

Copyright © 2014 by Software Craftsmanship Guild.

All rights reserved. No part of these materials may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the Software Craftsmanship Guild. For permission requests, write to the Software Craftsmanship Guild, addressed “Attention: Permissions Coordinator,” at the address below.

Software Craftsmanship Guild

526 S. Main St, Suite 609

Akron, OH 44311



# ASP.NET WebAPI

Software Craftsmanship Guild

# Why WebAPI?

HTTP is not just for serving web pages, it can also be used to expose services and data.

Because HTTP is supported in just about every language, browser, mobile device, etc. when you build services with Web API just about anyone can connect to and use your services.

WebAPI allows .NET Developers to write controller methods in their .NET language of choice, and automatically serializes results into JSON, XML, or basically whatever the client wants.

# Let's Create a Project


Empty template, check the MVC and WebAPI boxes.


Add bootstrap as well so the pages look nice and let's copy in our Contact and FakeContactDatabase from Lecture 02 – First MVC App. Add a static constructor to put some default data in it though.


```
static FakeContactDatabase()
{
    _contacts.AddRange(new[]
    {
        new Contact() {ContactId = 1, Name = "Jenny", PhoneNumber = "867-5309"},
        new Contact() {ContactId = 2, Name = "Bob", PhoneNumber = "555-1212"}
    });
}
```


New ASP.NET Project - WebAPISample


Select a template:


  
Empty

  
Web Forms

  
MVC

  
Web API

  
Single Page Application

  
Facebook

Add folders and core references for:

☐ Web Forms ☒ MVC ☒ Web API

☐ Add unit tests

Test project name:

An empty project template for creating ASP.NET applications. This template does not have any content in it.

[Learn more](#)

Change Authentication

Authentication: **No Authentication**

OK Cancel

# Add the index + layout page

(don't forget to update the \_Layout jQuery version)

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

namespace WebAPISample.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

**Add View**

View name:

Template:

Model class:

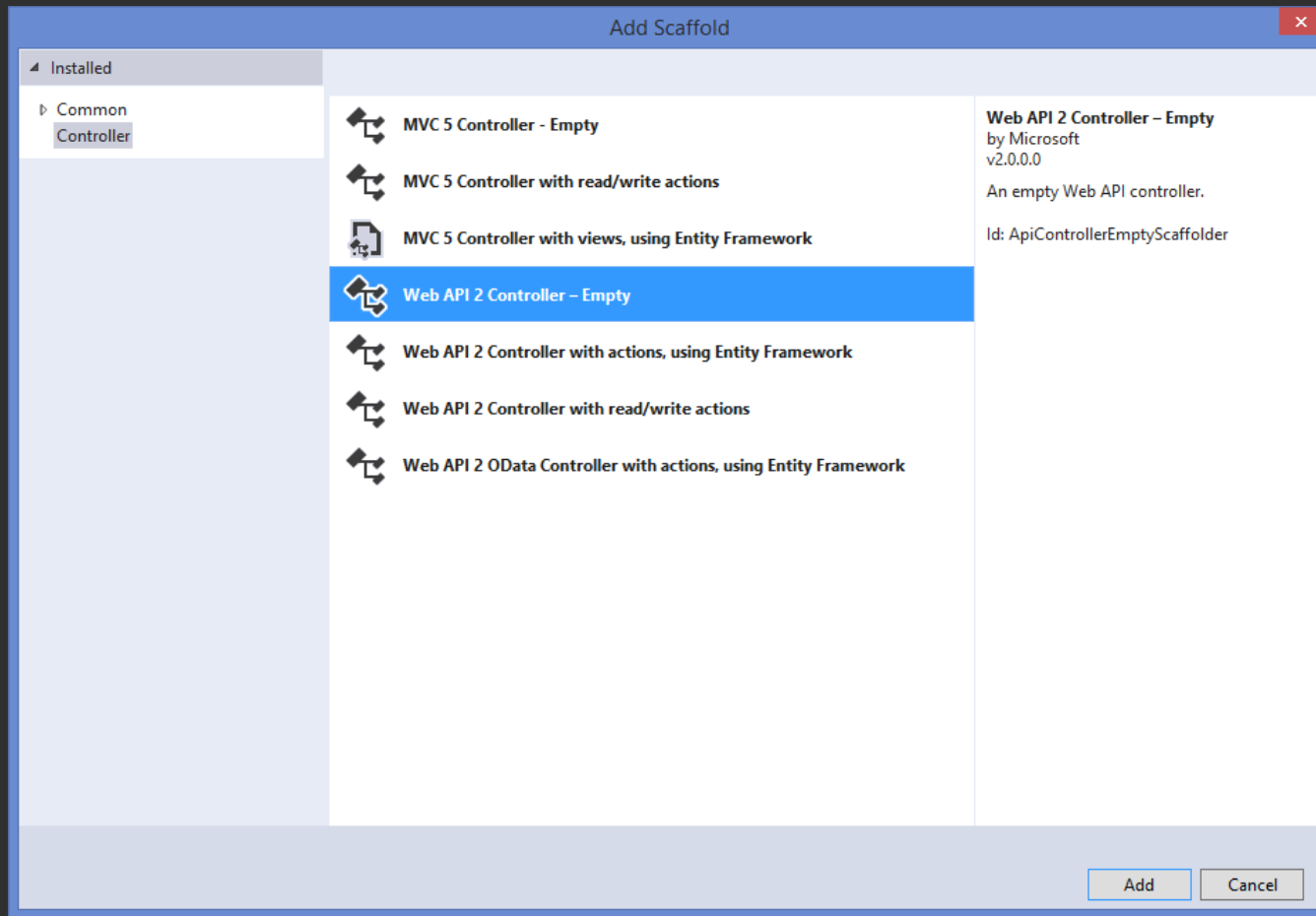
View options:

- ☐ Create as a partial view
- ☒ Reference script libraries
- ☒ Use a layout page:

...

(Leave empty if it is set in a Razor \_viewstart file)

# Now let's add an empty Web API2 Controller - ContactsController



# Routing with WebAPI

Notice in the App\_Start folder there is a new config file: WebApiConfig.cs.

This file serves the same purpose as the RouteConfig for MVC controllers. Also note that the default template has a hard coded api/ segment.

```
config.Routes.MapHttpRoute(  
    name: "DefaultApi",  
    routeTemplate: "api/{controller}/{id}",  
    defaults: new { id = RouteParameter.Optional }  
);
```



# Now add some controller methods

WebAPI automatically converts C# objects to a data format requested (XML or JSON). So writing a Web API method is like writing any other method... but we like to put the HTTP verb as the method name like so:

```
public List<Contact> Get()
{
    var repository = new FakeContactDatabase();
    return repository.GetAll();
}

public Contact Get(int id)
{
    var repository = new FakeContactDatabase();
    return repository.GetById(id);
}
```

This gives us the following URL Routes:

1. GET /api/contacts
2. GET /api/contacts/id

# Run the project

Go directly to the /api/contacts url using the address bar, see what happens.

Chrome requests XML, so it gets this:

```
▼<Contact>  
  <ContactId>1</ContactId>  
  <Name>Jenny</Name>  
  <PhoneNumber>867-5309</PhoneNumber>  
</Contact>  
▼<Contact>  
  <ContactId>2</ContactId>  
  <Name>Bob</Name>  
  <PhoneNumber>555-1212</PhoneNumber>  
</Contact>  
</ArrayOfContact>
```

# Using jQuery to load data

Let's edit our index page to include some jQuery that will call out to the WebApi to load our contact list. First we set up the HTML

```
<h2>My Contacts</h2>

<div class="row">
  <h2>All Contacts</h2>
  <table id="contacts" class="table table-striped">
    <thead>
      <tr>
        <th>Id</th>
        <th>Name</th>
        <th>Phone</th>
      </tr>
    </thead>
    <tbody>

  </tbody>
  </table>
</div>
```

# Now add the script

\* If necessary, add the @RenderSection("Scripts", false) to the \_Layout page.

```
<script>
    var uri = '/api/contacts/';

    $(document).ready(function () {
        loadContacts();
    });

    function loadContacts() {
        // Send GET request to WebAPI endpoint
        $.getJSON(uri)
            .done(function (data) {
                // clear the table
                $('#contacts tbody tr').remove();

                // On success, 'data' contains a list of contacts.
                $.each(data, function (index, contact) {
                    // Add a table row for the contact.
                    $(createRow(contact)).appendTo($('#contacts tbody'));
                });
            });
    };

    function createRow(contact) {
        return '<tr><td>' + contact.ContactId + '</td><td>' +
            contact.Name + '</td><td>' + contact.PhoneNumber + '</td></tr>';
    }
</script>
```

# Success!

## All Contacts

Id	Name	Phone
1	Jenny	867-5309
2	Bob	555-1212

# \$.getJSON

The jQuery `getJSON` function sends an AJAX request. The response contains JSON objects from the WebAPI and the `.done()` function is called if the request succeeds.

You can add a `.fail()` function if you want to handle errors.

# \$.each

\$.each() takes an array to loop over whose function takes two parameters: the index in the array and the value of the element.

So \$.each(data, function (index, contact) {

“take each element, which we will call contact, in the data array returned from WebApi”

# Looking up a contact

Create a Lookup() Action on the HomeController. Add this to the view:

```
<div class="row">
  <div class="col-xs-6">
    <h2>Search by ID</h2>
    <div class="form-group">
      <label>Contact Id</label>
      <input type="text" id="contactId" class="form-control" />
    </div>
    <button class="btn btn-primary" onclick="find()">Search</button>
  </div>
</div>
<div class="row">
  <div class="col-xs-6">
    <p id="contact" />
  </div>
</div>
```



# Now code the JavaScript

```
@section Scripts
{
    <script>
        var url = '/api/contacts/';

        function find() {
            // get the value from the textbox
            var id = $('#contactId').val();

            // append the id to the url (see WebApiConfig)
            $.getJSON(url + id)
                .done(function (data) {
                    // put the contact information into the paragraph tag
                    $('#contact').text(data.Name + ' - ' + data.PhoneNumber);
                })
                .fail(function (jqXHR, status, err) {
                    // print error if any
                    $('#contact').text('Error: ' + err);
                });
        }
    </script>
}
```



SOFTWARE  
CRAFTSMANSHIP GUILD

# Success!

## Search by ID

Contact Id

Search

Jenny - 867-5309

# How About Adding a Contact?

To add a contact, let's add a post method to our web api controller.

The REST/HTTP convention for a POST is a HTTP Status Code of Created (201) with a location in the header to a url where you can view the details of the new resource.

```
public HttpResponseMessage Post(Contact newContact)
{
    var repository = new FakeContactDatabase();
    repository.Add(newContact);

    var response = Request.CreateResponse(HttpStatusCode.Created, newContact);

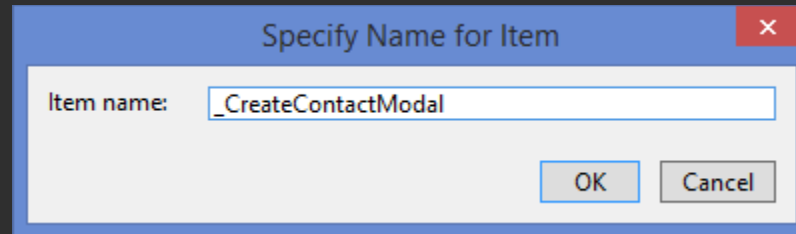
    string uri = Url.Link("DefaultApi", new { id = newContact.ContactId });
    response.Headers.Location = new Uri(uri);

    return response;
}
```

# Creating a Partial View

Just for fun, let's use a bootstrap modal dialogue and put the HTML into a Partial View. Then we will embed it in our index.cshtml page.

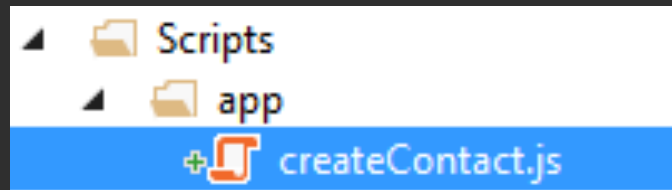
Some developers like to do this to keep things cleaner.



# Create a JavaScript File

Another good practice to get into is not to write your JavaScript in the actual page. In larger apps separating your JavaScript into files is more maintainable.

I prefer to create a folder: Scripts/app to keep my scripts separate from library scripts such as jQuery



# Bootstrap Modals

First we'll need a button that we will wire the click() event to open the modal dialogue (we will name the modal div 'addContactModal'):

```
<button type="button" class="btn btn-primary btn-lg" id="btnShowAddContact">  
  Add Contact  
</button>
```

```
$(document).ready(function () {  
  // on button click, show modal  
  $('#btnShowAddContact').click(function () {  
    $('#addContactModal').modal('show');  
  });  
});
```

# The Modal HTML

\* See the bootstrap documentation for full details

```
<div class="modal fade" id="addContactModal" tabindex="-1" role="dialog"
  aria-labelledby="myModallabel" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
        <h4 class="modal-title" id="myModallabel">Add Contact</h4>
      </div>
      <div class="modal-body">
        <div class="row">
          <div class="col-xs-6">
            <h2>Create Contact</h2>
            <div class="form-group">
              <label>Name</label>
              <input type="text" id="name" class="form-control" />
            </div>
            <div class="form-group">
              <label>Phone</label>
              <input type="text" id="phonenumber" class="form-control" />
            </div>
          </div>
        </div>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-default" data-dismiss="modal">Cancel</button>
        <button type="button" class="btn btn-primary" id="btnSaveContact">Save</button>
      </div>
    </div>
  </div>
</div>
```

# Now finish the JavaScript

We will use the \$.post jQuery command. It needs:

1. The URL
2. A JavaScript object with our data (name the props the same as the C# object)
3. What to do on success/failure

We don't have an edit page, so instead of reading the Header Location we'll just send the browser back to the home index.



# createContact.js

```
$().ready(function () {  
    // on button click, show modal  
    $('#btnShowAddContact').click(function () {  
        $('#addContactModal').modal('show');  
    });  
  
    $('#btnSaveContact').click(function () {  
        var contact = {}; // new object  
  
        // get the values from the inputs  
        contact.Name = $('#name').val();  
        contact.PhoneNumber = $('#phonenumber').val();  
  
        // post it to the WebAPI, passing the JavaScript object  
        $.post(uri, contact)  
            .done(function () {  
                loadContacts();  
                $('#addContactModal').modal('hide');  
            })  
            .fail(function (jqXHR, status, err) {  
                alert(status + ' - ' + err);  
            });  
    });  
});
```



# Success!

Application name    Lookup Contact

**Add Contact**

## All Contacts

Id	Name
1	Jenny
2	Bob

© 2015 - My ASP.NET Application

### Add Contact

### Create Contact

**Name**

**Phone**

# PUT AJAX Requests

For these verbs there is not a shortcut method, instead you use jQuery's \$.ajax function:

```
$.ajax({  
    url: '/api/contacts/',  
    type: 'PUT',  
    dataType: 'json',  
    data: contact,  
    success: function (data, status, xhr) {  
        // do stuff  
    },  
    error: function (xhr, status, err) {  
        alert('error: ' + err);  
    }  
});
```

# DELETE AJAX Requests

Generally, we'll just pass the id of the record we wish to delete as part of the url route, again with \$.ajax:

```
$.ajax({  
    url: '/api/contacts/' + id,  
    type: 'DELETE',  
    success: function (data, status, xhr) {  
        // do stuff  
    },  
    error: function (xhr, status, err) {  
        alert('error: ' + err);  
    }  
});
```

You can also do GET/POST with \$.ajax

But most people prefer \$.post() and \$.getJSON()

...

Under the covers both of these jQuery functions call \$.ajax anyways, so it doesn't really matter which you use.

# Conclusion

Web API is designed to make *serializing* C# objects into JSON or XML automatic.

Methods in Web API resemble normal C# methods, we just have to put HTTP Verbs as the method names. \*

Web API allows us to expose data services to any number of webpage, mobile, or other devices quickly and easily.

\* note that we can be more specific, like `GetContacts()` if we choose