second parameter is used to set the value of the value attribute

#### Html.TextBox

Html Helper.TextBox generates an input tag with the value of the type attribute set to text. TextBox helper is used to get user input information. So, rewrite the previous form with the replacement of input fields on the Html helper.TextBox:

|  |  |
| --- | --- |
|  | @using (Html.BeginForm("Create", "Home", FormMethod.Post))  {      <p>          <label>Модель</label><br />          @Html.TextBox("Name")      </p>      <p>          <label>Цена</label><br />          @Html.TextBox("Price","", new { type="number" })      </p>      <p>          <input type="submit" value="Отправить" />      </p>  } |

The helper allows you to set a number of additional parameters using overloaded versions. So, call helper:

|  |  |
| --- | --- |
| 1 | @Html.TextBox("Price","", new { type="number" }) |

The second parameter is set to the default value (here is an empty string). The third parameter in the form of an anonymous object allows you to set a number of attributes of the generated html element. So, in this case, we specify that the field will be numeric, because by default the created field is regarded as text, that is, with the attribute type="text".

As a result, we get almost the same result:

#### 

#### Html.Label

Helper Html.Label creates a <label/> element, and the parameter passed to the helper determines the value of the for attribute and the text on the element at the same time. The overloaded version of the helper allows you to define the value of the for attribute and the text on the label independently. For example, an HTML.Label helper declaration ("Name"," Model") creates the following markup:

One

<label for= "Name" > Model< / label>

The label is a simple label designed for attaching information to elements of the input, for example, to text fields. The for attribute of the label element must contain the id of the associated input element. If the user clicks on a label, the browser automatically passes focus to the input element associated with the label.

Html.TextArea

The TextArea helper is used to create a <textarea> element that represents a multiline text field. The result of the @Html expression.TextArea ("text", " Hello world")

the html markup will be as follows:

|  |  |
| --- | --- |
| 1  2 | <textarea cols="20" id="text" name="text" rows="2"> Hello world  </textarea> |

Note that the helper decodes the value to be placed in it, including html tags (all helpers decode model values and attribute values). Other versions of the TextArea helper allow you to specify the number of rows and columns that define the size of the text box.

|  |  |
| --- | --- |
| 1 | @Html.TextArea("text", " Hello world ", 5, 50, null) |

This helper will generate the following markup:

|  |  |
| --- | --- |
| 1 | <textarea cols="50" id="text" name="text" rows="5"> Hello world </textarea> |

#### Html.Hidden

Хелпер Html.Hidden generates a hidden field. For example, the following helper call:

@Html.Hidden("PhoneId", "2")

generate markup:

|  |  |
| --- | --- |
| 1 | <input id="PhoneId" name="PhoneId" type="hidden" value="2" /> |

#### Html.Password

Html.Password creates a password field. It is similar to the TextBox helper, but displays the password mask instead of the entered characters. Following code:

@Html.Password("UserPassword", "val")

|  |  |
| --- | --- |
| 1 | <input id="UserPassword" name="UserPassword" type="password" value="val" /> |

#### Html.RadioButton

Html helper is used to create switches.RadioButton. It generates an input element with a value of type= "radio". To create a group of switches, you must assign the same name to all of them (the name property):

|  |  |
| --- | --- |
|  |  |

This code creates the following markup:

|  |  |
| --- | --- |
|  | <input id="color" name="color" type="radio" value="red" />  <span>красный</span> <br />  <input id="color" name="color" type="radio" value="blue" />  <span>синий</span> <br />  <input checked="checked" id="color" name="color" type="radio" value="green" />  <span>зеленый</span> |

**Html.CheckBox**

Html.CheckBox is used to create a checkbox or checkbox element. For example, the following code:

|  |  |
| --- | --- |
| 1 | @Html.CheckBox("Enable", false) |

будет генерировать следующий HTML:

|  |  |
| --- | --- |
| 1  2 | <input id="Enable" name="Enable" type="checkbox" value="true">  <input name="Enable" type="hidden" value="false"> |

In fact, two elements are created-the actual check box and the hidden field, which is usually placed at the end of the form and is used to track changes in the value of the check box.

Html.DropDownList

htmlHelper.DropDownList creates a drop-down list, that is, the <select />element. To generate such a list, you need a collection of SelectListItem objects that represent the list items. You can create a collection of SelectListItem objects or use the SelectList helper. This helper scans IEnumerable objects and converts them into a sequence of SelectListItem objects. So, the following code:

|  |  |
| --- | --- |
| 1 | @Html.DropDownList("phone", new SelectList(new string[] { "iPhone 7 Pro", "Galaxy 7 Edge", "HTC 10", "Honor 5X" }), "Choose a model") |

generates the following markup:

|  |  |
| --- | --- |
|  | <select id="phone" name="phone"><option value="">Choose a model</option>  <option>iPhone 7 Pro</option>  <option>Galaxy 7 Edge</option>  <option>HTC 10</option>  <option>Honor 5X</option>  </select> |

The SelectListItem object has the Text (display text), Value (value itself, which may not be the same as the text), and Selected properties. Now a more complex example. Let's list the collection of Phone elements:

|  |  |
| --- | --- |
|  | public class Phone  {      public int Id { get; set; }      public string Name { get; set; }      public int Price { get; set; }  } |

In the controller, pass the list of Phone objects through the ViewBag:

|  |  |
| --- | --- |
|  | public IActionResult Create()  {      List<Phone> phones = new List<Phone>      {          new Phone {Id=1, Name="iPhone 7 Pro", Price=680 },          new Phone {Id=2, Name="Galaxy 7 Edge", Price=640 },          new Phone {Id=3, Name="HTC 10", Price=500 },          new Phone {Id=4, Name="Honor 5X", Price=400 },      };      ViewBag.Phones = new SelectList(phones, "Id", "Name");      return View();  } |

Here we create a SelectList object by passing a set of values (phones list) to its constructor, the name of the Phone model property to be used as the value (Id), and the name of the Phone model property to be used for display in the list. In this case, it is not necessary to install two different properties, it was possible to install and one for the value and display.

Then in the view we can use this SelectList:

One

@Html.DropDownList ("phoneid", ViewBag.Phones as SelectList)

And when rendering the view, all SelectList elements will be added to the drop-down list

Html.ListBox

Html Helper.ListBox, as well as a DropDownList, creates a <select />, but it makes it possible for multiple selection of items (i.e., multiple attribute set to multiple). To create a list that supports multiple selection, you can use the MultiSelectList class instead of the SelectList:

One

@Html.ListBox ("phones", new MultiSelectList (new string [] {"iPhone 7 Pro", "Galaxy 7 Edge", "HTC 10", " Honor 5X" }))

This code generates the following markup:

|  |  |
| --- | --- |
|  | <select id="phones" multiple="multiple" name="phones">  <option>iPhone 7 Pro</option>  <option>Galaxy 7 Edge</option>  <option>HTC 10</option>  <option>Honor 5X</option>  </select> |

#### Html.GetEnumSelectList

Html helper is used to create a drop-down list by enumeration.GetEnumSelectList. Let's say we have the following enumeration:

|  |  |
| --- | --- |
|  | using System.ComponentModel.DataAnnotations;    namespace HtmlHelpersApp.App\_Code  {      public enum TimeOfDay      {          [Display(Name ="Morning")]          Morning,          [Display(Name = "Day")]          Afternoon,          [Display(Name = "Evening")]          Evening,          [Display(Name = "Night")]          Night      }  } |

Then, in the view, we can display a drop-down list of values from this enumeration:

|  |  |
| --- | --- |
|  | @using HtmlHelpersApp.App\_Code    @using (Html.BeginForm("Create", "Home", FormMethod.Post))  {      @Html.DropDownList("daytime", Html.GetEnumSelectList(typeof(TimeOfDay)))      <br />      <p>          <input type="submit" value="Send" />      </p>  } |

## Practical task№ 5 creating HTML-helpers

To create a simple html-helper take the project ASP.NET Core 2.0 by type Web Application (Model-View-Controller) and add the App\_Code folder to it. Then add a new class ListHelper to this folder:

|  |  |
| --- | --- |
|  | using Microsoft.AspNetCore.Html;  using Microsoft.AspNetCore.Mvc.Rendering;    namespace HtmlHelpersApp.App\_Code  {      public static class ListHelper      {          public static HtmlString CreateList(this IHtmlHelper html, string[] items)          {              string result = "<ul>";              foreach (string item in items)              {                  result += $"<li>{item}</li>";              }              result += "</ul>";              return new HtmlString(result);          }      }  } |

The new helper class defines one static CreateList method that takes the object for which the method is created as the first parameter. Because this method extends the functionality of html helpers, which represents Microsoft.AspNetCore.Mvc.Rendering.IHtmlHelper interface, the object of this type is passed in this case as the first parameter. The second parameter of the CreateList method is an array of string values, which will then be displayed in the list.

In the method itself, just run through the array of strings and form html markup from them in the form of a string. The result of the method is an HtmlString object that receives html markup as a string in the constructor.

This very simple method can already make it easier to work with markup. Consider its use. Let's say we need to display an array of strings in the list in the view:

|  |  |
| --- | --- |
|  | @{      ViewData["Title"] = "Home Page";  }    @{      string[] cities = new string[] { "London", "Paris", " Berlin" };      string[] countries = new string[] { "Great Britain", "France", " Germany" };  }  @using HtmlHelpersApp.App\_Code    <h3>Cities</h3>  @Html.CreateList(cities)  <br />  <h3>Countries</h3>  <!-- or can be trigger so -->  @ListHelper.CreateList(Html, countries) |

Since the html helper is an extension method for the IHtmlhelper object, we only need to write Html.CreateList to use it and pass the required parameters to the method. Or we can call it as a method of the class in which it is defined: ListHelper.CreateList

And now, if we want to create a <ul> list, we just need to write one line with a helper call, passing it an array:

In the absence of such a helper, we would have to essentially duplicate the same html-code to create a list. However, this helper is still quite simple, and if we have to create a hundred times more complex, but the same type of HTML markup, then helpers will be even more useful.

### TagBuilder

To create html tags in the helper, we can use the class Microsoft.AspNetCore.Mvc.Rendering.TagBuilder. So, let's rewrite the helper code as follows:

|  |  |
| --- | --- |
|  | using Microsoft.AspNetCore.Html;  using Microsoft.AspNetCore.Mvc.Rendering;  using System.Text.Encodings.Web;    namespace HtmlHelpersApp.App\_Code  {      public static class ListHelper      {          public static HtmlString CreateList(this IHtmlHelper html, string[] items)          {              TagBuilder ul = new TagBuilder("ul");              foreach (string item in items)              {                  ….              }              ul.Attributes.Add("class", "itemsList");              var writer = new System.IO.StringWriter();              ul.WriteTo(writer, HtmlEncoder.Default);              return new HtmlString(writer.ToString());          }      }  } |

The Tagbuilder constructor passes the element for which the tag is created. TagBuilder has a number of properties and methods that you can use:

* The InnerHtml property allows to set or get the tag content as a string. To manipulate this property, you can call one of the methods:
  + Append (string text): add a test string to an element
  + Append Html (IHtmlContent html): adding HTML code to an element as an IHtmlcontent object-this can be another TagBuilder object
  + Clear (): clears the element
  + SetContent(string text): sets the text of an element
  + Set the Html Content(IHtmlContent html): set the inner html object IHtmlContent
* The Attributes property allows you to control the attributes of an element
* The MergeAttribute () method allows to add one attribute to the element
* The AddCssClass () method allows you to add a css class to an element
* The WriteTo() method allows you to create a string from an element and its internal content using TextWriter and HtmlEncoder objects.

As a result, we get the same list as before.