Today's Agenda

REST

- What is REST?
- The Six Constraints

ASP.NET Web API

- Web API controllers
- Data Transfer Objects (DTOs)

- Single Page Application (SPA)

Exercises

Interdisciplinary project

1. Kick-off
   Monday, 28. November at 8:30 – Presentation.

2. Group Size
   3-4 – Email the names to jear@eaaa.dk

3. Submissison & presentations
   Wednesday, 14. December at 12:00.

4. Feedback session
   Monday and Tuesday, 19. and 20. December.

What is REST?

REST Transfer

- REST (REpresentational State Transfer) is architectural principles by which you can design Web services.
- Introduced in 2000 by Roy Fielding at the University of California, Irvine, in his academic dissertation, "Architectural Styles and the Design of Network-based Software Architectures".
- Today REST has mostly displaced SOAP- and WSDL-based interface design because it's simpler to us.

REST overview

- Resource-based
- Representations
- Six Constraints
  - Uniform interface
  - Stateless
  - Client-Server
  - Cacheable
  - Layered system
  - Code on demand (optional)

Resource based

- Things vs. action
- Nouns vs. verbs
- Versus SOAP
- Identified by URIs
  - Multiple URIs might point to the same resource
- Separate from their representation(s)

Representations

- Information of how resources get manipulated (i.e. update, delete)
- Represents (part of) the resource state
  - Transferred between client and server
- Typically as JSON or XML
- Example:
  - Resource: i.e. person, movie, book, author etc.
  - Service: Contact Info (GET)
  - Representation of a resource state
    - name, address, phone number, email
    - JSON or XML format

The Six Constraints

1. Uniform Interface

- Defines the interface between client and server
- Simplifies and decouples the architecture between client and server
- For most web-based RESTfull services it means:
  - HTTP verbs (GET, PUT, POST, DELETE)

200	DELETE	28	localhost:9264	json
204	PUT	28	localhost:9264	xml
200	GET	28	localhost:9264	json
201	POST	/	localhost:9264	json

  - URI (resource name, i.e. `http://localhost:9264/bookservice/1`)
  - HTTP response (status, body)

2. Stateless

- Server contain no client state
- Each request contains enough context to process the message
- Any state is held on the client

How to secure an ASP.NET Web API web service:
Secure a Web API with Individual Accounts and Local Login in ASP.NET Web API 2.2

3. Client-Server

- A disconnected system
- Separation of concerns
- The uniform interface (HTTP-protocol) is the link between the two

4. Cacheable

- Server responses (representations) are cacheable
  - Implicitly
  - Explicitly
  - Negotiated

5. Layered system

- A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary server
- Improves scalability by enabling load balancing

6. Code on demand (optional)

- Server can temporarily extend the server
- Transfer logic to the server
- Client executes logic
- For example
  - Java Applet
  - JavaScript

Summery

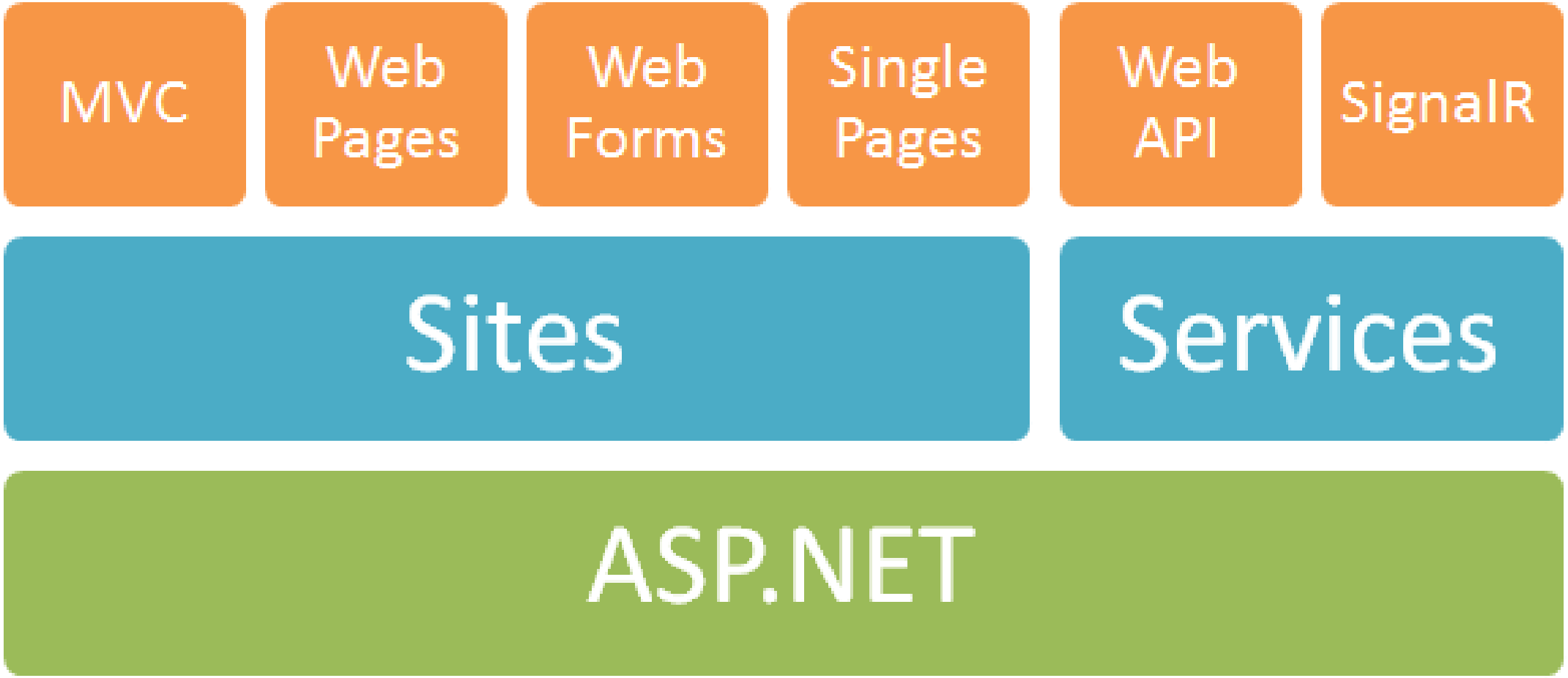
- Violation any of the first 5 constraints means the service is not strictly RESTful
- Compliance with REST constraints allows:
  - Scalability
  - Simplicity
  - Modifiability
  - Visibility
  - Portability
  - Reliability

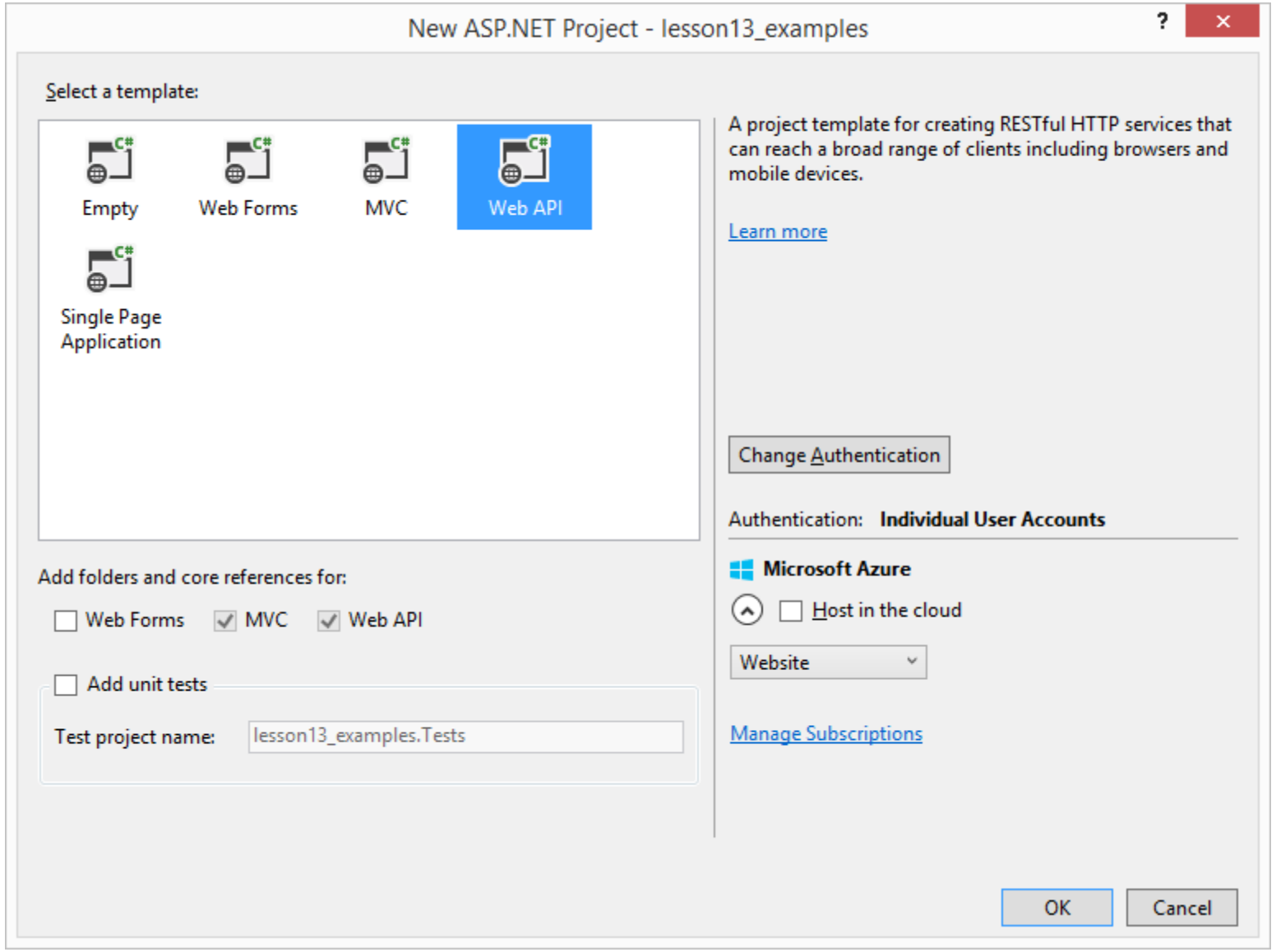
When to use Web API?

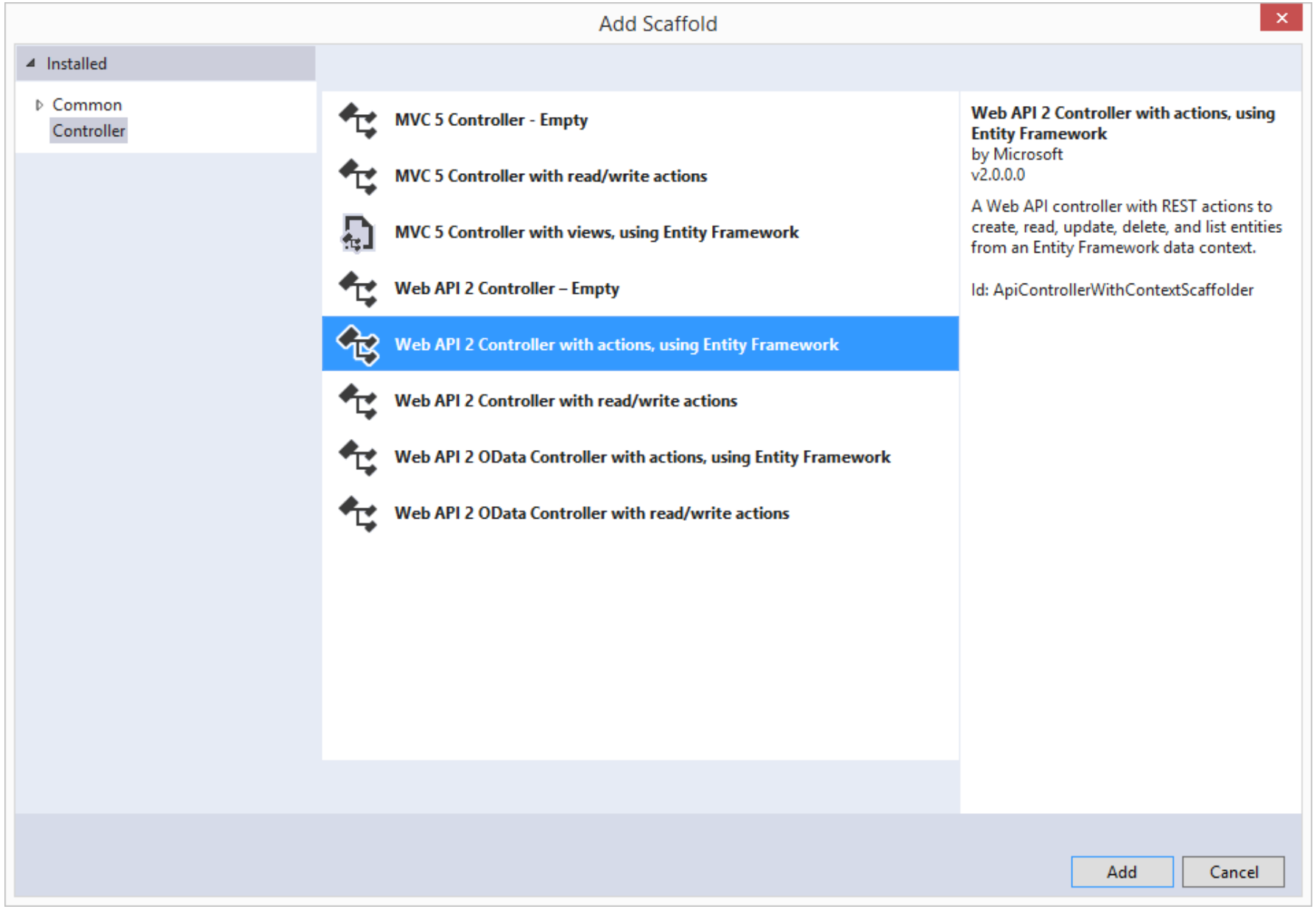
- Ajax application – Ajax calls to Web API services
- Single Page Application (SPA)
- Service for external websites

ASP.NET Web API

Web API is not part of core ASP.NET MVC




New ASP.NET Project - lesson13_examples
Select a template:
Empty
Web Forms
MVC
Web API
Single Page Application
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.
Learn more
Change Authentication
Authentication: Individual User Accounts
Microsoft Azure
Host in the cloud
Website
Manage Subscriptions
Add folders and core references for:
Web Forms
MVC
Web API
Add unit tests
Test project name:
lesson13_examples.Tests
OK
Cancel


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 Controller with actions, using Entity Framework
Web API 2 OData 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
Add
Cancel

App_Start/WebApiConfig.cs

```
public static class WebApiConfig {
        public static void Register(HttpConfiguration config) {

            // Web API routes
            config.MapHttpAttributeRoutes();

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

The Model

```
public class Book {
    public int Id { get; set; }
    public string Title { get; set; }
    public int Year { get; set; }
    public decimal Price { get; set; }
    public string Genre { get; set; }

    // Foreign Key
    public int AuthorId { get; set; }
    // Navigation property
    public Author Author { get; set; }
}
```

```
public class Author {

    public int Id { get; set; }

    public string Name { get; set; }

}
```

The REST API created by the Web API Scaffolding enables CRUD operations on the database

HTTP Verb	URI	Description
GET	/api/authors	Get a list of all authors
GET	/api/authors/{id}	Get the author with ID equal to {id}
PUT	/api/authors/{id}	Update the author with ID equal to {id}
POST	/api/authors	Add a new author to the database
DELETE	/api/authors/{id}	Delete a author from the database

GET	/api/books	Get a list of all books
GET	/api/books/{id}	Get the book with ID equal to {id}
PUT	/api/books/{id}	Update the book with ID equal to {id}
POST	/api/books	Add a new book to the database
DELETE	/api/books/{id}	Delete a book from the database

AuthorsController

```
public class AuthorsController : ApiController
{
    ...
}
```

HTTP Methods Naming Convention

```
/api/authors/{id}
```

- By default the controller name is given in the URI-segment
- Web API selects actions based on the HTTP method of the request (GET, POST, PUT, DELETE).
- By default, Web API looks for a case-insensitive match with the start of the controller method name.
- For example, a controller method named **`PutAuthor`** matches a HTTP PUT request.

HTTP Method	Action Method
GET	`GetAuthor`
PUT	`PutAuthor`
POST	`PostAuthor`
DELETE	`DeleteAuthor`

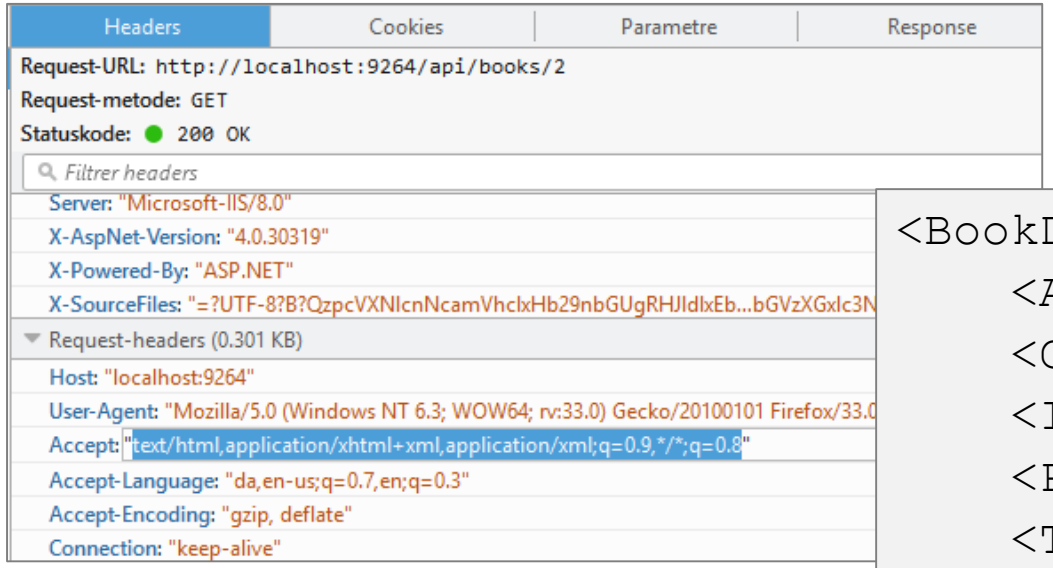
GetAuthors()

```
 // GET: api/Authors
public IQueryable<Author> GetAuthors()
{
      return dbContext.Authors;
}
```

GetAuthor(int id)

```
// GET: api/Authors/5
[ResponseType(typeof(Author))] // new to Web AP! 2.1
public IHttpActionResult GetAuthor(int id) {
      Author author = dbContext.Authors.Find(id);
      if (author == null)   {
                  return NotFound();
      }
            return Ok(author);
}
```

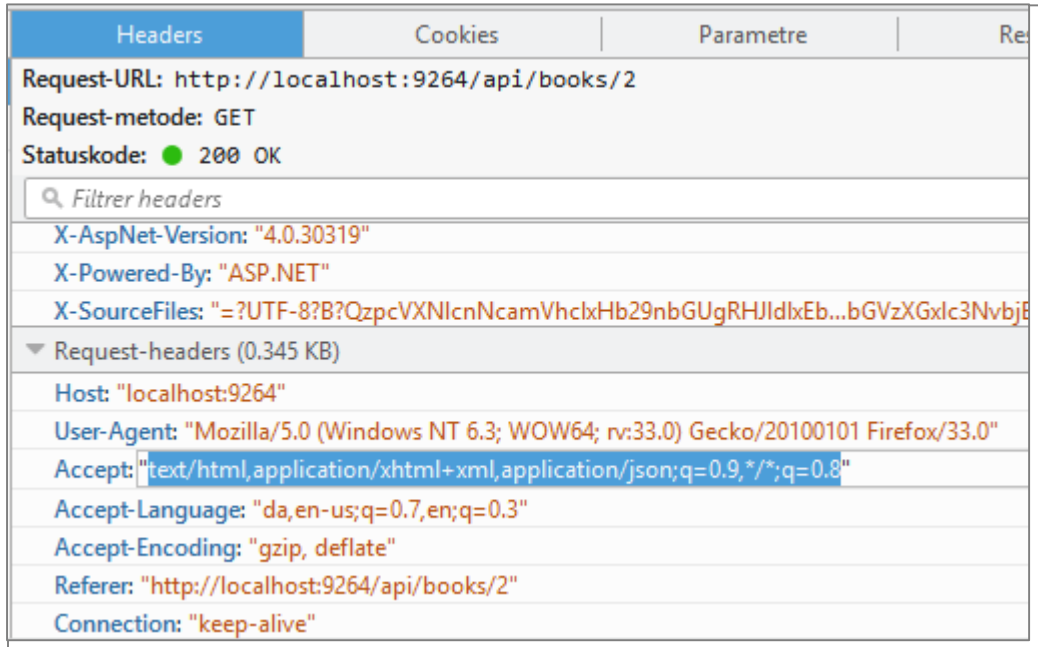
Content negotiation
& Response (representation of a resource state)

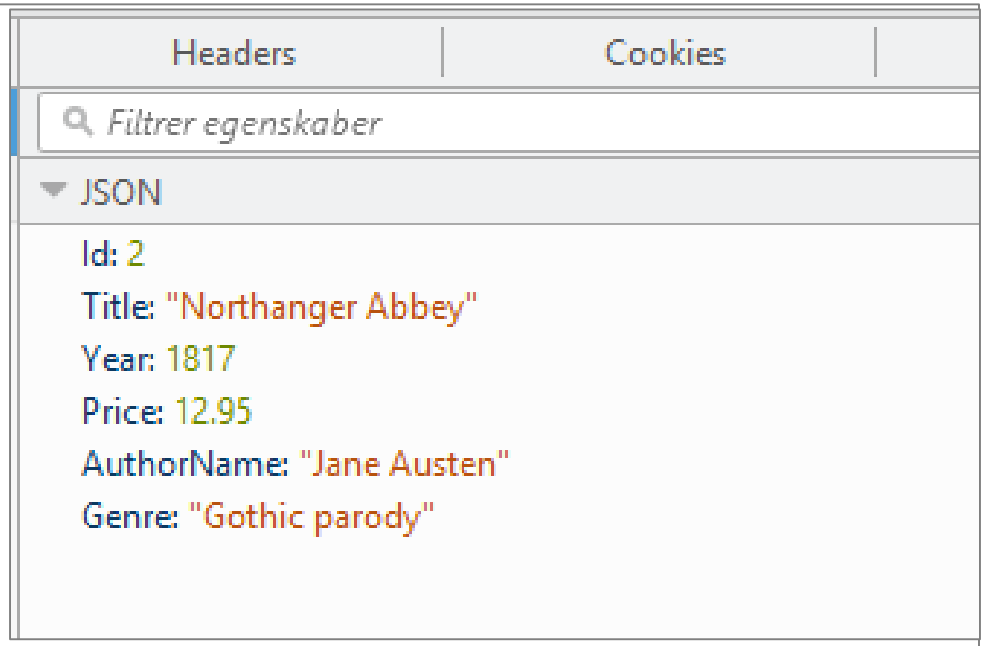



```
<BookDetailDTO>
    <AuthorName>Jane Austen</AuthorName>
    <Genre>Gothic parody</Genre>
    <Id>2</Id>
    <Price>12.95</Price>
    <Title>Northanger Abbey</Title>
    <Year>1817</Year>
</BookDetailDTO>
```

Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"

Content negotiation
& Response (representation of a resource state)





Accept: "text/html,application/xhtml+xml,application/json;q=0.9,*/*;q=0.8"

`DefaultContentNegotiator : IDefaultContentNegotiator`

- XML, JSON, BSON, and form-urlencoded data supported ny default
- You can support additional media types by writing a media formatter.

- You can determined the format you want back:

```
$.ajax({
  url: "/api/news/all",
  type: "GET",
  headers: { Accept: "application/json; charset=utf-8" }
});
```

Read more:

- http://www.asp.net/web-api/overview/formats-and-model-binding/content-negotiation
- http://www.asp.net/web-api/overview/formats-and-model-binding/media-formatters

PutAuthor(int id, Author author)

```
 // PUT: api/Authors/5
[ResponseType(typeof(void))]
public IHttpActionResult PutAuthor(int id, Author author){
      if (!ModelState.IsValid){
            return BadRequest(ModelState);
      }
      if (id != author.Id){
            return BadRequest();
      }

      dbContext.Entry(author).State = EntityState.Modified;

      ...
```

PutAuthor (cont.)

```
        try {
                dbContext.SaveChanges();
        }
        catch (DbUpdateConcurrencyException)
        {
                if (!AuthorExists(id)) {
                    return NotFound();
                } else {
                    throw;
                }
        }
        return StatusCode(HttpStatusCode.NoContent);
}
```

PostAuthor(Author author)

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

      dbContext.Authors.Add(author);
      dbContext.SaveChanges();

      return CreatedAtRoute("DefaultApi", new { id = author.Id },
author);
}
```

DeleteAuthor(int id)

```
// DELETE: api/Authors/5
[ResponseType(typeof(Author))]
public IHttpActionResult DeleteAuthor(int id){

      Author author = dbContext.Authors.Find(id);
      if (author == null) {
            return NotFound();
      }
      dbContext.Authors.Remove(author);
      dbContext.SaveChanges();
      return Ok(author);
}
```

Override HTTP Methods Naming Convention

- The following example maps the `CreateBook` method to HTTP POST requests:

```
[Route("api/books")]
[HttpPost]
public HttpResponseMessage CreateBook(Book book) { ... }
```

Data Transfer Objects (DTOs)

Problem: If you add a navigation property to the Author class it creates a circular object graph and when JSON or XML formatter tries to serialize the graph, it will throw an exception

Serialize error message

```
{
  "Message": "An error has occurred.",
  "ExceptionMessage": "The 'ObjectContent`1' type failed to serialize the response body
for content type 'application/json; charset=utf-8'.",
  "ExceptionType": "System.InvalidOperationException",
  "StackTrace": null,
  "InnerException": {
    "Message": "An error has occurred.",
    "ExceptionMessage": "Self referencing loop detected with type
'BookService.Models.Book'.
        Path '[0].Author.Books'.",
    "ExceptionType": "Newtonsoft.Json.JsonSerializationException",
    "StackTrace": "...”
     }
}
```

Solution

1. Configure the JSON and XML formatters to handle graph cycles
   See: Handling Circular Object References
   http://www.asp.net/web-api/overview/formats-and-model-binding/json-and-xml-serialization#handling_circular_object_references
2. Use Data Transfer Objects (DTOs)

Why use Data Transfer Objects (DTOs)?

- Remove circular references (as in our case).
- Hide particular properties that clients are not supposed to view.
- Omit some properties in order to reduce payload size.
- Flatten object graphs that contain nested objects, to make them more convenient for clients.
- Avoid “over-posting” vulnerabilities
- Decouple your service layer from your database layer.

BookDTO

```
namespace BookService.Models
{
    public class BookDTO
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string AuthorName { get; set; }
    }
}
```

BookDetailsDTO

```
namespace BookService.Models
{
    public class BookDetailDTO
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public int Year { get; set; }
        public decimal Price { get; set; }
        public string AuthorName { get; set; }
        public string Genre { get; set; }
    }
}
```

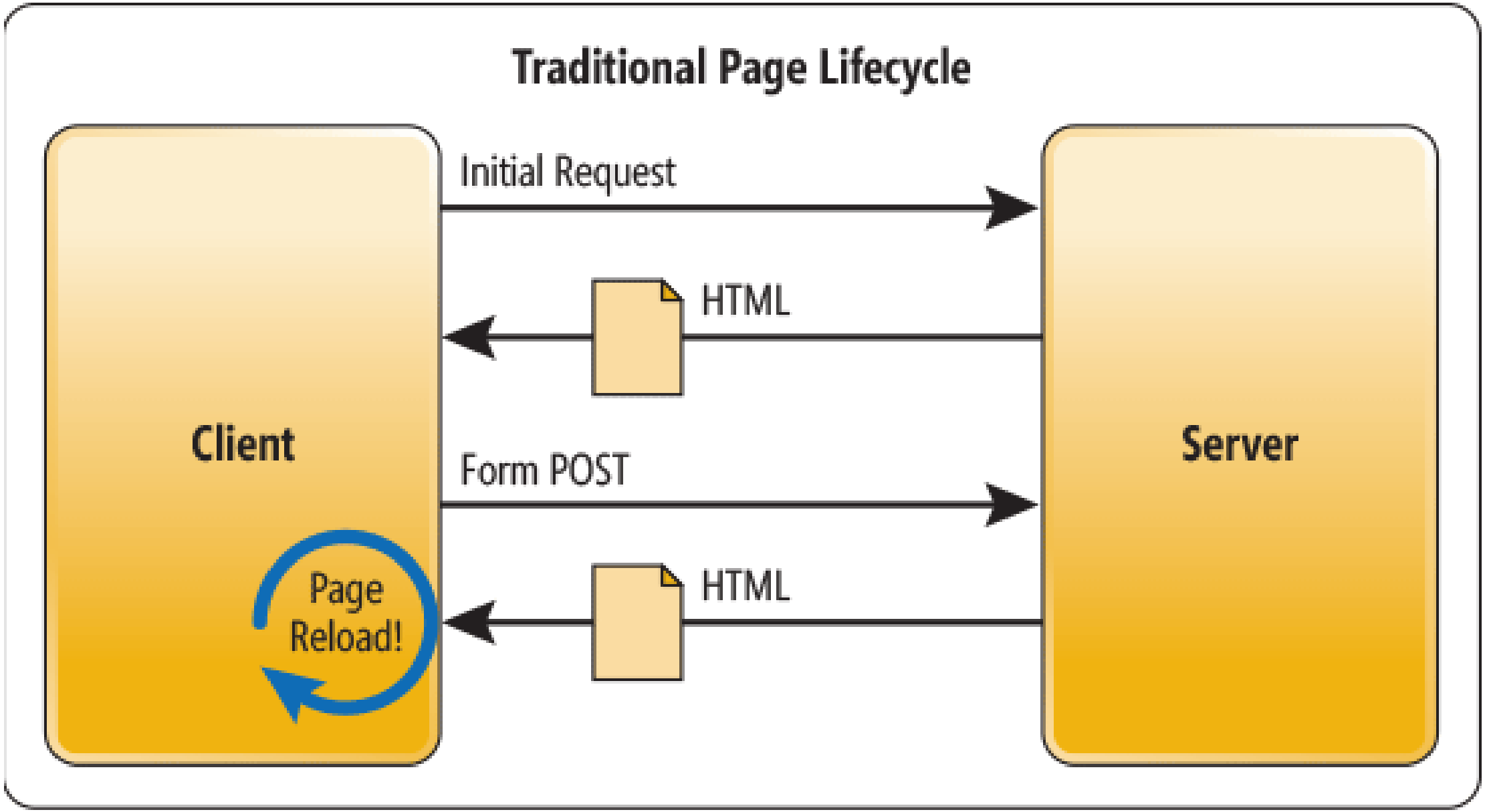
LINQ **Select** statement converts Book entities into DTO

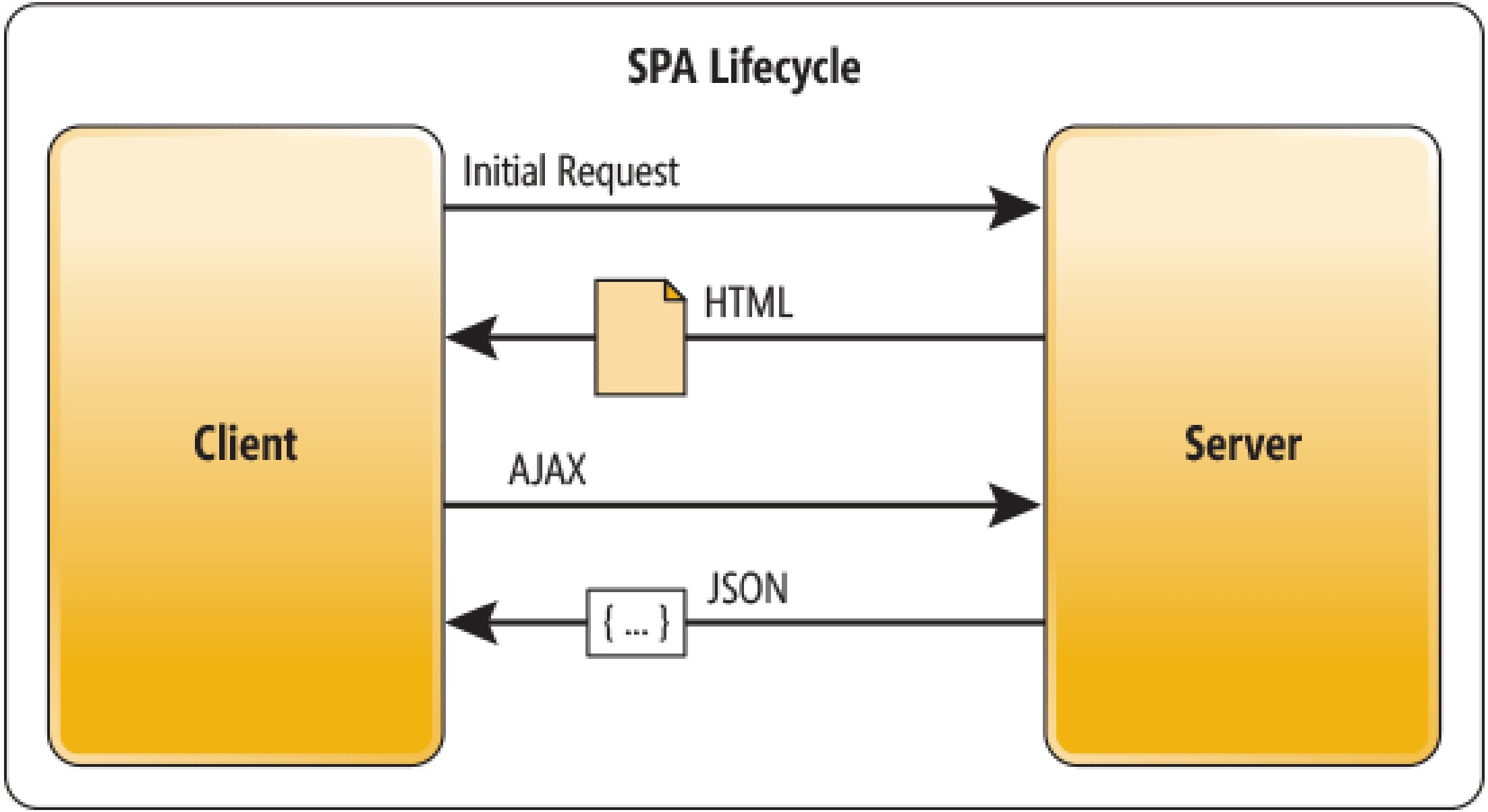
```
// GET api/Books
public IQueryable<BookDTO> GetBooks()
{
    var books = from b in db.Books
                select new BookDTO()
                {
                    Id = b.Id,
                    Title = b.Title,
                    AuthorName = b.Author.Name
                };

    return books;
}
```

BookDetailDTO

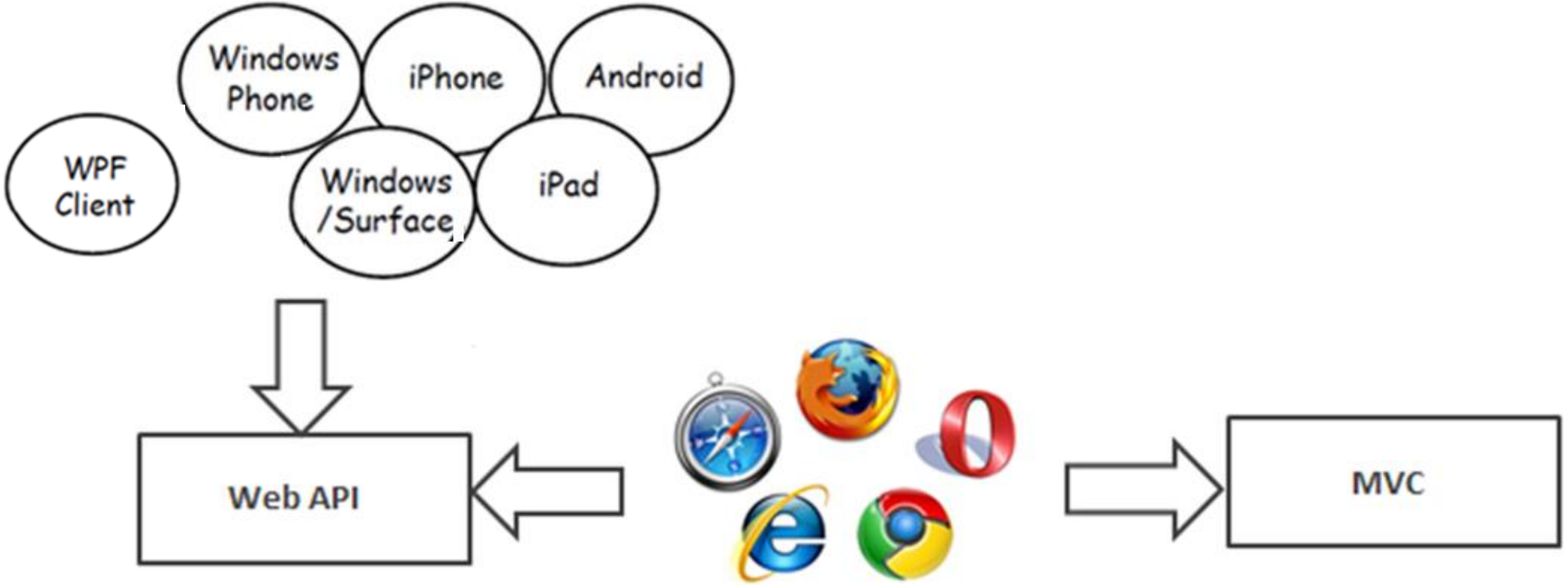
```
// GET api/Books/5
[ResponseType(typeof(BookDetailDTO))]
public async Task<IHttpActionResult> GetBook(int id)
{
    var book = await db.Books.Include(b => b.Author).Select(b =>
        new BookDetailDTO()
        {
            Id = b.Id,
            Title = b.Title,
            Year = b.Year,
            Price = b.Price,
            AuthorName = b.Author.Name,
            Genre = b.Genre
        }).SingleOrDefaultAsync(b => b.Id == id);
    if (book == null)
    {
        return NotFound();
    }

    return Ok(book);
}
```

Single Page Application


Traditional Page Lifecycle
Initial Request
HTML
Client
Server
Form POST
Page Reload!
HTML

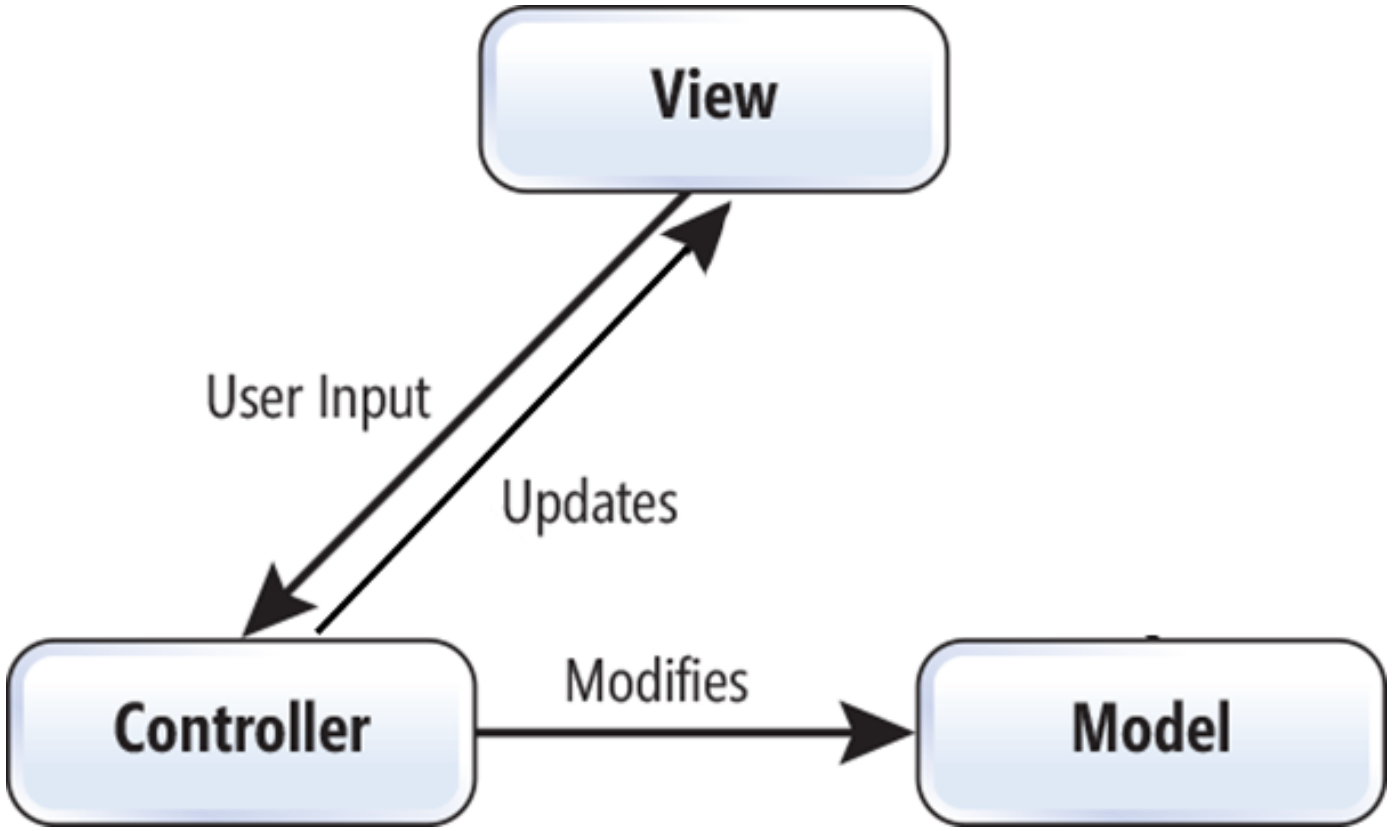

SPA Lifecycle
Client
Server
Initial Request
HTML
AJAX
JSON
{ ... }

ASP.NET MVC and ASP.NET Web API



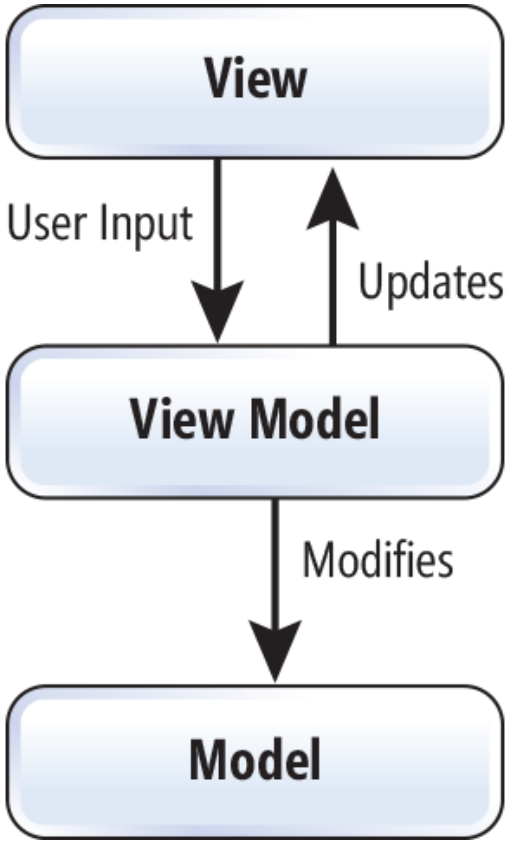
The MVC and MVVM Patterns

The MVC Pattern



- The view displays the model.
- The controller receives user input and updates the model.
- The model represents the domain data and business logic.

A variant of MVC is the MVVM pattern – used by Knockout



- The view displays the view model and sends user input to the view model.
- The view model is an abstract representation of the view. In a JavaScript MVVM framework, the view is markup and the view model is code.
- The model still represents the domain data.

Read more: Understanding MVVM – A Guide For JavaScript Developer
http://addyosmani.com/blog/understanding-mvvm-a-guide-for-javascript-developers/

JavaScript MV* Frameworks

Differences between ASP.NET MVC and Web API?

- Write down 3 essential differences

Differences between ASP.NET MVC and Web API

	ASP.NET MVC	ASP.NET WEB API
Client Side Data	Views (HTML + data + JavaScript)	Only data
Number of Applications	One application	Many (1..*)
Mapping	Each request is mapped to `controller/action` (default)	Request mapped to a controller + an action method based on HTTP verbs (i.e. PUT, GET, POST, DELETE)
Client/Server Load	Server based application logic ("Fat server")	Supports applications with client based application logic ("Thin server/Fat client")
Data formats	Fixed format	Based on content negotiation determined by the HTTP Accept header
Namespace	`System.Web.Mvc`	`System.Web.Http`

ASP.NET Core

- ASP.NET MVC Core combines
  - ASP.NET MVC
  - ASP.NET Web API into one singular framework

Evaluation

- Three things you liked most about the course
- Three things you didn't like so much
- Suggestions for improvement?
- Link
  - https://docs.google.com/forms/d/e/1FAIpQLScSRONPnbawHh-rS-rxjvWW6kcdCh3aKaY4NLxfiolWpjqzNQ/viewform
  - http://tinyurl.com/zh6ksfj

Exercises