AJAX
and
WebAPI

AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

Agenda

- AJAX
- Clint side of AJAX
- Server side of AJAX
- Web.API
- REST

AJAX
Asynchronous Javascript And Xml

Or Json

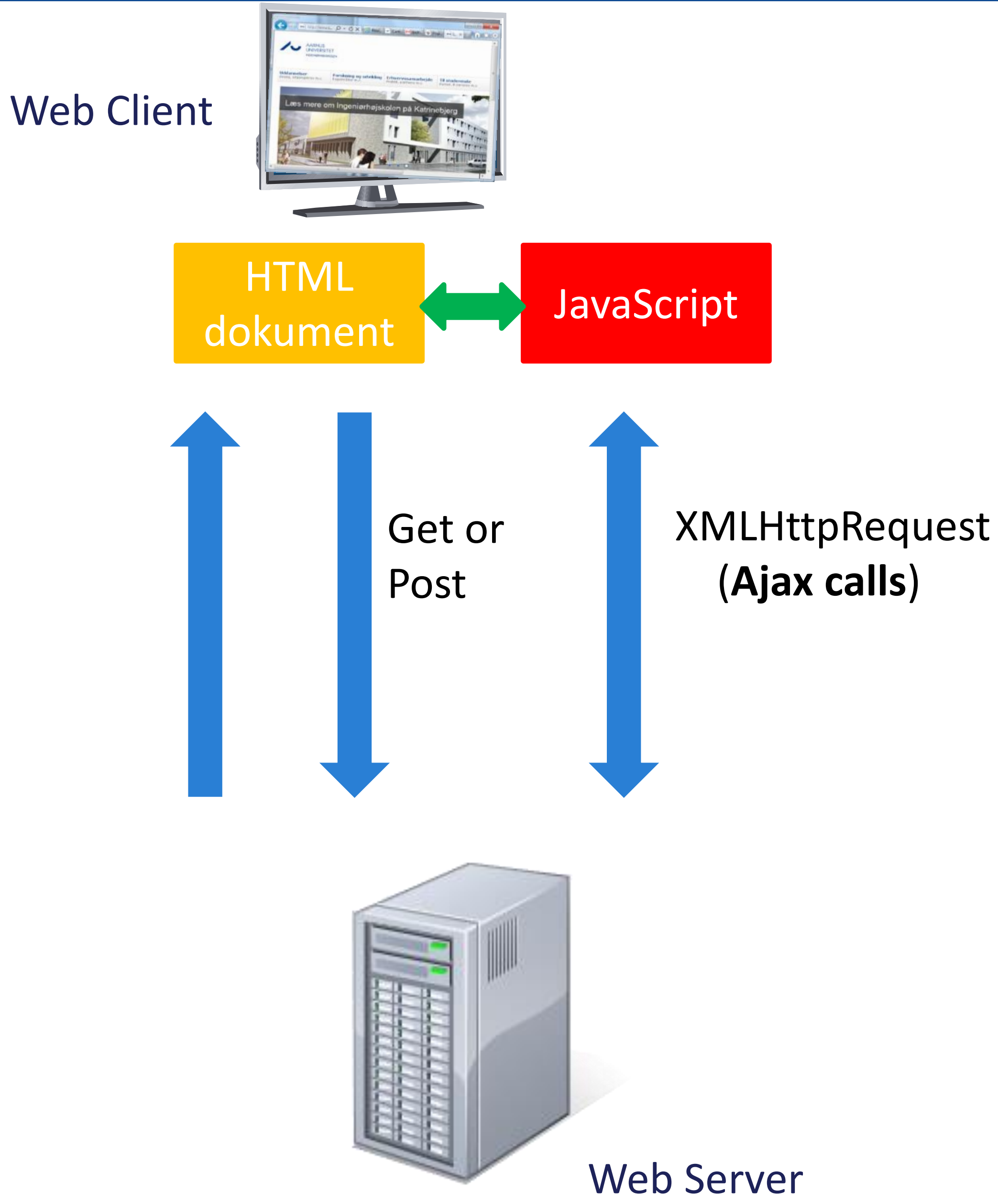
Background

- In the 1990s, most web sites were based on complete HTML pages:
  - each user action required that the page be re-loaded from the server (or a new page loaded)
  - This process is inefficient: all page content disappears then reappears
  - Each time a page is reloaded due to a partial change, all of the content must be re-sent instead of only the changed information
  - This can place additional load on the server and use excessive bandwidth

Ajax

- Is a group of interrelated web development techniques used on the client-side to create asynchronous web applications
- With Ajax:
  - **web applications can send data to, and retrieve data from, a server asynchronously** (in the background)
- Data can be retrieved using the **XMLHttpRequest** object
  - Despite the name, the use of XML is not required
    - **JSON is often used instead**
- ***Ajax is not a single technology, but a group of technologies used in combination***

AJAX Communication



CLINT SIDE OF AJAX

HTTP request

- To make an HTTP request you create a new XMLHttpRequest object:

```
var request = new XMLHttpRequest();
request.open("GET", "files/data.txt", false);
request.send(null);
alert(request.responseText);
```

Synchronous XMLHttpRequest is deprecated

- The **open** method is used to configure the request
- The **send** method perform the actual request to the server
  - When the request is a POST request, the data to be sent to the server can be passed to this method as a string
  - For GET requests just pass null
- The **responseText** property contains the content of the retrieved document (after the request has been made)
- The headers that the server sent back can be inspected with the getResponseHeader and getAllResponseHeaders functions

Asynchronous HTTP Request

- When the third argument to open is true, the request is set to be '**asynchronous**'
  - This means that send will return right away, while the request happens in the background

```
var request = new XMLHttpRequest();}
request.open("GET", "files/data.txt", true);
request.send(null);
request.onreadystatechange = function() {
  if (request.readyState == 4)
    alert(request.responseText.length);
};
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

XMLHttpRequest - load Event

- Use of the load event is often a better solution

```
var req = new XMLHttpRequest();
req.addEventListener("load", transferComplete);
req.addEventListener("error", transferFailed);
req.open("GET", "files/data.txt", true);
req.send(null);

function transferFailed(evt) {
 alert("An error occurred while transferring the file.");
 }

function transferComplete(evt) {
 console.log("The transfer is complete.", req.status);
}
```

Working With XML

- When the file retrieved by the request object is an XML document, the request's **`responseXML`** property will hold a representation of this document
- Such XML documents can be used to exchange structured information with the server
- Their form — tags contained inside other tags — is often very suitable to store things that would be tricky to represent as simple flat text
- The DOM interface is rather clumsy for extracting information though, and XML documents are notoriously wordy

```
var catalog = request.responseXML.documentElement;
alert(catalog.childNodes.length);
```

Working With JSON

- A JSON document is a file containing a single JavaScript object or array
  - which in turn contains any number of other objects, arrays, strings, numbers, booleans, or null values

```
request.open("GET", "files/fruit.json", true);
request.addEventListener("load", function(){
  console.log(request.responseText);
});
request.send(null);
```

```
{banana: "yellow", lemon: "yellow",
cherry: "red"}
```

Working With JSON

- A string that contains JSON data can be converted to a normal JavaScript value by using the JSON.parse function

```
var req = new XMLHttpRequest();
req.open("GET", "example/fruit.json", false);
req.send(null);
console.log(JSON.parse(req.responseText));
```

```
{banana: "yellow", lemon: "yellow",
cherry: "red"}
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

SERVER SIDE OF WEB.API

ASP.NET web.api

- If you use ASP.NET MVC Core then the server side of AJAX is very simple

- The AJAX request will hit a controller just like an ordinary web-request
  - **The difference is in what the controller returns!**

A Controller Returning Html

- A normal view will include all the shared “chrome” from _Layout.cshtml. We don’t want that so we **return a PartialView**
  - This ensures that the surrounding chrome that’s inside the layout page is not included in the markup returned from our controller

Index.cshtml

```
<a href="/Home/PrivacyPolicy“
   id="privacyLink">
   Show the privacy policy</a>

<div id="privacy"></div>
```

AjaxDemo.js

```
$('#privacyLink').click(function (event) {
    event.preventDefault();
    var url = $(this).attr('href');
    $('#privacy').load(url);
```

```
// In the controller class
public ActionResult PrivacyPolicy()
{
    return PartialView();
}
```

PrivacyPolicy.cshtml

```
<h2>Our Commitment to Privacy</h2>
<p>
    Your privacy is important to us.
    Bla Bla Bla
</p>
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Progressive Enhancement

- Progressive enhancement means that we begin with basic functionality (in this case, a simple hyperlink) and then layer additional behavior on top (our Ajax functionality)
- This way, if the user doesn't have JavaScript enabled in their browser, the link will gracefully degrade to its original behavior and instead send the user to the privacy policy page without using Ajax
- We can check to see whether the action has been requested via Ajax or not:

```
// In the controller class
public ActionResult PrivacyPolicy()
{
    if (Request.IsAjaxRequest())
    {
        return PartialView();
    }
    return View();
}
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

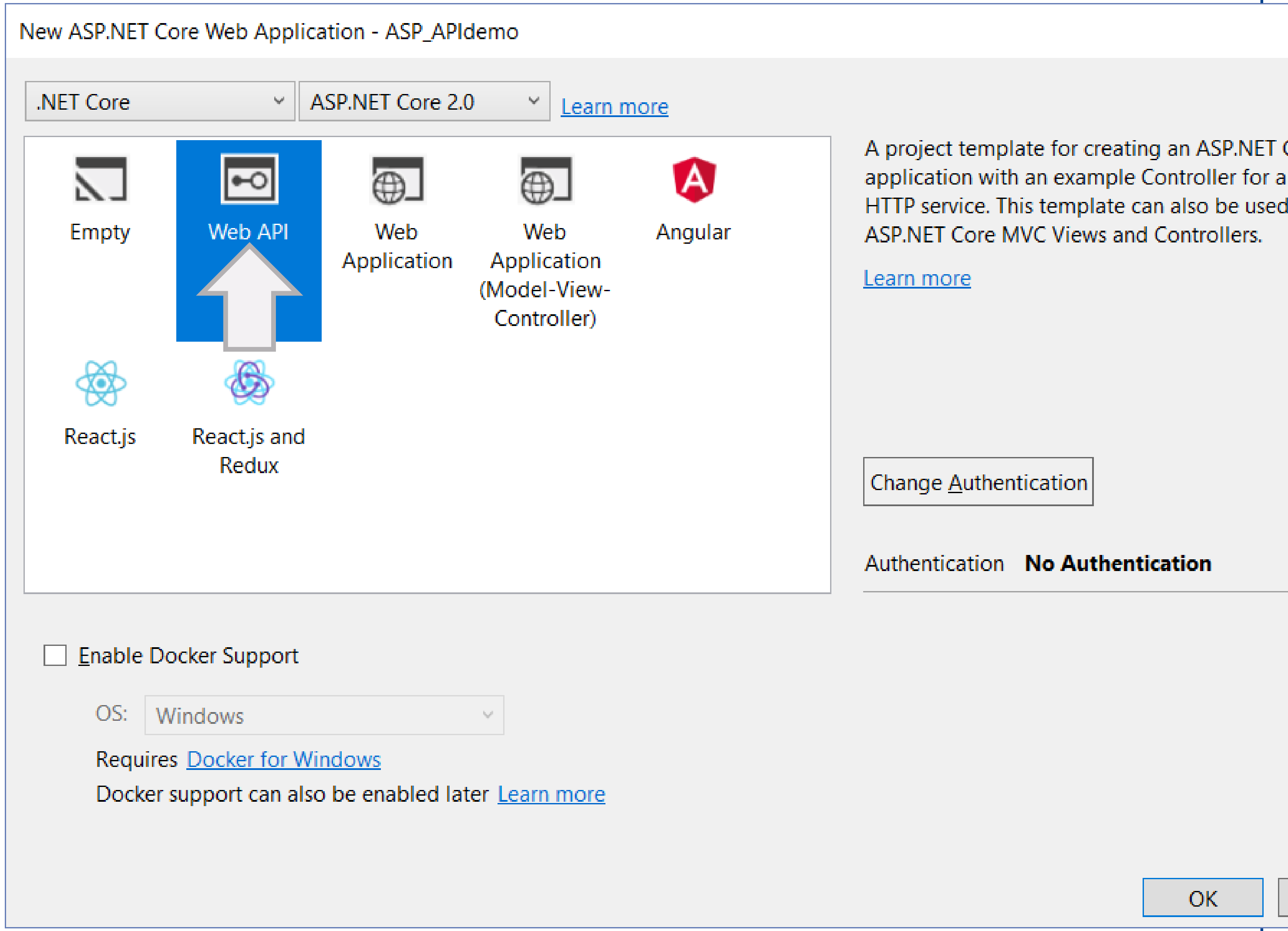
WEB.API

Why WebAPI?

- HTTP is not just for serving up web pages
- It is also a powerful platform for building APIs that expose services and data
- **HTTP services can reach a broad range of clients, including browsers, mobile devices, and traditional desktop applications**

Create a WebAPI project

- Use Visual Studio template



New ASP.NET Core Web Application - ASP_APIdemo

.NET Core | ASP.NET Core 2.0 | Learn more

A project template for creating an ASP.NET C
application with an example Controller for a
HTTP service. This template can also be used
ASP.NET Core MVC Views and Controllers.

Learn more

Authentication **No Authentication**

☐ Enable Docker Support

OS: Windows

Requires Docker for Windows

Docker support can also be enabled later Learn more

The generated controller

Uses attributed routing

http verb
(web method)

```
namespace ASPWebApi.Controllers {
    [Route("api/[controller]")]
    public class ValuesController : Controller {

        [HttpGet] // GET api/values
        public IEnumerable<string> Get() {
            return new string[] { "value1", "value2" };
        }

        [HttpGet("{id}")] // GET api/values/5
        public string Get(int id) {
            return "value";
        }

        [HttpPost] // POST api/values
        public void Post([FromBody]string value) {
        }

        [HttpPut("{id}")] // PUT api/values/5
        public void Put(int id, [FromBody]string value) {
        }

        [HttpDelete("{id}")] // DELETE api/values/5
        public void Delete(int id) {
} } }
```

What is an API Controller

- An *API controller* is an MVC controller that is responsible for providing access to the data in an application without encapsulating it in HTML

```
[HttpGet("{id}")] // GET api/values/5
public string Get(int id) {
    return "value";
}
```

The Appointment controller

MS style

```
[Route("api/appointment")]
public class AppointmentController : Controller
{
    private IRepository repository;
    public AppointmentController(IRepository repo)
    {
        repository = repo;
    }

    [HttpGet]
    public IEnumerable<Reservation> Get() {
        return repository.Reservations;
    }

    [HttpGet("{id}", Name = "GetAppointment")]
    public IActionResult Get(int id) {
        var item = repository[id];
        if (item == null)
        {
            return NotFound();
        }
        return new ObjectResult(item);
    }
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Web.API in ASP – V4

- You can create a Web.API project in VS or you can just add a Web.API controller in one of the other web-projects

- In Web API, a *controller* is an object that handles HTTP requests
  - Web API controllers are similar to MVC controllers, but inherit the **ApiController** class instead of the **Controller** class

```
public class ProductsController : ApiController
{
    public IEnumerable<Product> GetAllProducts() {
        return products;
    }
public IHttpActionResult GetProduct(int id) {
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Content Formatting

Default Content Policy:

1. If the action method **returns a string** , the string is sent unmodified to the client, and the Content-Type header of the response is set to **text/plain**
2. **For all other data types**, including other simple types such as int , the data is formatted as JSON, and the Content-Type header of the response is set to **application/json**

```
[HttpGet("string")]
public string GetString() =>
    "This is a string";
```

```
[HttpGet("object/{format?}")]
public Reservation GetObject() =>
    new Reservation {
            ReservationId = 100,
            ClientName = "Joe",
            Location = "Board Room"
    };
```

Content Negotiation

- Most clients will include an Accept header in a request, which specifies the set of formats that they are willing to receive in the response, expressed as a set of MIME types
- Here is the Accept header that Google Chrome sends in requests:

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

- This means that the Accept header sent by Chrome provides the server with the following information:
  1. Chrome prefers to receive HTML or XHTML data or WEBP images
  2. If those formats are not available, then the next most preferred format is XML
  3. If none of the preferred formats is available, then Chrome will accept any format

Enabling XML Formatting

- Add the XML Formatting Package in project.json

```
"dependencies": {
  . . .
  "Microsoft.AspNetCore.Mvc.Formatters.Xml": "1.0.0"
```

- Enable XML formatting in the Startup.cs file

```
public void ConfigureServices(IServiceCollection services) {
    services.AddSingleton<IRepository, MemoryRepository>();
    services.AddMvc()
            .AddXmlDataContractSerializerFormatters();
}
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Override the content negotiation

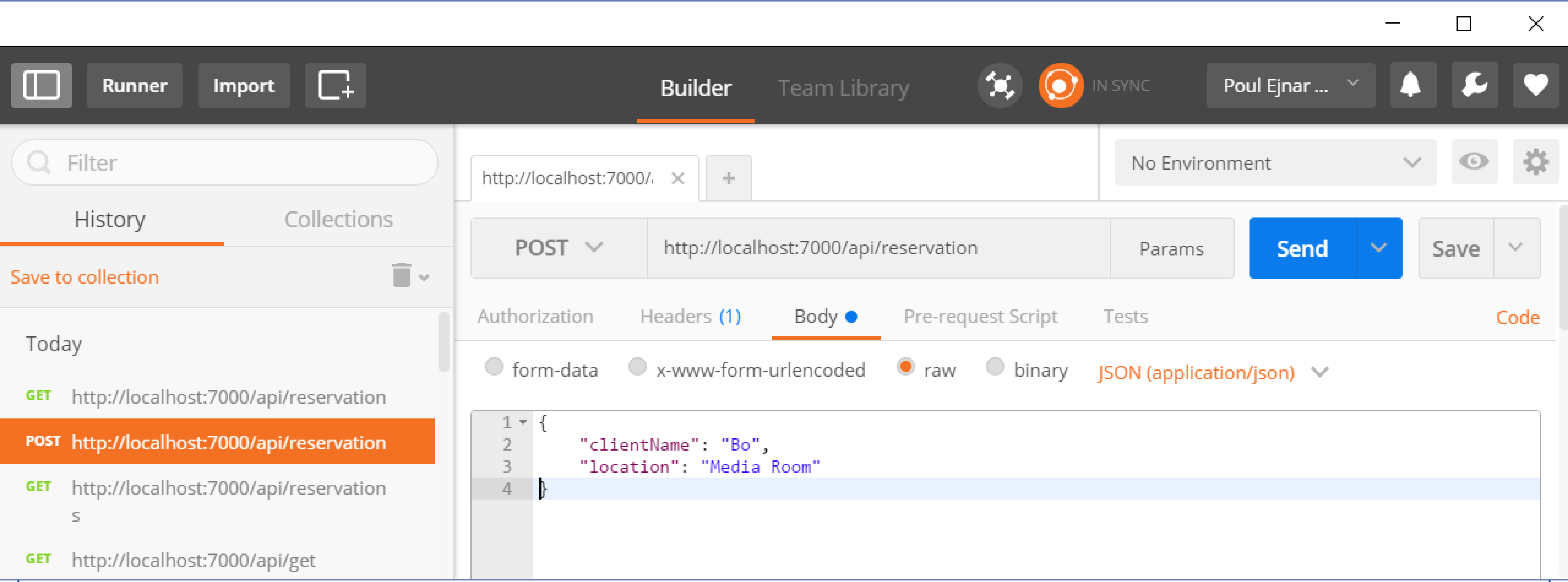
- You can override the content negotiation system and specify a data format directly on an action method by applying the Produces attribute

```
[Produces("application/json")]
public Reservation GetObject() => new Reservation {
            ReservationId = 100,
            ClientName = "Joe",
            Location = "Board Room"
        };
```

Testing an API Controller

- Many choices:
  - **Postman (Chrome extension)**
  - Curl (https://curl.haxx.se/)
  - Fiddler (www.telerik.com/fiddler)
  - PowerShell
  - ?

Using Postman



GET http://localhost:7000/api/reservation
POST http://localhost:7000/api/reservation
GET http://localhost:7000/api/reservations
GET http://localhost:7000/api/get

No Environment

POST http://localhost:7000/api/reservation

form-data x-www-form-urlencoded [✓ raw] binary JSON (application/json)

```json
{
    "clientName": "Bo",
    "location": "Media Room"
}
```

CORS

- Cross-origin resource sharing
- XMLHttpRequest (and Embedded web fonts) requests have traditionally been limited to accessing the same domain as the parent web page
- CORS defines a way in which a browser and server can interact to safely determine whether or not to allow the cross-origin request
- Client – prerequest:

```
Origin: http://www.foo.com
```

- Server – response:

```
Access-Control-Allow-Origin: http://www.foo.com
```

  - or

```
Access-Control-Allow-Origin: *
```

CORS How-To

- To enable CORS for WebAPI add the Microsoft.AspNetCore.Cors package to your project
  - In Version 2.0 Cors is include in the default package

```
<ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
</ItemGroup>
```

- Add the CORS services in Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IRepository, MemoryRepository>();
    // Add framework services.
    services.AddCors();
    services.AddMvc();
}
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

Enabling CORS with middleware

- To enable CORS for your entire application add the CORS middleware to your request pipeline using the UseCors extension method

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
                      ILoggerFactory loggerFactory)
{
    app.UseCors(builder =>
                builder.WithOrigins("http://example.com")
                       .AllowAnyMethod()
                       .AllowAnyHeader()
                       .AllowCredentials()
                );
    app.UseMvc();
}
```

Auth

- ASP Web.API has the same possibilities for autorization as ASP.MVC
  - But no authorization is default
- But no register or login page
  - This must be done by use of XMLHttpRequest calls

REST
REPRESENTATIONAL STATE TRANSFER

REST didn't attract much attention when it was first 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," which analyzes a set of software architecture principles that use the Web as a platform for distributed computing.

Representational State Transfer (REST) has gained widespread acceptance across the Web as a simpler alternative to SOAP- and Web Services Description Language (WSDL)-based Web services.

RESTful Web Services: the Basics

- REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages

- A concrete implementation of a REST Web service follows four basic design principles:
  - **Use HTTP methods explicitly**
  - **Be stateless**
  - **Expose directory structure-like URIs**
  - **Transfer XML, JSON,** or any other valid Internet media type content (such as an image or plain text)

- REST is not a standard but an architectural style

Use HTTP Methods Explicitly

- One of the key characteristics of a RESTful Web service is the explicit use of HTTP methods in a way that follows the protocol as defined by RFC 2616
- This basic REST design principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods
- According to this mapping:
  - **To create a resource on the server, use POST**
  - **To retrieve a resource, use GET**
  - **To change the state of a resource or to update it, use PUT**
  - **To remove or delete a resource, use DELETE**

Use HTTP Methods Explicitly

API Design Principles:

- Use nouns in URIs instead of verbs.
  - In a RESTful Web service, the verbs—POST, GET, PUT, and DELETE—are already defined by the protocol
- And the Web service should not define more verbs or remote procedures, such as /adduser or /updateuser
- The body of an HTTP request should be used to transfer resource state
  - not to carry the name of a remote method to be invoked

Use HTTP Methods Explicitly

- A non-RESTful API:

```
GET /adduser?name=Robert HTTP/1.1
```

- A RESTful API:

```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/json
{name : Robert}
```

Example - Collection URI

Resource	Collection URI, such as **http://example.com/resources**
GET	**List** the URIs and perhaps other details of the collection's members.
PUT	**Replace** the entire collection with another collection.
POST	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.
DELETE	**Delete** the entire collection.

Example - Element URI

Resource	Element URI, such as **http://example.com/resources/item27**
GET	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type (typical JSON).
PUT	**Replace** the addressed member of the collection, or if it doesn't exist, **create** it.
POST	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it.
DELETE	**Delete** the addressed member of the collection.

References & Links

- http://json.org/
- Json.net
  http://james.newtonking.com/projects/json-net.aspx
- An Introduction to JavaScript Object Notation (JSON) in JavaScript and .NET
  http://msdn.microsoft.com/en-us/library/bb299886.aspx
- **ASP.NET Web API**
  https://docs.microsoft.com/da-dk/aspnet/core/tutorials/first-web-api
- Gratis bog: **ASP.NET Web API Succinctly**
- **Postman** (a Chrome extension very useful for testing Web.API)
  https://www.getpostman.com/
- REST
  http://www.ibm.com/developerworks/webservices/library/ws-restful/
  http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm