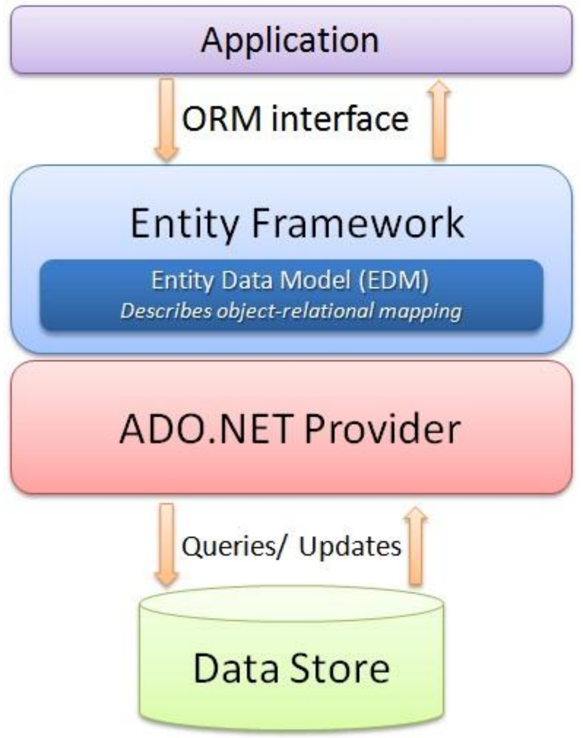
Advanced Programming 2
Recitation 11 – Web Applications Server Side Part II

Roi Yehoshua
2017

Entity Framework

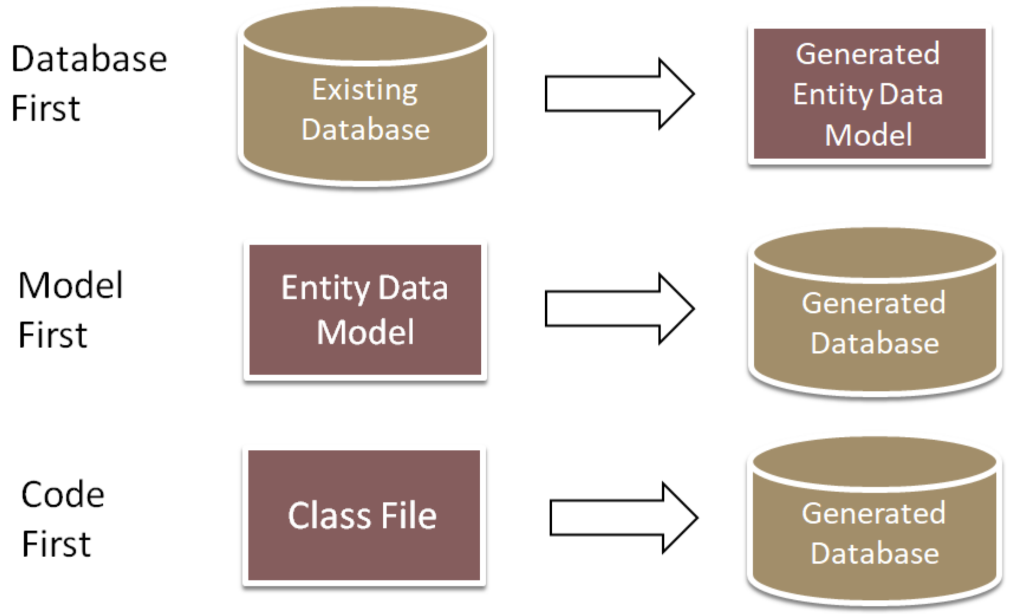
Entity Framework (EF)

- An ORM (Object Relational Mapping) tool
- Enables you to work against a conceptual view of the data
- Generates strongly-typed entity objects
- Generates mapping/plumbing code
- Enables customized mapping scenarios, beyond one-to-one
- Translates LINQ queries to database queries
- Materializes objects from data store calls
- Automatic change tracking



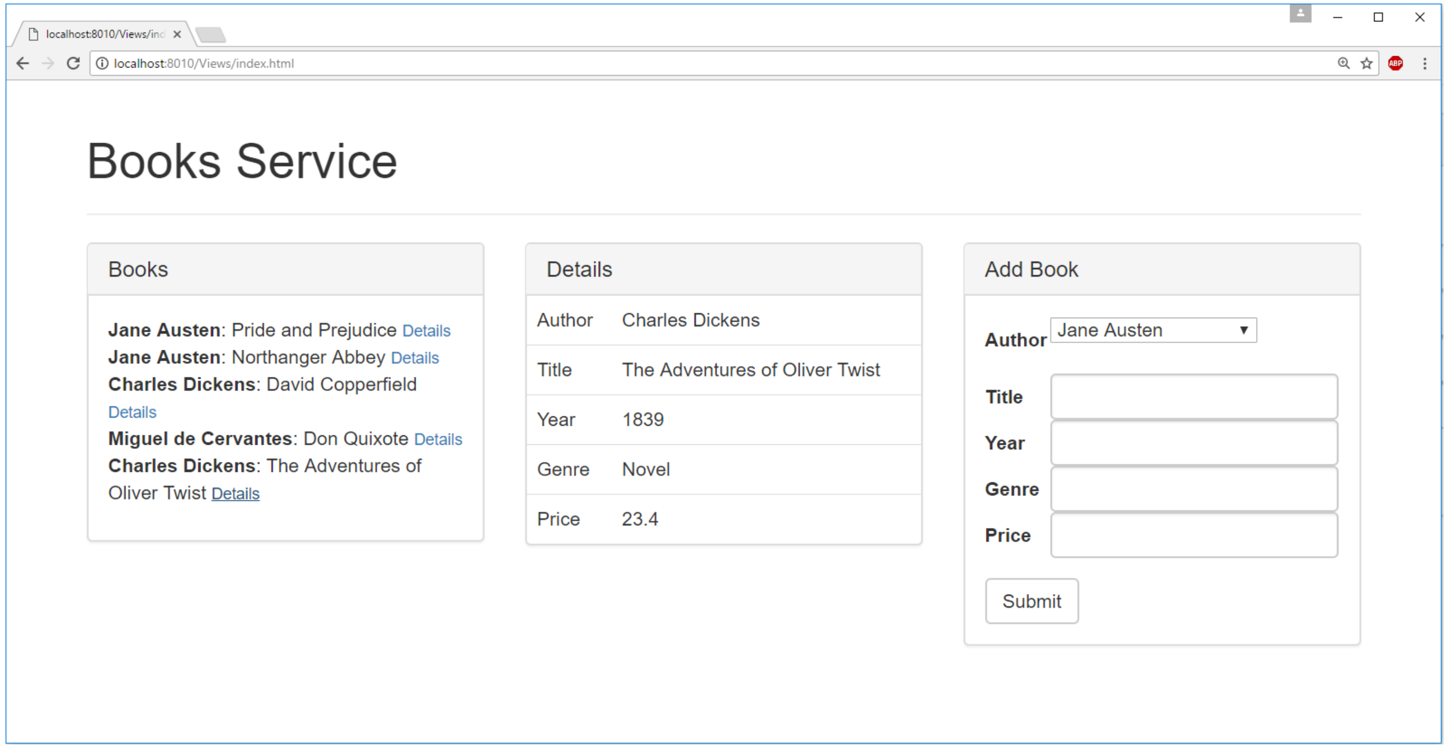
EF Modes of Operations

- **Model First** – you first define the entity data model and then EF creates the database
- **Database First** – you first create the database and then EF generates the data model
- **Code First** – you first write C# classes that correspond to database tables, and EF creates the database (the newest approach)



Web API with Entity Framework

- The following example uses ASP.NET Web API with Entity Framework 6 to create a web application that manipulates a back-end database of books
- In this example we'll create the database by using the "Code First" approach



localhost:8010/Views/index.html

Books Service

Books

Jane Austen: Pride and Prejudice Details
Jane Austen: Northanger Abbey Details
Charles Dickens: David Copperfield Details
Miguel de Cervantes: Don Quixote Details
Charles Dickens: The Adventures of Oliver Twist Details

Details

Author	Charles Dickens
Title	The Adventures of Oliver Twist
Year	1839
Genre	Novel
Price	23.4

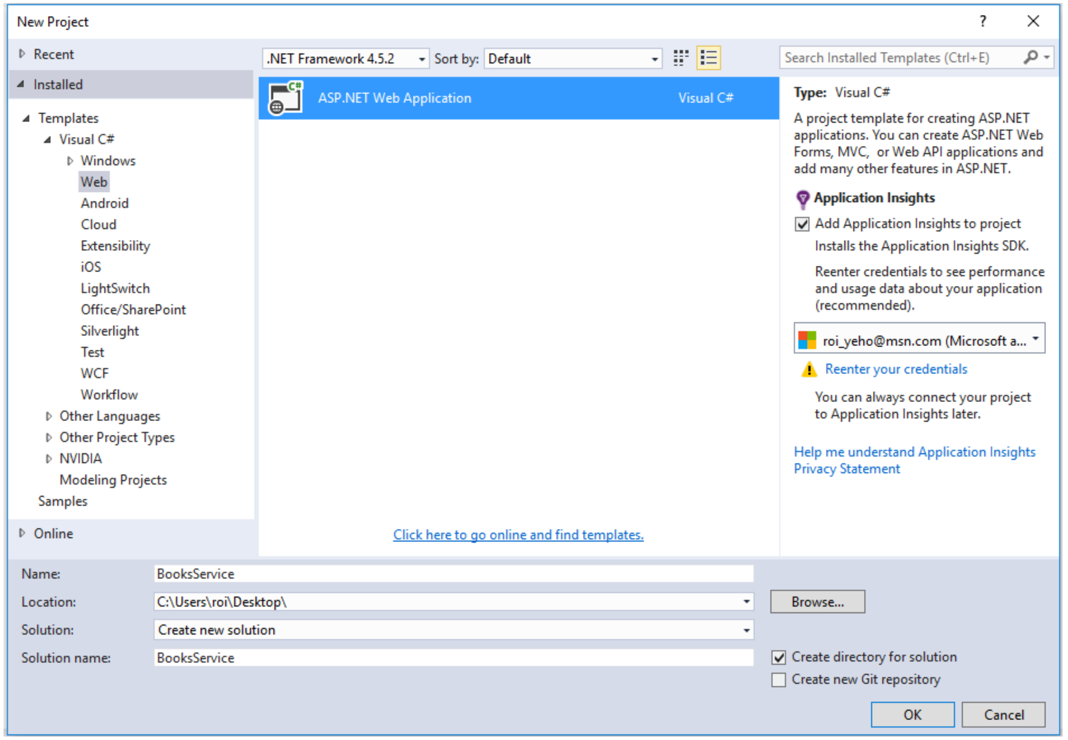
Add Book

Author	Jane Austen
Title	
Year	
Genre	
Price	

Submit

Create the Project

- In the **New Project** dialog, click **Web** in the left pane and **ASP.NET Web Application** in the middle pane
- Name the project BooksService and click **OK**



.NET Framework 4.5.2
Sort by: Default

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

☑ Add Application Insights to project

Installs the Application Insights SDK.

Reenter credentials to see performance and usage data about your application (recommended).

roi_yeho@msn.com (Microsoft a...

You can always connect your project to Application Insights later.

Name: BooksService

Location: C:\Users\roi\Desktop\

Solution: Create new solution

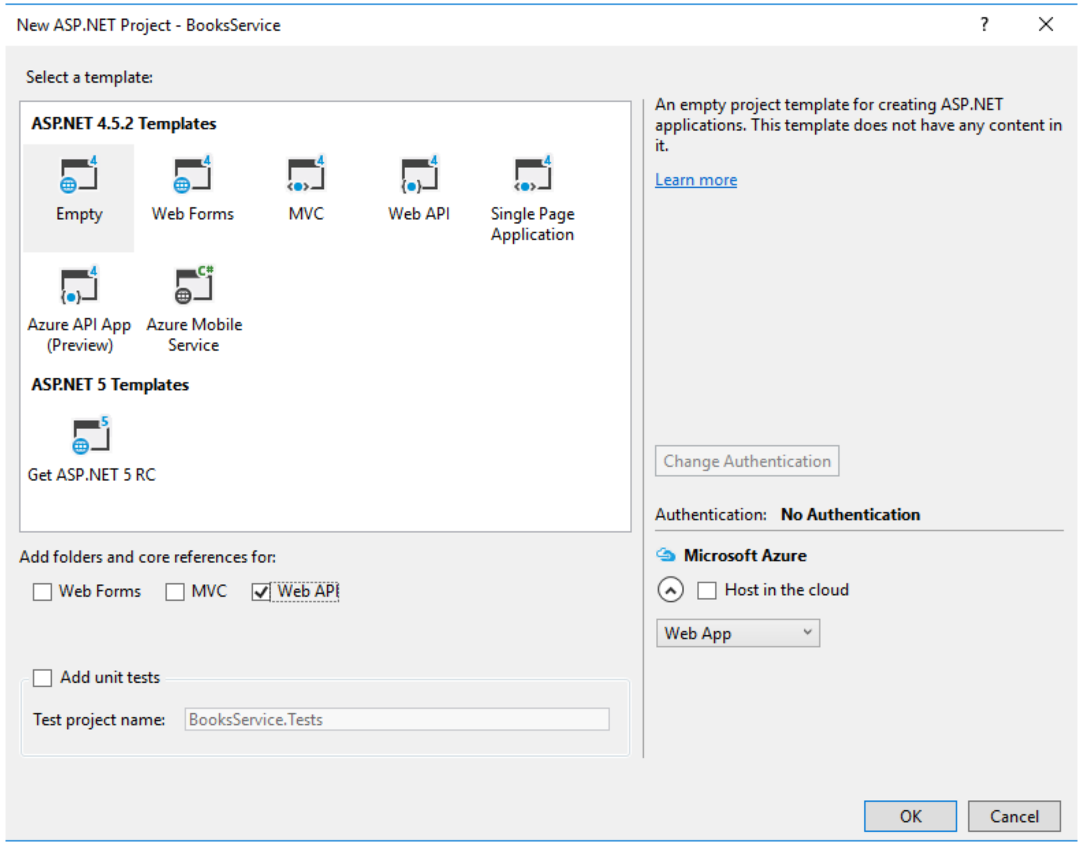
Solution name: BooksService

☑ Create directory for solution

☐ Create new Git repository

Create the Project

- In the **New ASP.NET Project** dialog, select the **Empty** template and check the Web API component



New ASP.NET Project - BooksService

Select a template:

ASP.NET 4.5.2 Templates

Empty, Web Forms, MVC, Web API, Single Page Application, Azure API App (Preview), Azure Mobile Service

ASP.NET 5 Templates

Get ASP.NET 5 RC

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

Learn more

Add folders and core references for:

☐ Web Forms ☐ MVC ☑ Web API

☐ Add unit tests

Test project name: BooksService.Tests

Authentication: **No Authentication**

Microsoft Azure

☐ Host in the cloud

Web App

Add Model Classes

- We start by defining our domain objects as POCOs (plain-old CLR objects)
- EF will use these models to create database tables

```
namespace BooksService.Models
{
    public class Author
    {
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
    }
}
```

The Id property becomes the primary key of the table (use [Key] attribute for a non-standard name)

Data annotations define extra attributes of the column (e.g., validation)

The navigation property can be used to access the related Author

```
namespace BooksService.Models
{
    public class Book
    {
        public int Id { get; set; }
        [Required]
        public string Title { get; set; }
        public string Genre { get; set; }
        public int Year { get; set; }
        [Range(0, double.MaxValue)]
        public decimal Price { get; set; }

        // Foreign key
        public int AuthorId { get; set; }

        // Navigation property
        public Author author { get; set; }
    }
}
```

EF Data Annotations

- There are three types of data annotations
  - **Validation Attributes**: Used to enforce validation rules
  - **Modeling Attributes**: Specify the intended use of class member or class relationships
  - **Display Attributes**: Specify how data from a class /member is displayed in the UI

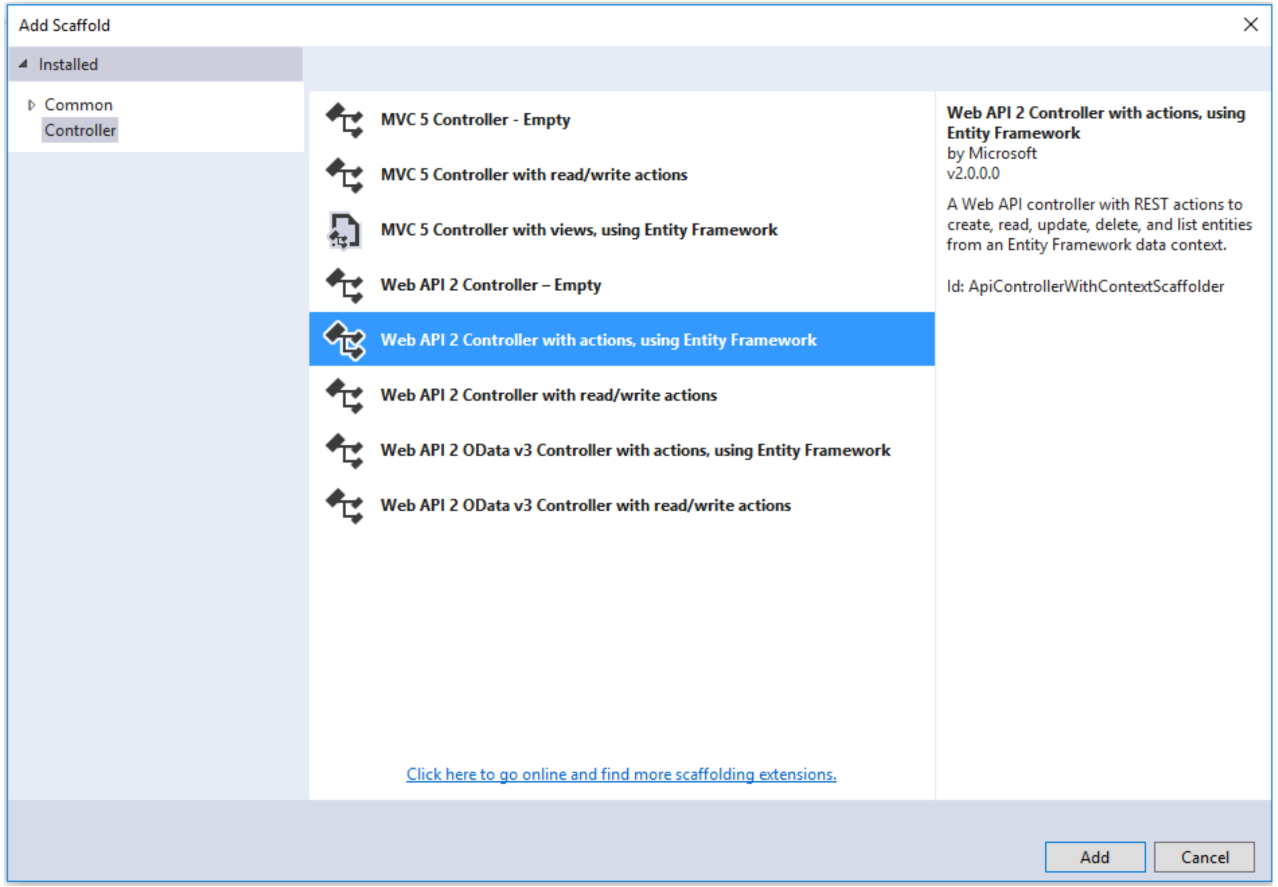
Attribute	Description
Key	Mark property as EntityKey which will be mapped to PK of the related table
Required	Force EF to ensure that property has data in it
MinLength	validates property whether it has minimum length of array or string
MaxLength	maximum length of property; also sets the maximum length of a column in the database
Range	Specifies the numeric range constraints for the value of a data field
Column	Specify column name and datatype which will be mapped with the property
Index	Create an Index for specified column
ForeignKey	Specify Foreign key property for Navigation property
NotMapped	Specify that property will not be mapped with database

Add Web API Controllers

- We'll now add Web API controllers that support CRUD operations (create, read, update, and delete)
- The controllers will use Entity Framework to communicate with the database layer
- First, build the project
  - The Web API scaffolding uses reflection to find the models, so it needs the compiled assembly
- In Solution Explorer, right-click the Controllers folder
- Select **Add**, then select **Controller**

Add Web API Controllers

- In the **Add Scaffold** dialog, select "Web API 2 Controller with actions, using Entity Framework"



Add Scaffold

- Installed
  - Common
    - Controller

MVC 5 Controller - Empty
MVC 5 Controller with read/write actions
MVC 5 Controller with views, using Entity Framework
Web API 2 Controller – Empty
Web API 2 Controller with actions, using Entity Framework
Web API 2 Controller with read/write actions
Web API 2 OData v3 Controller with actions, using Entity Framework
Web API 2 OData v3 Controller with read/write actions

Web API 2 Controller with actions, using Entity Framework
by Microsoft
v2.0.0.0

A Web API controller with REST actions to create, read, update, delete, and list entities from an Entity Framework data context.

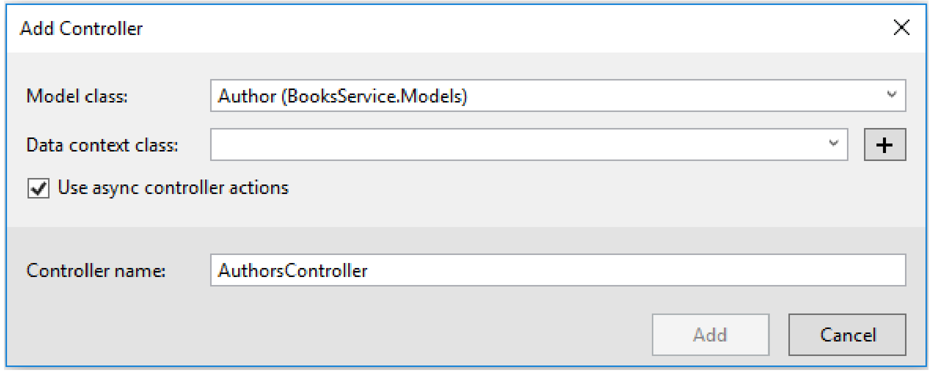
Id: ApiControllerWithContextScaffolder

Click here to go online and find more scaffolding extensions.

Add Cancel

Add Web API Controllers

- In the **Add Controller** dialog, do the following:
  - In the **Model class** dropdown, select the Author class
  - Check "Use async controller actions"
  - Leave the controller name as "AuthorsController"
  - Click plus (+) button next to **Data Context Class**
- In the **New Data Context** dialog, leave the default name
- Click **Add** to complete the **Add Controller** dialog

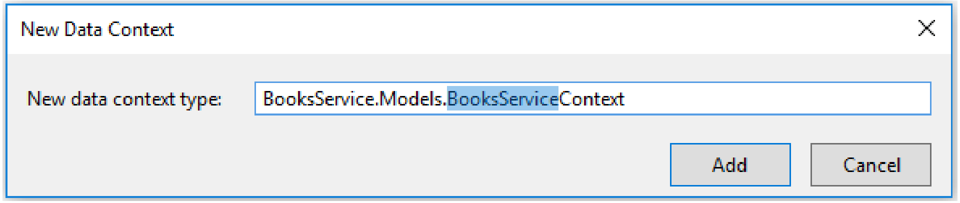

Add Controller

Model class: Author (BooksService.Models)

Data context class:

☑ Use async controller actions

Controller name: AuthorsController

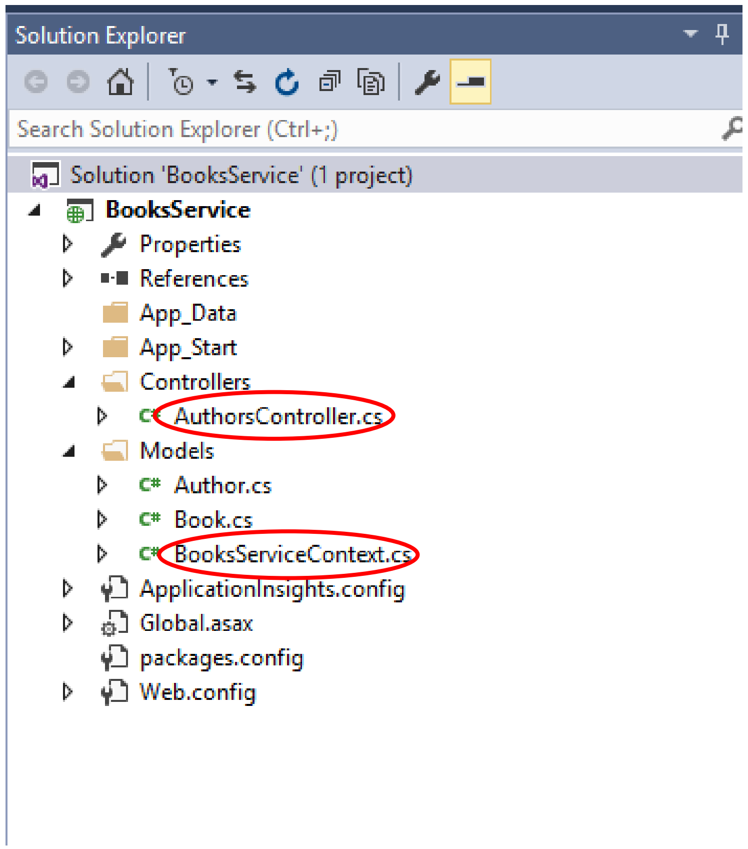

New Data Context

New data context type: BooksService.Models.BooksServiceContext

Add Web API Controllers

- The dialog adds two classes to your project:
  - **AuthorsController** defines a Web API controller
    - The controller implements the REST API that clients use to perform CRUD operations on the list of authors
  - **BooksServiceContext** manages entity objects at runtime
    - which includes populating objects with data from a database, change tracking, and persisting data to the database

- Repeat the same steps for creating the BooksController
  - This time, select Book for the model class, and select the existing BookServiceContext class for the data context class



Books Context Generated Code

- Derives from DbContext
- Adds DbSet for each entity
- DbContext’s Ctor can receive DB name or connection string

```
namespace BooksService.Models
{
    public class BooksServiceContext : DbContext
    {
        public BooksServiceContext() : base("name=BooksServiceContext")
        {
        }

        public DbSet<Author> Authors { get; set; }

        public DbSet<Book> Books { get; set; }
    }
}
```

Web API Controller Generated Code

```
public class AuthorsController : ApiController
{
    private BooksServiceContext db = new BooksServiceContext();

    // GET: api/Authors
    public IQueryable<Author> GetAuthors()
    {
        return db.Authors;
    }
    // GET: api/Authors/5
    [ResponseType(typeof(Author))]
    public async Task<IHttpActionResult> GetAuthor(int id)
    {
        Author author = await db.Authors.FindAsync(id);
        if (author == null)
        {
            return NotFound();
        }
        return Ok(author);
    }
    // PUT: api/Authors/5
    [ResponseType(typeof(void))]
    public async Task<IHttpActionResult> PutAuthor(int id,
Author author)
    {
        …
    }
```

```
    // POST: api/Authors
    [ResponseType(typeof(Author))]
    public async Task<IHttpActionResult> PostAuthor(Author author)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }

        db.Authors.Add(author);
        await db.SaveChangesAsync();
        return CreatedAtRoute("DefaultApi", new { id = author.Id },
author);
    }
    // DELETE: api/Authors/5
    [ResponseType(typeof(Author))]
    public async Task<IHttpActionResult> DeleteAuthor(int id)
    {
        …
    }
    protected override void Dispose(bool disposing)
    {
        …
    }
}
```

Lazy vs. Eager Loading

- With lazy loading, EF automatically loads a related entity when the navigation property for that entity is dereferenced
  - Lazy loading requires multiple database trips
  - Generally, you want lazy loading disabled for objects that you serialize
- With *eager loading*, EF loads related entities as part of the initial database query
- To perform eager loading, use the **Include()** method:

```
public class AuthorsController : ApiController
{
    private BooksServiceContext db = new BooksServiceContext();

    // GET: api/Authors
    public IQueryable<Author> GetAuthors()
    {
        return db.Books.Include(b => b.Author);
    }
}
```

Code First Migration

- EF Code First can monitor changes to the conceptual model
  - Automatically updates the database schema when your model changes, without losing data
- Code-First has two commands for code based migration:
  - **Add-migration:** generates the code for the database to apply the changes you have made to your domain classes
  - **Update-database:** executes the code that you created using "Add-Migration" command
- To enable migrations, from the **Tools** menu, select **Library Package Manager**, then select **Package Manager Console**
- In the Package Manager Console window, enter the following command:

```
Enable-Migrations
```

  - This command adds a folder named Migrations to your project, plus a code file named Configuration.cs in the Migrations folder

Seeding the Database

- You can insert data into your database tables during database initialization
- This enables you to provide some test or some default master data
- Open the Configuration.cs file
- Then add the following code to the **Configuration.Seed** method:

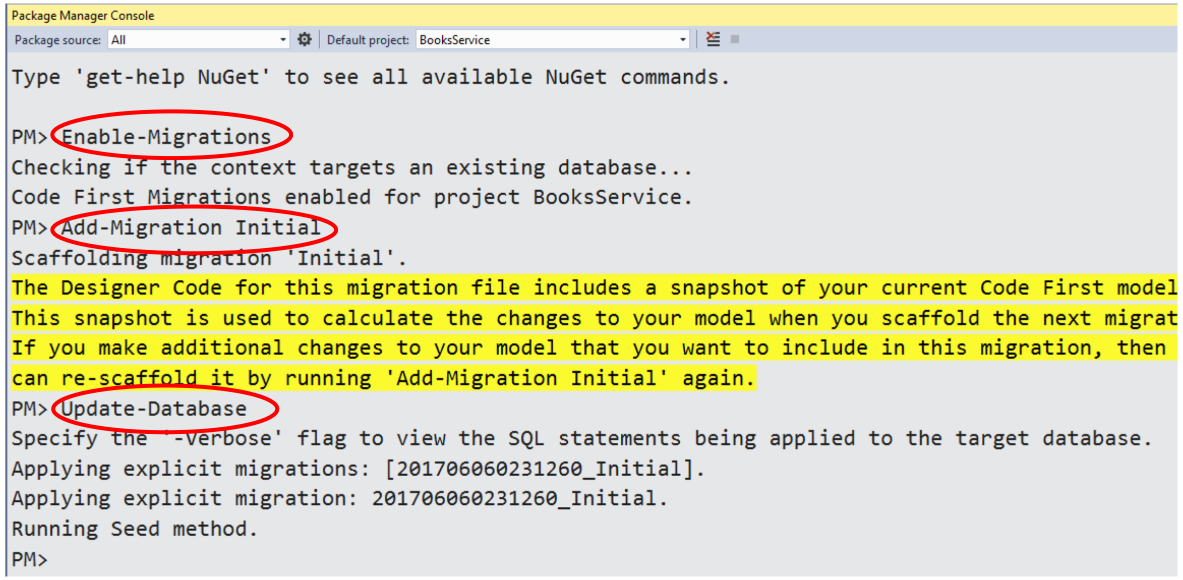
```
protected override void
Seed(BooksService.Models.BooksServiceContext context) {
    context.Authors.AddOrUpdate(x => x.Id,
        new Author() { Id = 1, Name = "Jane Austen" },
        new Author() { Id = 2, Name = "Charles Dickens" },
        new Author() { Id = 3, Name = "Miguel de Cervantes" }
    );

    context.Books.AddOrUpdate(x => x.Id,
        new Book()
        {
            Id = 1,
            Title = "Pride and Prejudice",
            Year = 1813,
            AuthorId = 1,
            Price = 9.99M,
            Genre = "Comedy of manners"
        },
        new Book()
        {
            Id = 2,
            Title = "Northanger Abbey",
            Year = 1817,
            AuthorId = 1,
            Price = 12.95M,
            Genre = "Gothic parody"
        },
        …
    );
}
```

Code First Migration

- In the Package Manager Console window, type the following commands:

```
Add-Migration Initial
Update-Database
```

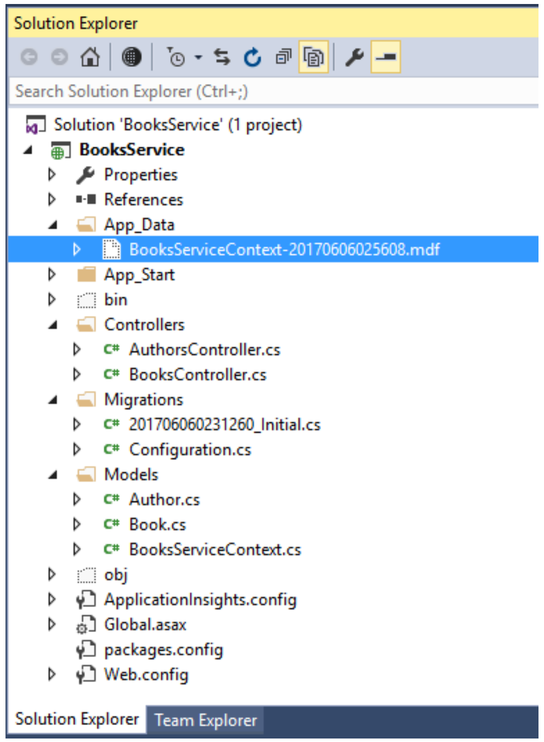
  - The first command generates code that creates the database
  - The second command executes that code



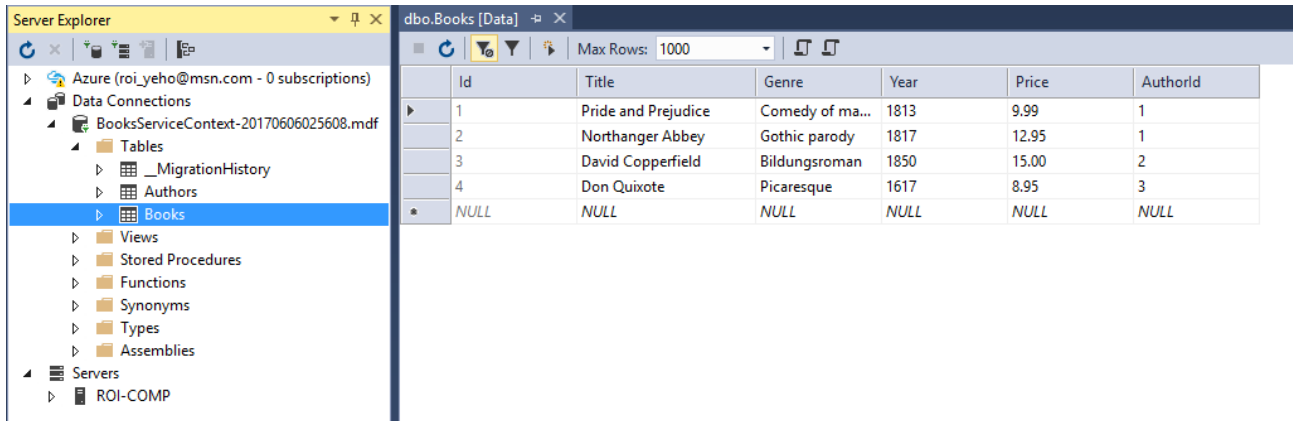
```
Type 'get-help NuGet' to see all available NuGet commands.

PM> Enable-Migrations
Checking if the context targets an existing database...
Code First Migrations enabled for project BooksService.
PM> Add-Migration Initial
Scaffolding migration 'Initial'.
The Designer Code for this migration file includes a snapshot of your current Code First model
This snapshot is used to calculate the changes to your model when you scaffold the next migrat
If you make additional changes to your model that you want to include in this migration, then
can re-scaffold it by running 'Add-Migration Initial' again.
PM> Update-Database
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying explicit migrations: [201706060231260_Initial].
Applying explicit migration: 201706060231260_Initial.
Running Seed method.
PM>
```

LocalDB

- The database is created locally, using SQL Server Express LocalDB
- The database file (.mdf) is saved in the App_Data folder
  - Its name will be based on the name of the DbContext class
  - Click **Show All Files** in Solution Explorer to see the file
- Double-clicking the .mdf file will open the database in the Server Explorer window
  - You can expand the nodes to see the tables that EF created



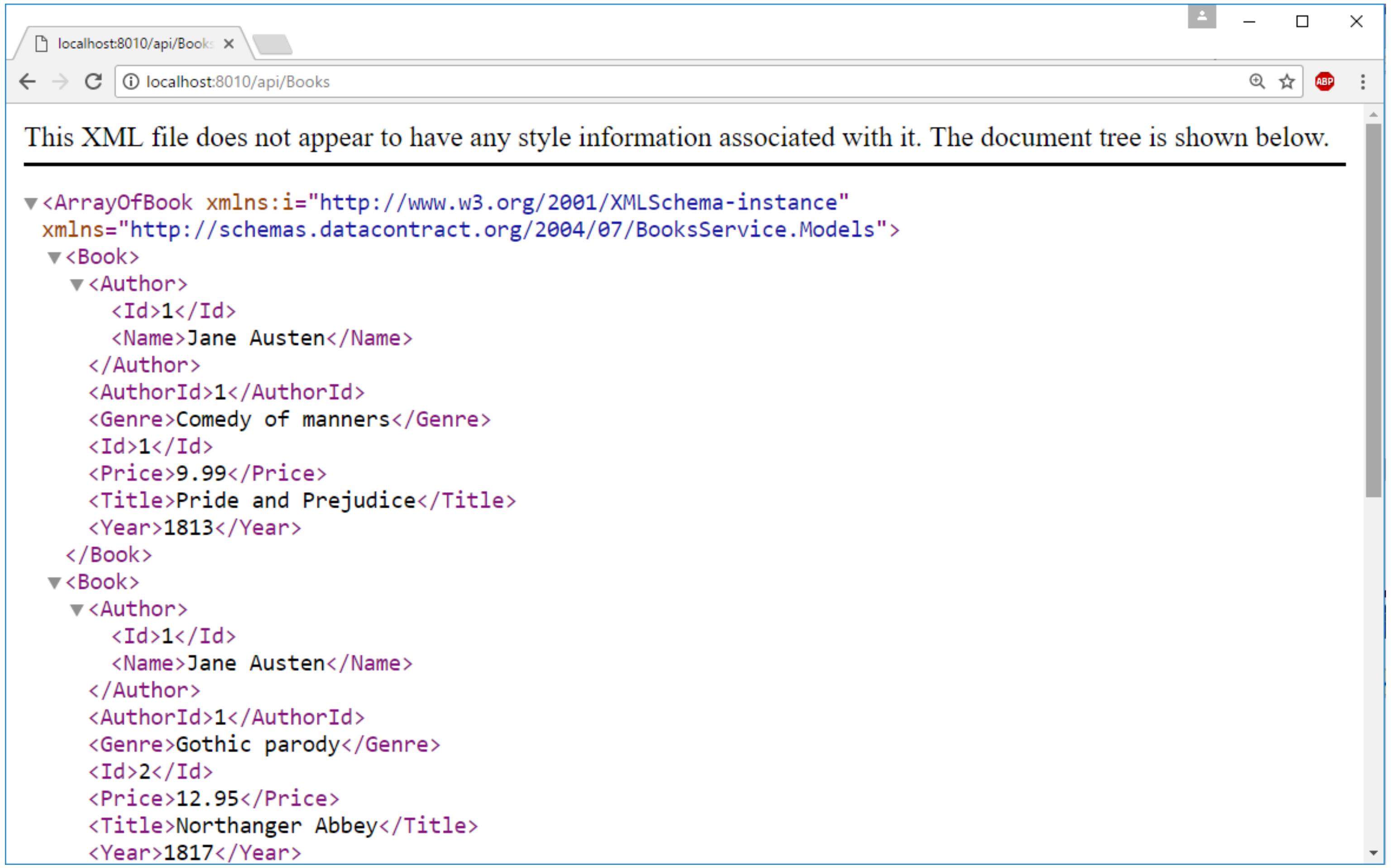
```
Solution 'BooksService' (1 project)
  BooksService
    Properties
    References
    App_Data
      BooksServiceContext-20170606025608.mdf
    App_Start
    bin
    Controllers
      AuthorsController.cs
      BooksController.cs
    Migrations
      201706060231260_Initial.cs
      Configuration.cs
    Models
      Author.cs
      Book.cs
      BooksServiceContext.cs
    obj
    ApplicationInsights.config
    Global.asax
    packages.config
    Web.config
```



```
Azure (roi_yeho@msn.com - 0 subscriptions)
Data Connections
  BooksServiceContext-20170606025608.mdf
    Tables
      _MigrationHistory
      Authors
      Books
    Views
    Stored Procedures
    Functions
    Synonyms
    Types
    Assemblies
Servers
  ROI-COMP
```

Id	Title	Genre	Year	Price	AuthorId
1	Pride and Prejudice	Comedy of ma...	1813	9.99	1
2	Northanger Abbey	Gothic parody	1817	12.95	1
3	David Copperfield	Bildungsroman	1850	15.00	2
4	Don Quixote	Picaresque	1617	8.95	3
NULL	NULL	NULL	NULL	NULL	NULL

Testing the Controller



localhost:8010/api/Books

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<ArrayOfBook xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://schemas.datacontract.org/2004/07/BooksService.Models">
  ▼<Book>
    ▼<Author>
       <Id>1</Id>
       <Name>Jane Austen</Name>
     </Author>
     <AuthorId>1</AuthorId>
     <Genre>Comedy of manners</Genre>
     <Id>1</Id>
     <Price>9.99</Price>
     <Title>Pride and Prejudice</Title>
     <Year>1813</Year>
   </Book>
  ▼<Book>
    ▼<Author>
       <Id>1</Id>
       <Name>Jane Austen</Name>
     </Author>
     <AuthorId>1</AuthorId>
     <Genre>Gothic parody</Genre>
     <Id>2</Id>
     <Price>12.95</Price>
     <Title>Northanger Abbey</Title>
     <Year>1817</Year>
```

Create the JavaScript Client

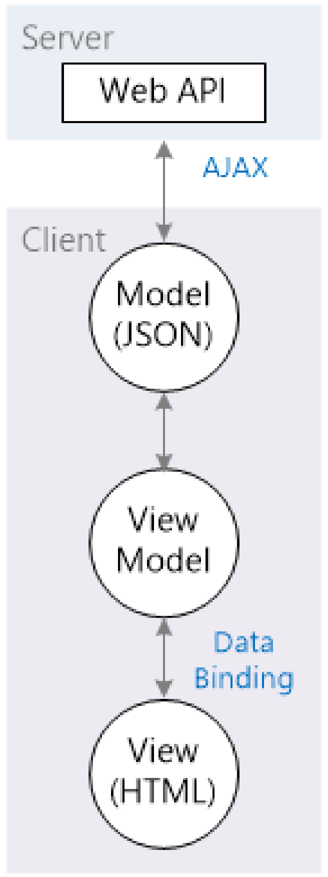
- Now we will create the client for the application, using HTML, JavaScript, and the Knockout.js library
- We'll build the client app in stages:
  - Showing a list of books
  - Showing a book detail
  - Adding a new book

Knockout

- Knockout is a JS implementation of the MVVM pattern:
  - The **model** is the server-side representation of the data in the business domain (in our case, books and authors)
  - The **view** is the presentation layer (HTML)
  - The **view model** is a JS object that holds the models
    - It represents abstract features of the view, e.g. "a list of books"
- To add the knockout library:
  - Open **Package Manager Console**
  - In the console enter the following command:

```
Install-Package knockoutjs
```

    - This command adds the Knockout files to the Scripts folder



Create the View Model

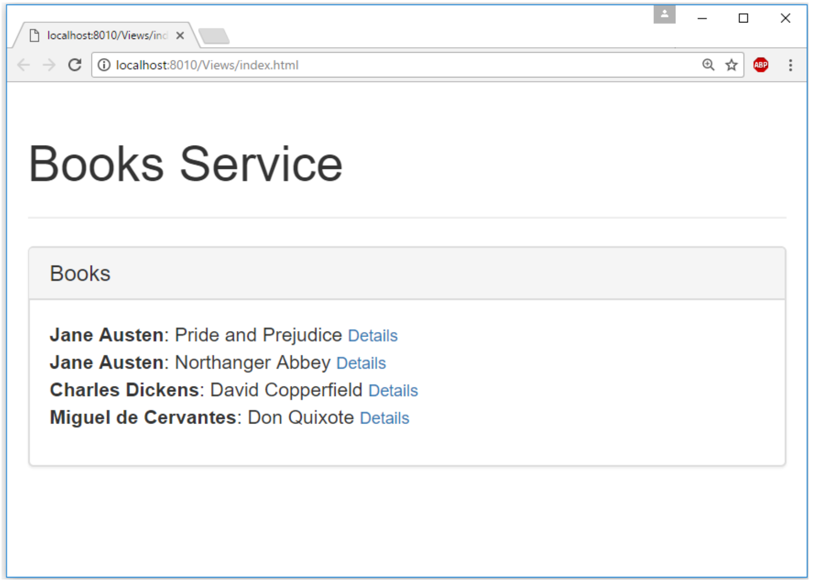
- Add a JavaScript file named app.js to the Scripts folder
- Paste in the following code:

```
var ViewModel = function () {
    var self = this; // make 'this' available to subfunctions or closures
    self.books = ko.observableArray(); // enables data binding

    var booksUri = "/api/books";

    function getAllBooks() {
        $.getJSON(booksUri).done(function (data) {
            self.books(data);
        });
    }

    // Fetch the initial data
    getAllBooks();
};

ko.applyBindings(new ViewModel()); // sets up the data binding
```

Create the View

- Add an HTML file named index.html to the Views folder

```
<div class="container">
    <div class="page-header">
        <h1>Books Service</h1>
    </div>
    <div class="row">
        <div class="col-sm-4">
            <div class="panel panel-default">
                <div class="panel-heading">
                    <h2 class="panel-title">Books</h2>
                </div>
                <div class="panel-body">
                    <ul class="list-unstyled" data-bind="foreach: books">
                        <li>
                            <span data-bind="text: Author.Name"
style="fontweight:bold"></span>:
                            <span data-bind="text: Title"></span>
                            <a href="#" style="font-size:smaller">Details</a>
                        </li>
                    </ul>
                </div>
            </div>
        </div>
    </div>
</div>
```



localhost:8010/Views/index.html

Books Service

Books

Jane Austen: Pride and Prejudice Details
Jane Austen: Northanger Abbey Details
Charles Dickens: David Copperfield Details
Miguel de Cervantes: Don Quixote Details

Data Binding

- The **data-bind** attribute links the HTML to the view model, e.g.:

```
<ul class="list-unstyled" data-bind="foreach: books">
```

  - The **foreach** binding tells Knockout to loop through the contents of the books array
  - For each item in the array, Knockout creates a new <li> element
  - The books property of the view model is defined as an **observableArray**, which allows the view to respond to changes in the array
- Bindings inside the context of the foreach refer to properties on the array item, e.g.:

```
<span data-bind="text: Title"></span>
```

  - The "text" binding reads the Title property of each book

Display Item Details

- We will now add the ability to view details for each book
- In app.js, add to the following code to the view model:

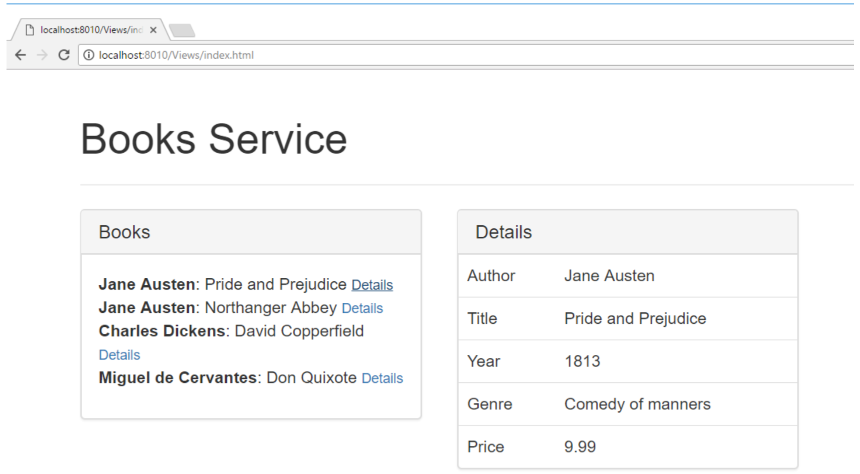
```
var ViewModel = function () {
    …
    self.currBook = ko.observable();
    self.getBookDetails = function (book) {
        $.getJSON(booksUri + "/" + book.Id).done(function (data) {
            self.currBook(data);
        });
    }
};
```

- ko.observable() – an object that can notify subscribers about changes
  - Updates the UI automatically when the view model changes

Display Item Details

- In Views/index.html, add a data-bind element to the Details link:

```
<!-- ko if:currBook() -->
<div class="col-sm-4">
    <div class="panel panel-default">
        <div class="panel-heading">
            <h2 class="panel-title">Details</h2>
        </div>
        <table class="table">
            <tr><td>Author</td><td data-bind="text:
currBook().Author.Name"></td></tr>
            <tr><td>Title</td><td data-bind="text: currBook().Title"></td></tr>
            <tr><td>Year</td><td data-bind="text: currBook().Year"></td></tr>
            <tr><td>Genre</td><td data-bind="text: currBook().Genre"></td></tr>
            <tr><td>Price</td><td data-bind="text: currBook().Price"></td></tr>
        </table>
    </div>
</div>
<!-- /ko -->
```



localhost:8010/Views/index.html

Books Service

Books

Jane Austen: Pride and Prejudice Details
Jane Austen: Northanger Abbey Details
Charles Dickens: David Copperfield Details
Miguel de Cervantes: Don Quixote Details

Details

Author	Jane Austen
Title	Pride and Prejudice
Year	1813
Genre	Comedy of manners
Price	9.99

  - "<!– ko if: currBook()-->" causes this section of markup to be displayed only when currBook is non-null

Add a New Book

- We will now add the ability for users to create a new book
- In app.js, add the following code to the view model:

```
var authorsUri = '/api/authors/';

function getAuthors() {
    $.getJSON(authorsUri).done(function (data) {
        self.authors(data);
    });
}
self.addBook = function () {
    var book = {
        AuthorId: self.newBook.Author().Id,
        Genre: self.newBook.Genre(),
        Price: self.newBook.Price(),
        Title: self.newBook.Title(),
        Year: self.newBook.Year()
    };
    $.post(booksUri, book).done(function (item) {
        self.books.push(item);
    });
}
getAuthors();
```

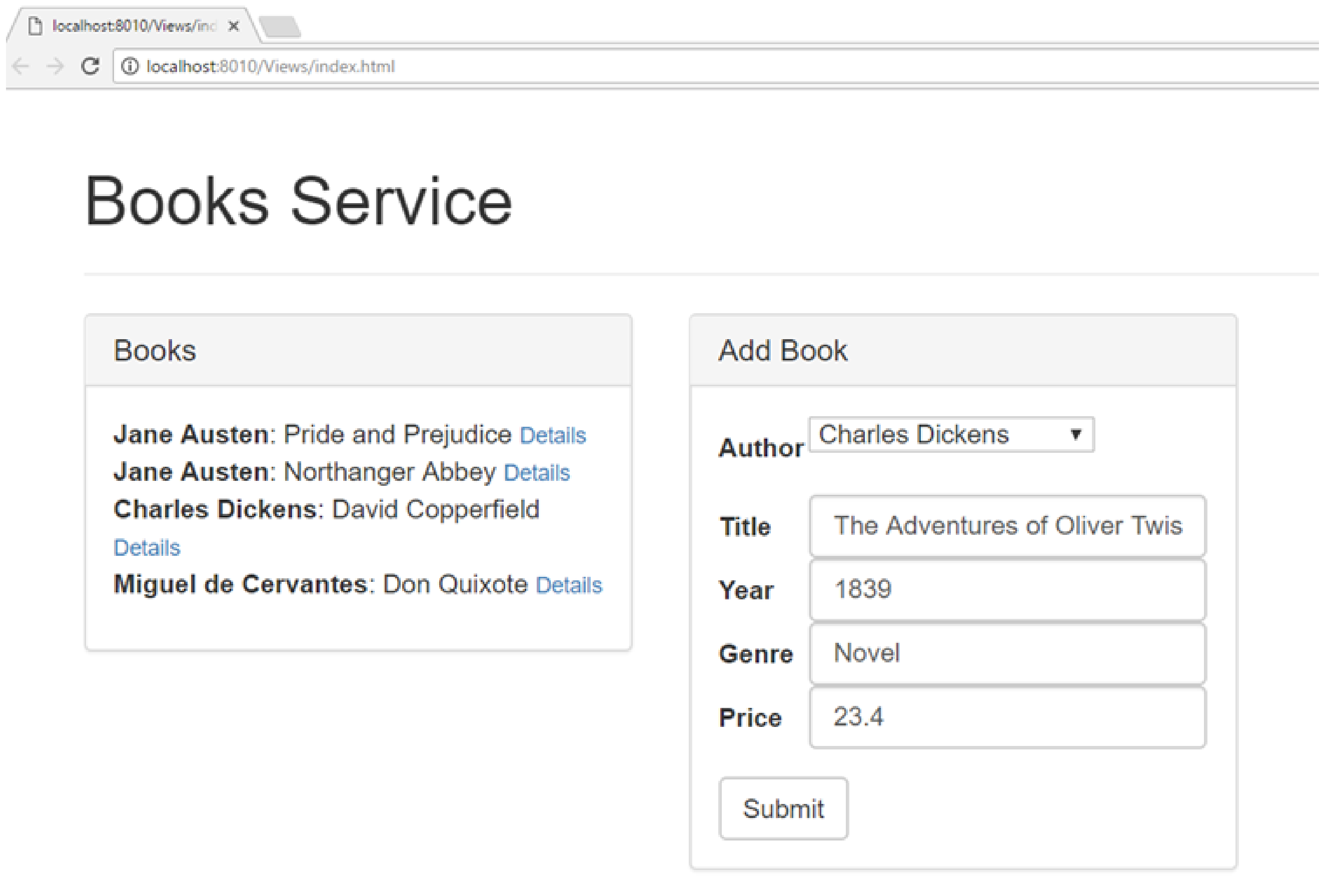
Add a New Book

- In Index.html, add the following markup:

```
<div class="col-sm-4">
    <div class="panel panel-default">
        <div class="panel-heading">
            <h2 class="panel-title">Add Book</h2>
        </div>
        <div class="panel-body">
            <form class="form-horizontal" data-bind="submit:
addBook">
                <div class="form-group">
                    <label for="inputAuthor" class="col-sm-2
control-label">Author</label>
                    <div class="col-sm-10">
                        <select data-bind="options:authors,
optionsText: 'Name', value: newBook.Author"></select>
                    </div>
                </div>
                <div class="form-group" data-bind="with:
newBook">
                    <label for="inputTitle" class="col-sm-2
control-label">Title</label>
                    <div class="col-sm-10">
                        <input type="text" class="form-
control" id="inputTitle" data-bind="value:Title" />
                    </div>
```

```
                    <label for="inputYear" class="col-sm-2
control-label">Year</label>
                    <div class="col-sm-10">
                        <input type="number" class="form-
control" id="inputYear" data-bind="value:Year" />
                    </div>
                    <label for="inputGenre" class="col-sm-2
control-label">Genre</label>
                    <div class="col-sm-10">
                        <input type="text" class="form-control"
id="inputGenre" data-bind="value:Genre" />
                    </div>
                    <label for="inputPrice" class="col-sm-2
control-label">Price</label>
                    <div class="col-sm-10">
                        <input type="number" step="any"
class="form-control" id="inputPrice" data-bind="value:Price" />
                    </div>
                </div>
                <button type="submit" class="btn btn-
default">Submit</button>
            </form>
        </div>
    </div>
</div>
```

Add a New Book



localhost:8010/Views/index.html

Books Service

Books

Jane Austen: Pride and Prejudice Details
Jane Austen: Northanger Abbey Details
Charles Dickens: David Copperfield Details
Miguel de Cervantes: Don Quixote Details

Add Book

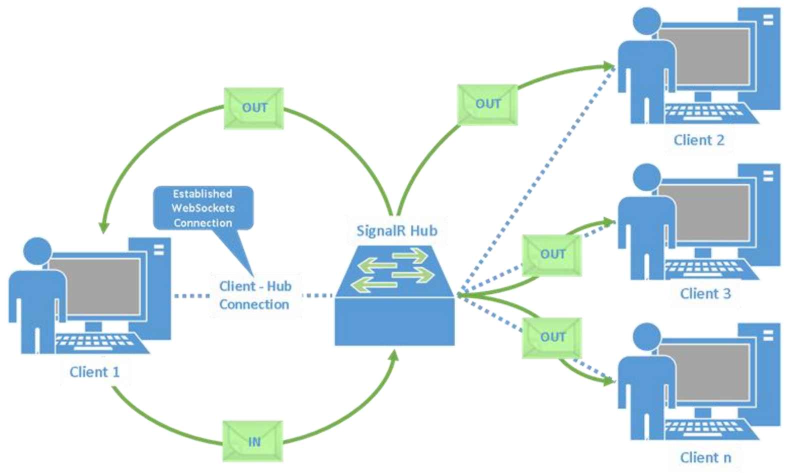
Author	Charles Dickens
Title	The Adventures of Oliver Twis
Year	1839
Genre	Novel
Price	23.4

Submit

SignalR

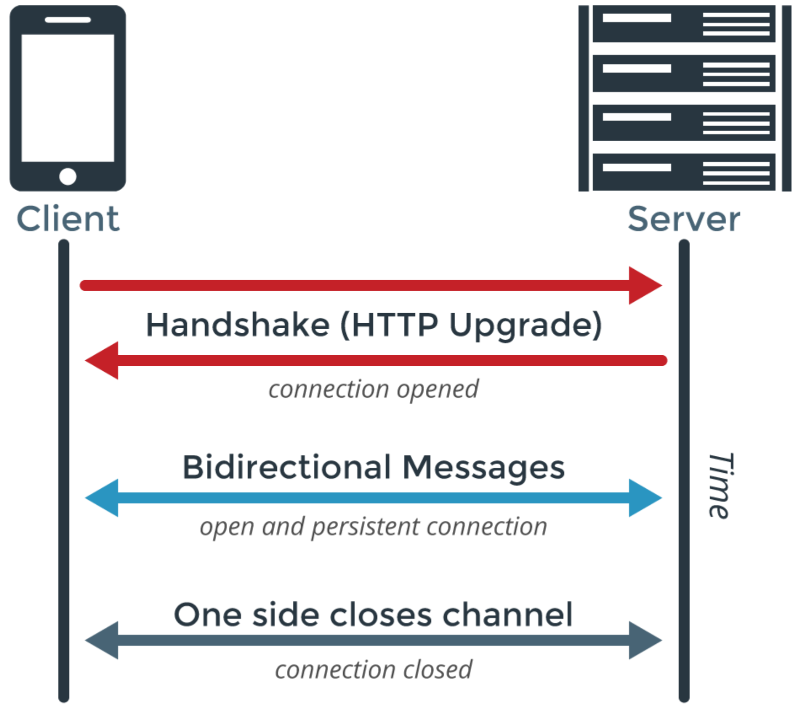
ASP.NET SignalR

- A framework for building real-time web applications
- Supports "server push" functionality, in which server code can call out to client code in the browser
- The connection between the client and server is persistent, unlike a classic HTTP connection, which is re-established for each communication
- Useful for chatrooms, dashboards and monitoring applications, collaborative applications, etc.
- A signalR application consists of two components:
  - a hub as the main coordination object on the server
  - SignalR jQuery library to send and receive messages

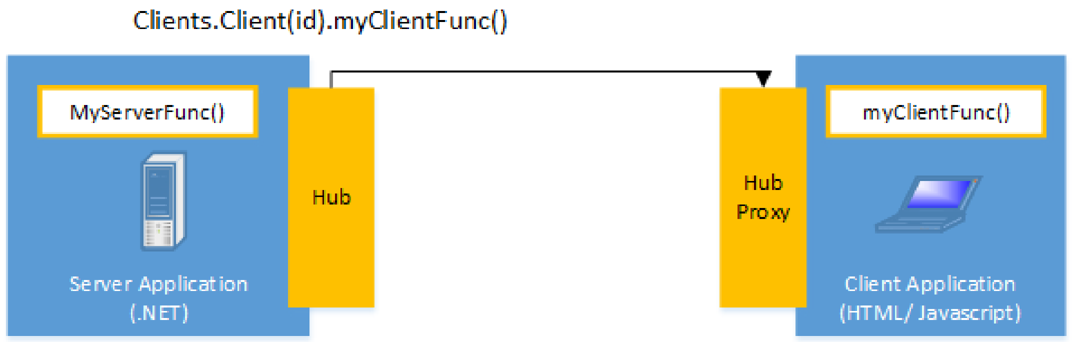


SignalR and WebSockets

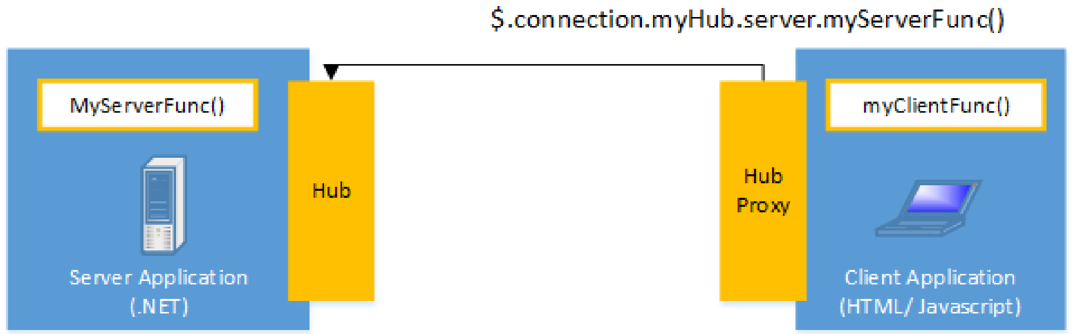
- **WebSockets** is a full-duplex communication protocol that allows to open an interactive session between the user's browser and a server
- Allows the client to receive event-driven responses without having to poll the server
- SignalR uses WebSocket where available, and falls back to older transports where necessary
- SignalR will continue to be updated to support changes in the underlying transport



Communication with SignalR


Clients.Client(id).myClientFunc()
MyServerFunc()
Hub
Server Application (.NET)
Hub Proxy
myClientFunc()
Client Application (HTML/ Javascript)

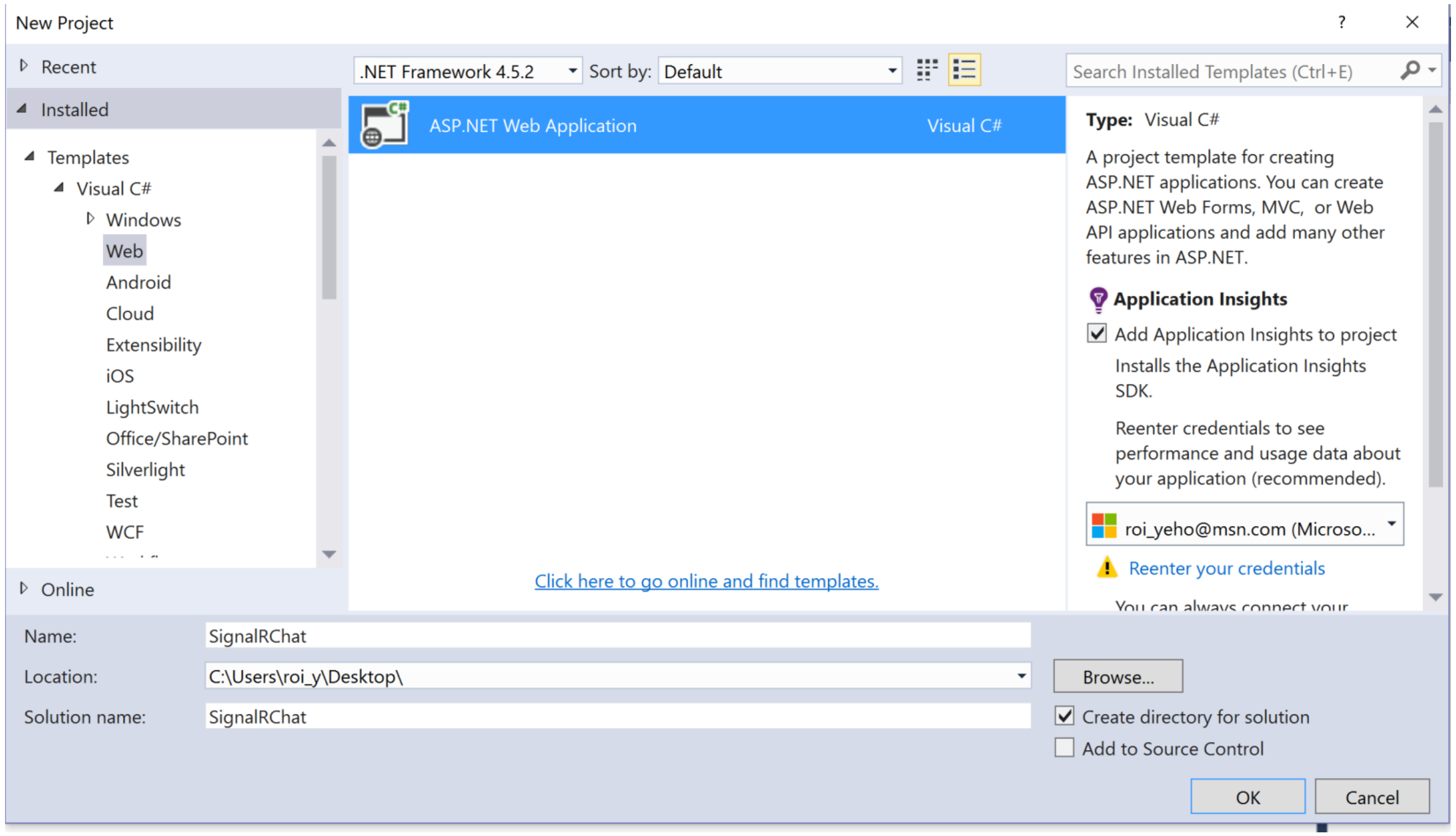
Server invocation of client method myClientFunc()


$.connection.myHub.server.myServerFunc()
MyServerFunc()
Hub
Server Application (.NET)
Hub Proxy
myClientFunc()
Client Application (HTML/ Javascript)

Client invocation of server method MyServerFunc()

SignalR Chat Example

- Create an empty ASP.NET Web Application



New Project

Recent
Installed
Templates
Visual C#
Windows
Web
Android
Cloud
Extensibility
iOS
LightSwitch
Office/SharePoint
Silverlight
Test
WCF
Online

.NET Framework 4.5.2 Sort by: Default

Search Installed Templates (Ctrl+E)

ASP.NET Web Application Visual C#

Click here to go online and find templates.

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights

☑ Add Application Insights to project

Installs the Application Insights SDK.

Reenter credentials to see performance and usage data about your application (recommended).

roi_yeho@msn.com (Microso...

Reenter your credentials

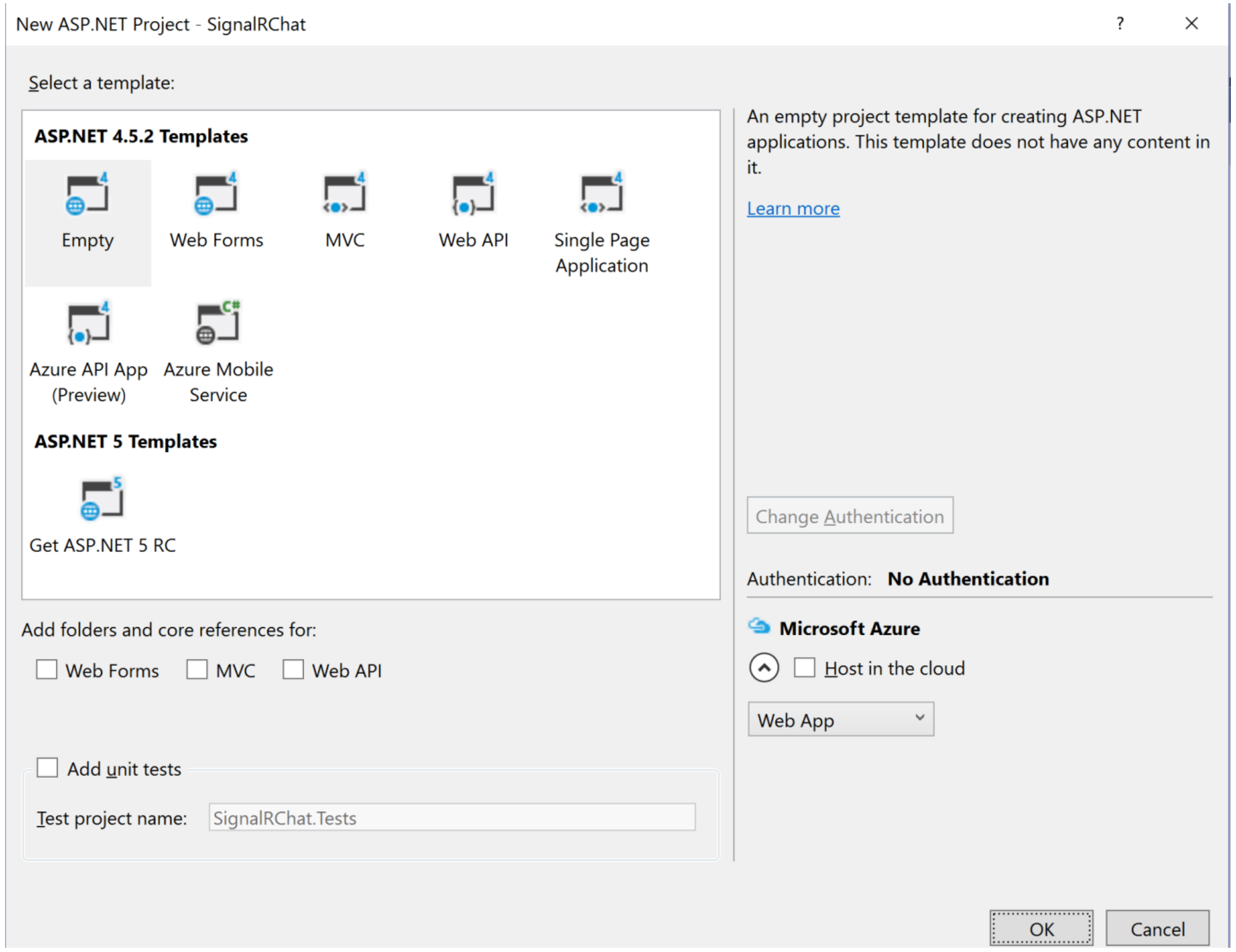
You can always connect your

Name: SignalRChat
Location: C:\Users\roi_y\Desktop\
Solution name: SignalRChat

Browse...
☑ Create directory for solution
☐ Add to Source Control

OK Cancel

SignalR Chat Example



New ASP.NET Project - SignalRChat

Select a template:

ASP.NET 4.5.2 Templates

ASP.NET 5 Templates

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

Learn more

Authentication: **No Authentication**

Microsoft Azure

☐ Host in the cloud

Web App

Add folders and core references for:

☐ Web Forms ☐ MVC ☐ Web API

☐ Add unit tests

Test project name: SignalRChat.Tests

SignalR Chat Example

- Add SignalR to the project by opening the **Tools | Library Package Manager | Package Manager Console** and running the command:

```
install-package Microsoft.AspNet.SignalR
```

  - This step will add a set of script files and assembly references that support SignalR
- In **Solution Explorer**, right-click the project, select **Add Item**
- Choose SignalR Hub Class (v2)
- Name the class **ChatHub.cs** and add it to the project
- Replace the code in the new **ChatHub** class with the following code

ChatHub Class

Public methods on the hub can be called from the client

The **Clients** dynamic property refers to all clients connected to this hub

```
namespace SignalRChat
{
    public class ChatHub : Hub
    {
        public void Send(string name, string message)
        {
            // Call the broadcastMessage method to update clients
            Clients.All.broadcastMessage(name, message);
        }
    }
}
```

- To call a specific client use **Clients.Client(clientId)**
- **Context.ConnectionId** retrieves the id of the client that currently invoked the method on the hub

OWIN Startup Class

- **OWIN** (Open Web Interface for .NET) is a standard for an interface between .NET Web applications and Web servers
- Every OWIN Application has a startup class where you specify components for the application pipeline
- In **Solution Explorer**, right-click the project, then click **Add Class | OWIN Startup Class**. Name the new class Startup and change its contents to:

```
namespace SignalRChat
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

SignalR and jQuery

- In **Solution Explorer**, right-click the project, then click **Add | HTML Page**
- Name the new page index.html
- Replace the default code in the HTML page with the following code:

```html
<!DOCTYPE html>
<html>
<head>
    <title>SignalR Simple Chat</title>
    <style type="text/css">
        .container {
            background-color: #99CCFF;
            border: thick solid #808080;
            padding: 20px;
            margin: 20px;
        }
    </style>
</head>
<body>
    <div class="container">
        <input type="text" id="message" />
        <input type="button" id="btnSendMessage" value="Send" />
        <ul id="chat">
        </ul>
    </div>

    <script src="Scripts/jquery-3.1.1.js"></script>
    <script src="Scripts/jquery.signalR-2.2.2.js"></script>
    <!-- Reference the autogenerated SignalR hub script -->
    <script src="signalr/hubs"></script>
    <script src="Scripts/chatclient.js"></script>
</body>
</html>
```

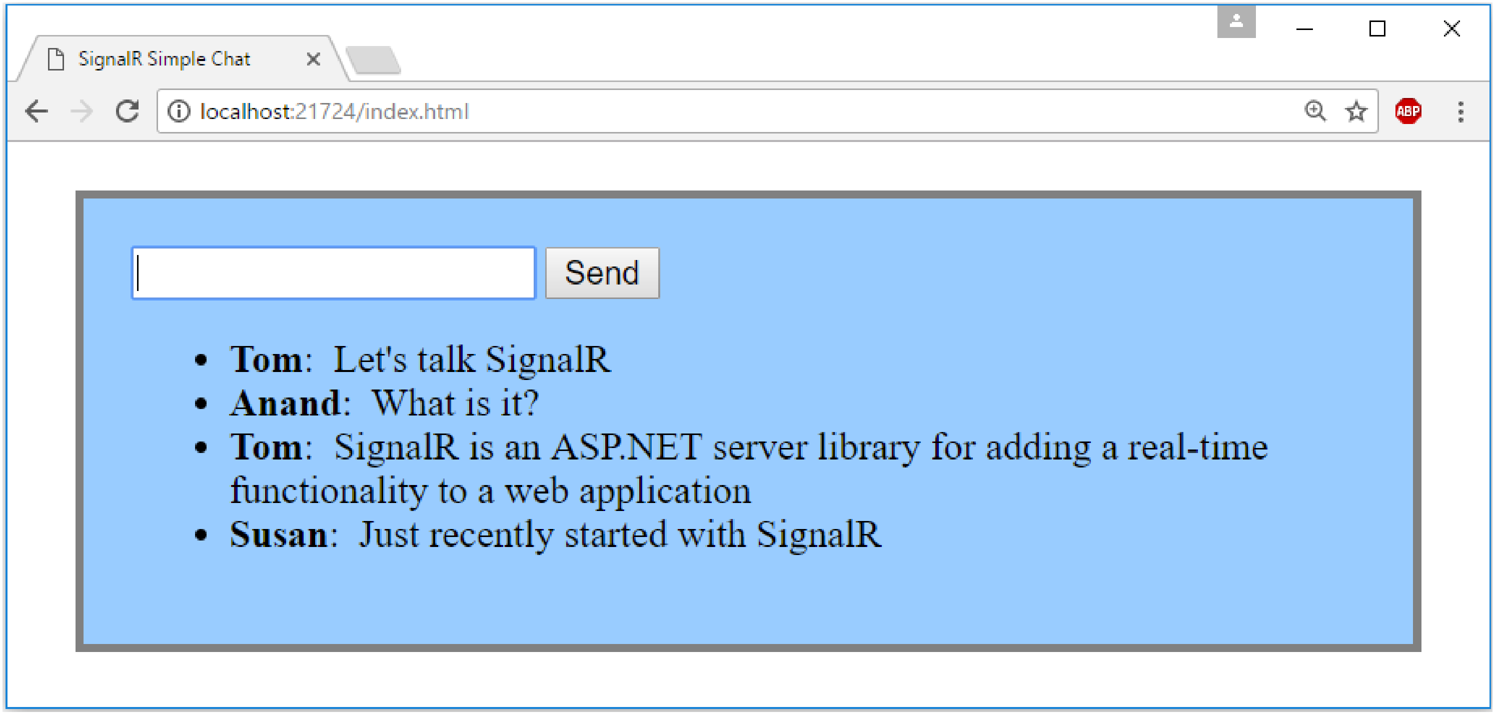
SignalR and jQuery

- Now add the file chatclient.js
- The essential tasks in the code are:
  - Declaring a proxy to reference the hub
  - Declaring a function that the server can call to push content to clients
  - Starting a connection to send messages to the hub

```javascript
// Declare a proxy to reference the hub
var chat = $.connection.chatHub;

// Create a function that the hub can call to broadcast messages
chat.client.broadcastMessage = function (name, message) {
    // Add the message to the page
    $('#chat').append('<li><strong>' + name
        + '</strong>:&nbsp;&nbsp;' + message + '</li>');
};

// Get the user name and store it to prepend to messages
var username = prompt('Enter your name:');

// Set initial focus to message input box
$('#message').focus();

// Start the connection
$.connection.hub.start().done(function () {
    $('#btnSendMessage').click(function () {
        // Call the Send method on the hub
        chat.server.send(username, $('#message').val());

        // Clear text box and reset focus for next comment
        $('#message').val('').focus();
    });
});
```

Run the Application

- The following screen shot shows the chat application running in three browser instances, all of which are updated when one instance sends a message:



- **Tom**: Let's talk SignalR
- **Anand**: What is it?
- **Tom**: SignalR is an ASP.NET server library for adding a real-time functionality to a web application
- **Susan**: Just recently started with SignalR