Web services
Overblik

REST

- Hvad er REST?
- Et eksempel
- De 5 karakteristikker

ASP.NET Web API

- Web API controllers
- Mapping REST til controller methods ved brug af HTTP
- Test af vores API (POSTman)
- Anvendelse af API fra klient

Hvad menes med en web service?

- En service, der kaldes over HTTP, og som (i modsætning til en webside) returnerer data, som ikke er html, men i stedet er beregnet til f.eks. at blive gemt i en database.
- Eksempel med DMI

Hvad er REST?

Rest er en bestemt slags web service. Der findes andre, f.eks. SOAP/WSDL baserede

REST

- REST (REpresentational State Transfer) er nogle architectural principles som kan bruges til at designe Web services.
- Introduceret i 2000 af Roy Fielding i hans akademiske dissertation, "Architectural Styles and the Design of Network-based Software Architectures".
- I dag har REST overtaget fra SOAP- and WSDL-based interfaces,

  fordi REST er simplere at bruge.
- REST er en god måde at buge web services på – både intern og ekstern (som til en API).

Eksempler på REST Services (readonly)

Alle metoder her returnerer JSON

Spotify :
https://beta.developer.spotify.com/documentation/web-api/

Usecases for en Web Service

- **Ajax** applikationer – Ajax kald til Web API services (mere om Ajax senere)
- **Single Page Application** (SPA)
- **Service til** external websites (API)

REST overview

- Resource-based
- Representations
- 5 REST characteristics
  - Uniform interface
  - Stateless
  - Client-Server
  - Cacheable
  - Layered system

Resource baseret

- Identificeres vha. URIs (URLs)
  - Multiple URIs kan pege på den samme ressource.
- Separat fra deres repræsentation (som typisk er json)

Repræsentationer

- Information omkring hvordan ressourcer kan manipuleres – f.eks. update, delete,
- Typisk er en ressource i JSON eller XML format.
- Eksempel:
  - Resource: person, movie, book, author etc.
  - Service: Contact Info (GET)
  - Representation of a resource state
    - name, address, phone number, email
    - JSON or XML format

REST
Karakteristikker (5 principper)

1. Uniform Interface

- Definerer et interface mellem client og server
- Simplificerer og decoupler arkitekturen mellem klient og server
- For de fleste web-based RESTfull services betyder det:
  - 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 har ikke nogen client state
- Hvert request via REST skal indeholde nok info til at klienten kan bruge data.
- Alt state i applikationen er på client siden.

3. Client-Server

- "Separation of concerns"
- Det uniforme interface (HTTP-protocol) er linket mellem client og server.
- Bemærk: Det er en Client/Server ”protocol” – så ikke peer-to-peer eller noget.

4. Cacheable

- Server responses (dvs. data) skal være cacheable
  - Implicitly
  - Explicitly
  - Negotiated

5. Lagdelt system

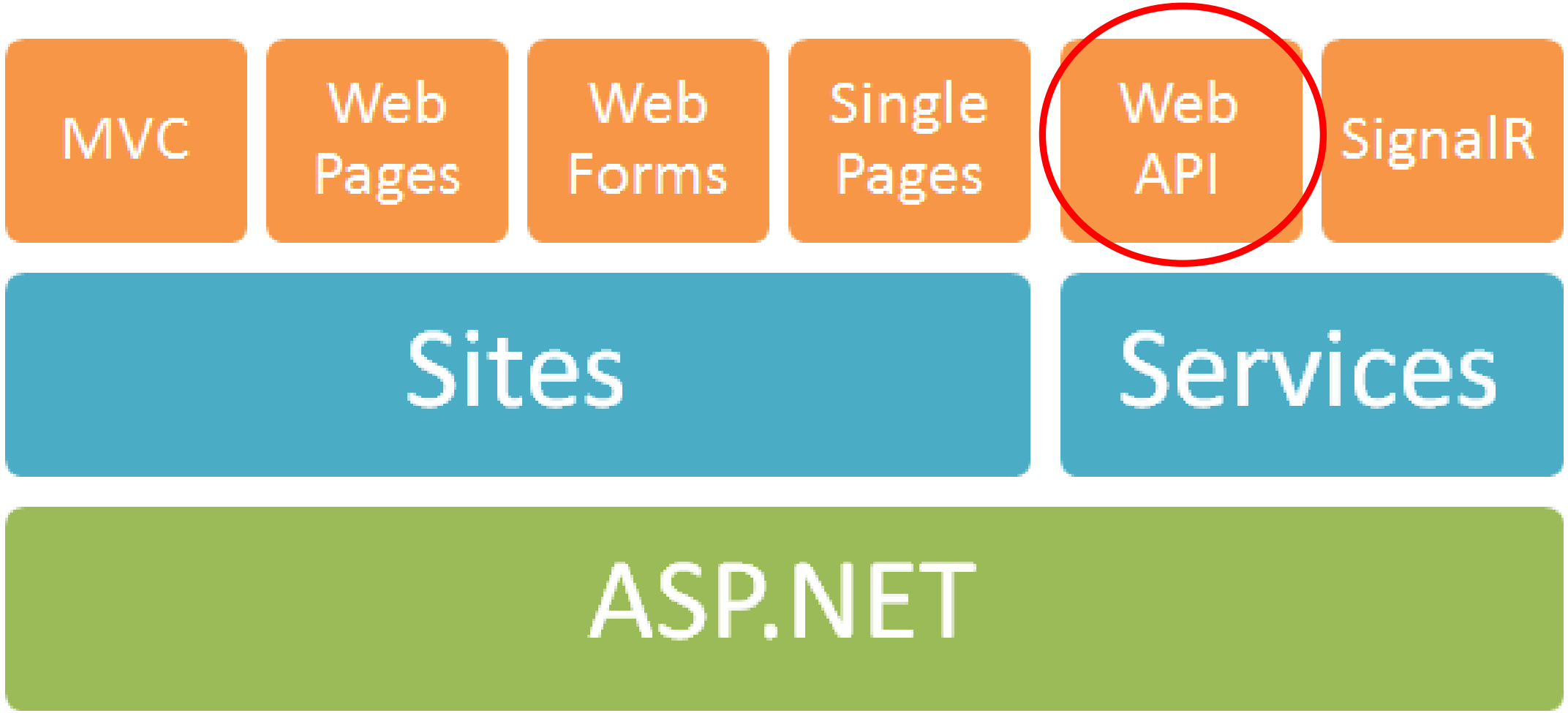
- A client kan ikke normalt skelne imellem om den er connected til ”end server” eller f.eks. til en proxy server.
- Det forbedrer scalability og performance

Opsummering

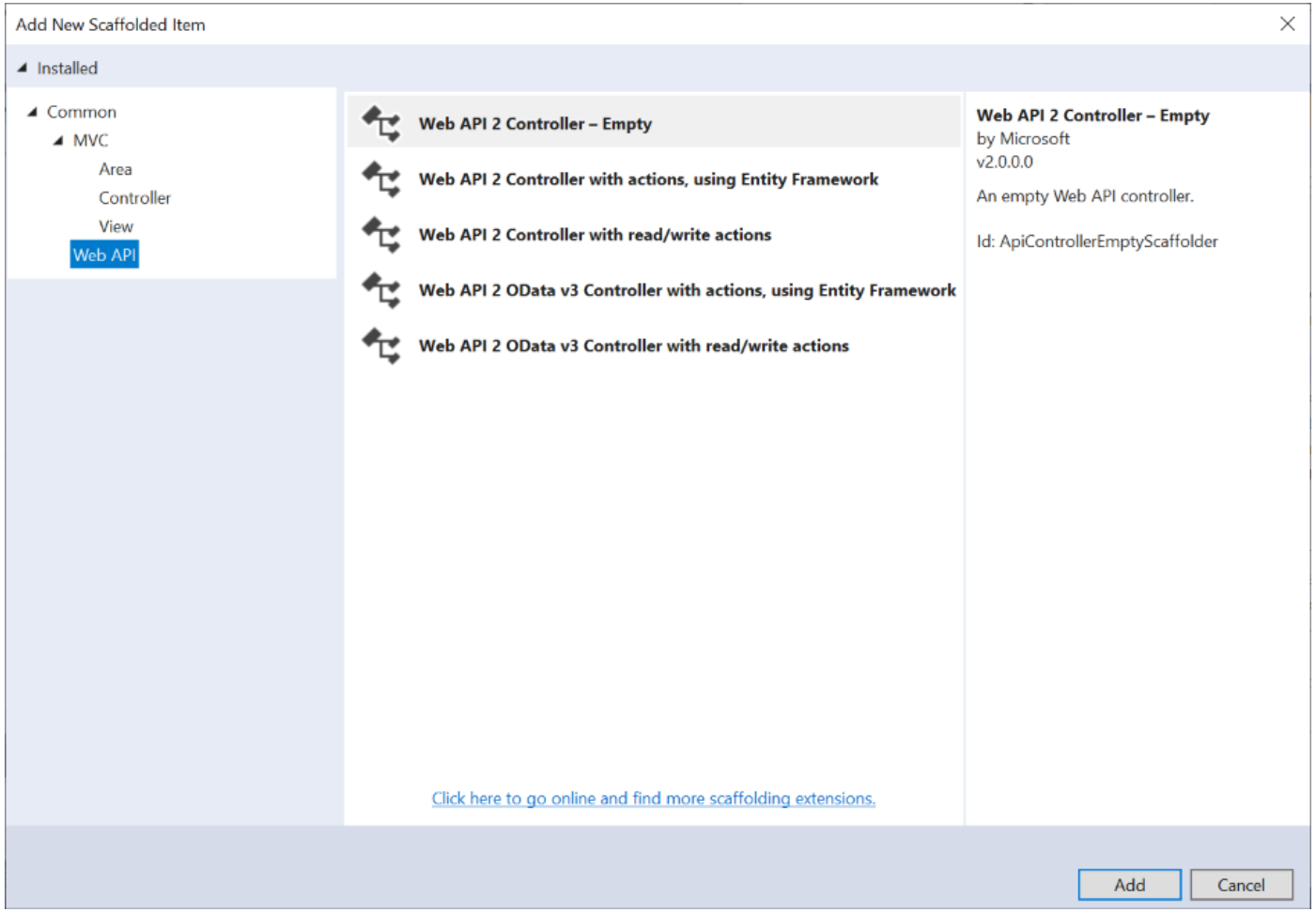
- Hvis en webservice skal være RESTfull, så skal disse 5 principper overholdes.
- Compliance med REST giver:
  - Scalability
  - Simplicity
  - Modifiability
  - Visibility
  - Portability
- Se her for detaljer: https://en.wikipedia.org/wiki/Representational_state_transfer

ASP.NET Web API
Rest i MVC

Overblik


MVC
Web Pages
Web Forms
Single Pages
Web API
SignalR
Sites
Services
ASP.NET

Vi kan tilføje en WebAPI controller – højre klik på Controller folderen og så vælg Add->Controller og så vælg Web API. Der er så nu flere muligheder som vist på screenshottet her.



Modellen i eksemplet

```
public class Student
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public string City { get; set; }
        public string Course { get; set; }
    }
```

REST API Lavet af Web Api tillader CRUD operationerne

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

StudentController (Bemærk nedarving)

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

Denne controller vil håndtere HTTP requests og returnere data.
Det er en speciel controller, men den minder meget om en almindelig controller.
HTTP metoderne mappes så til Action metoder i denne controller.

App_Start/WebApiConfig.cs
Laves automatisk – der kan man se urls til webapi.
BEMÆRK: ingen action

```
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 }
                );
        }
    }
```

HTTP Metoder Naming Convention

BEMÆRK: ingen action i URL: `/api/student/{id}`

- Per default er controller navnet angivet i URL segmentet – her student efter StudentControlller
- Web API udvælger actionmetoden baseret på HTTP method of the request (GET, POST,PUT, DELETE).
- Per default, så vil Web API se efter et match (case-insensitive) med starten af controller method name.
- F.eks. så vil en Action metode i controlleren kaldet **`PutStudent`** matche en HTTP PUT request.

HTTP Method	DefaultAction Method
GET	`GetStudent`
PUT	`PutStudent`
POST	`PostStudent`
DELETE	`DeleteStudent`

Override af HTTP Methods Naming Convention

- **Det er muligt og i mange tilfælde godt at eksplicit mappe HTTP metoderne til en specifik metoden i Controlleren.** Vi kan bruge [HttpPost], [HttpPut], [HttpGet] and [HttpDelete] attributter på vores metoder for at gøre dette.

- Det fjerner evt. stavefejls bugs i metode navnene (F.eks. deliteStudent) og gør måske også koden nemmere at læse, da man nemt eksplicit kan se hvilke HTTP kald bliver mappet til hvilke metoder.

GetAll()

```
// GET: api/student
        [HttpGet]
         public IEnumerable<Student> GetAll()
         {
             return studentRepository.GetAllStudents();
         }
```

GetStudent(int id)

```
// GET: api/student/5
[HttpGet]
public Student GetStudent(int id)

{
        return studentRepository.GetStudent(id);


}
```

PutStudent(Student student) – opdatering af eksisterende

```
 //  PUT: api/student
[HttpPut]
public Student UpdateStudent(Student student)
{
    return studentRepository.updateStudent(student);
 }
```

PostStudent(Student student) – tilføjelse af ny studerende

```
// POST: api/student

[HttpPost]
public Student AddStudent(Student student)
  {
      studentRepository.AddStudent(student);
      return student;

  }
```

DeleteStudent(int id)

```
//      DELETE:      api/student/5
        [HttpDelete]
        public Student DeleteStudent(int id)
        {
            return studentRepository.DeleteStudent(id);
        }
```

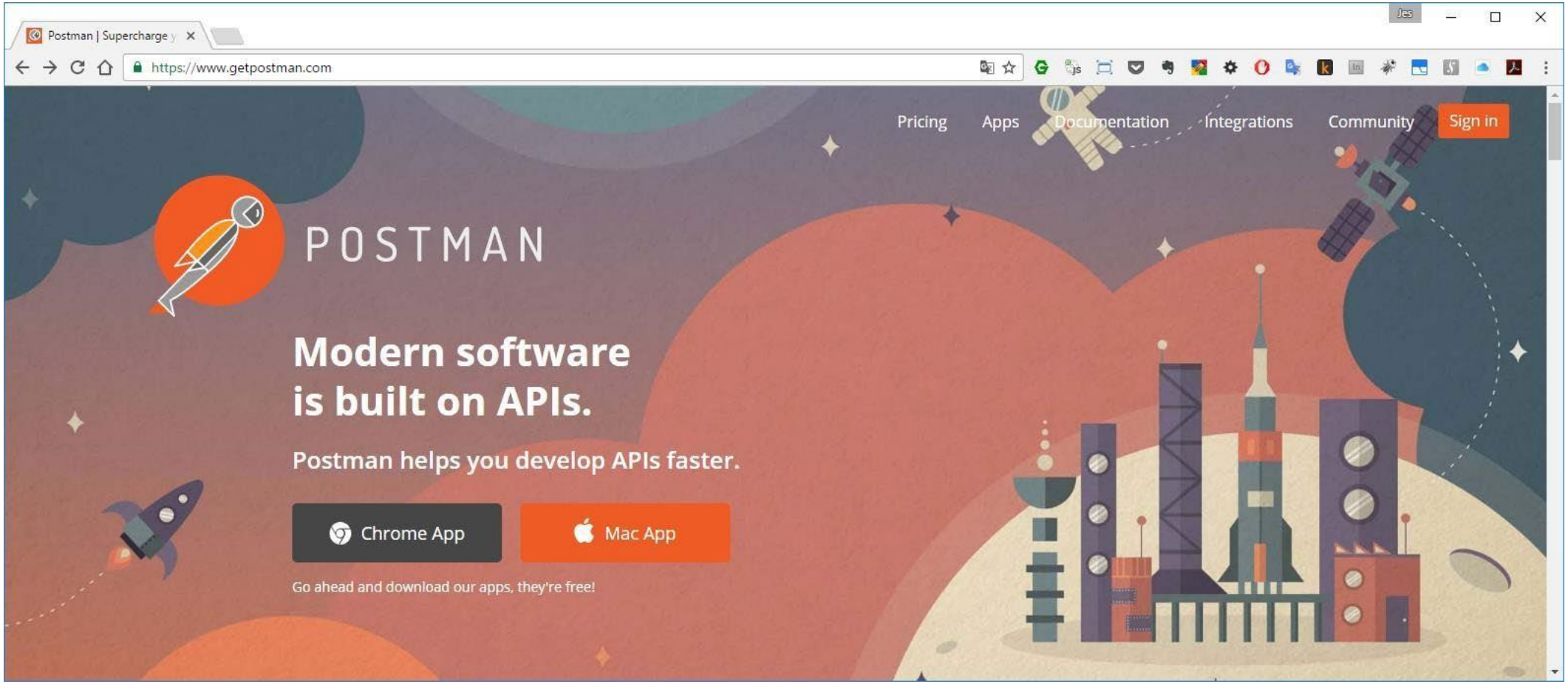
DeleteAllStudents()

```
//  DELETE: api/student
        [HttpDelete]
        public void DeleteAllStudents()
        {
            studentRepository.DeleteAllStudent();
        }
```

Anvendelse af en web service

- Test vha. f.eks. Postman
- Fra C#, f.eks. en windows-klient eller en controller
  - HttpClient
- Fra en web-klient, client side (dvs. javascript)
  - Ajax: https://www.c-sharpcorner.com/UploadFile/dacca2/web-api-with-ajax-understand-post-request-in-web-api/
  - Javascript vha. fetch: https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-javascript?view=aspnetcore-6.0

Test din Web Service - brug Postman



Eksempel med Windows klient, DMI

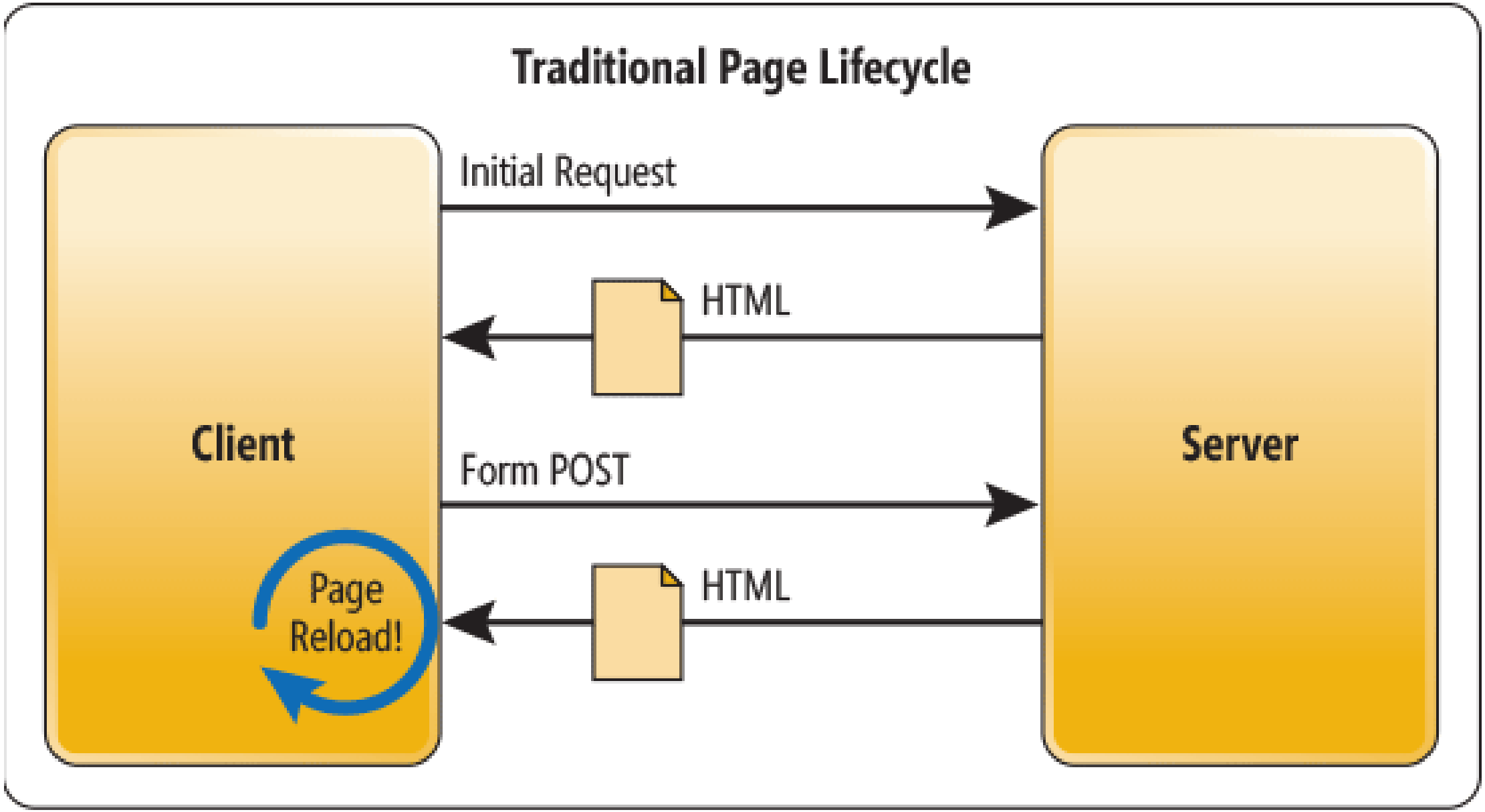
(bemærk den smukke formatering)

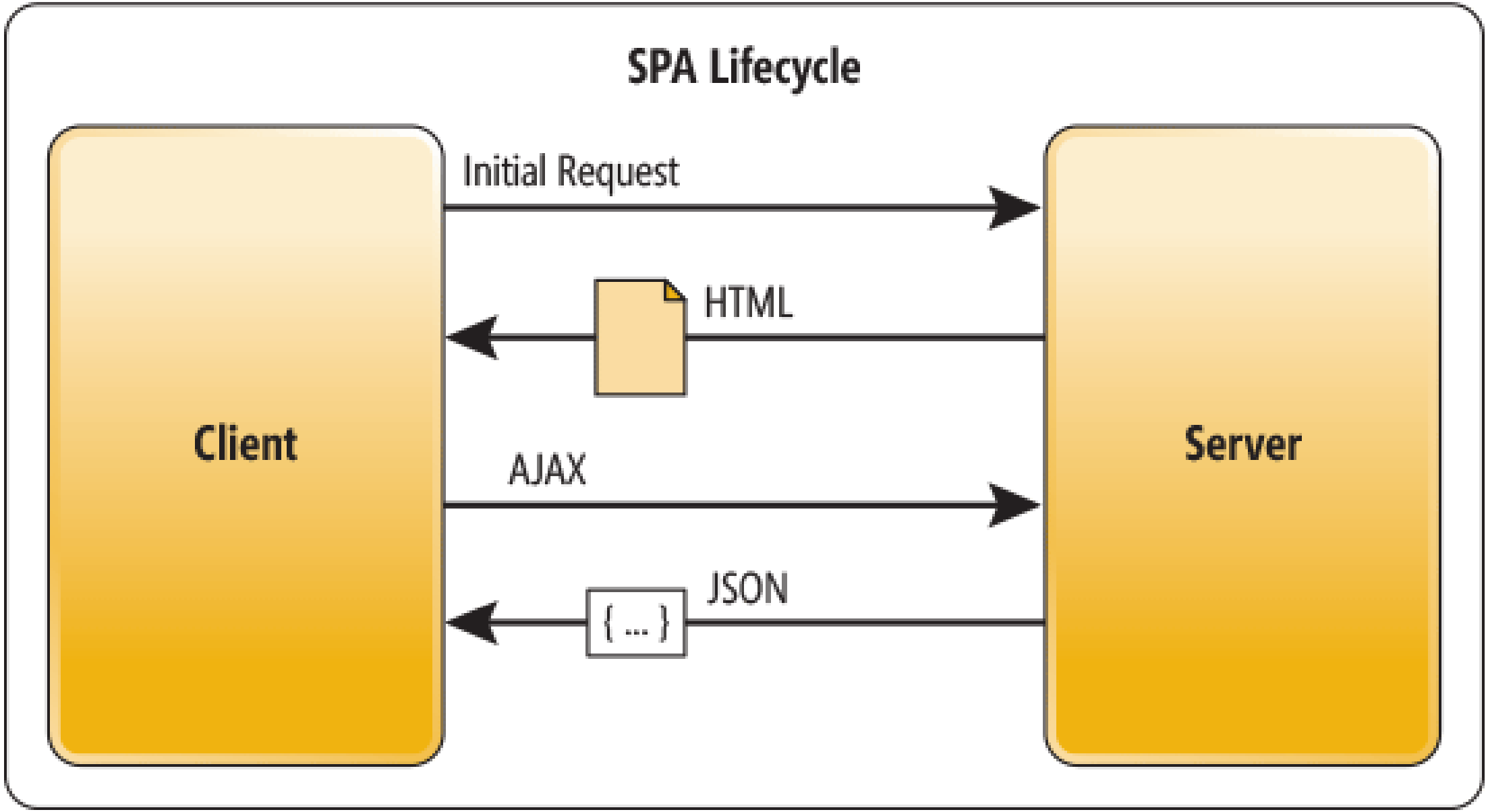
```
 HttpClient client = new HttpClient();
            client.DefaultRequestHeaders.Accept.Clear();
            client.DefaultRequestHeaders.Add("X-Gravitee-Api-Key", "00c07fc3-40ee-
4422-b0b3-d3ee26dba47a");
            string start = DateTime.Now.ToUniversalTime().AddDays(-
30).ToString("O");
            var stringTask =
client.GetStringAsync("https://dmigw.govcloud.dk/v2/oceanObs/collections/observatio
n/items?stationId=23132&limit=20000&parameterId=sealev_dvr&datetime=" + start +
"/..");

            var msg = stringTask.Result;
            var x = JsonSerializer.Deserialize<FeatureCollection>(msg);
```

Eksempel: lisbeth.dk, tilmelding

- Lavet med en helt tredie teknik, men stadig web services
- Opdatering af listbox uden at lave postback

Single Page Applications (SPA)
Et eksempel use-case

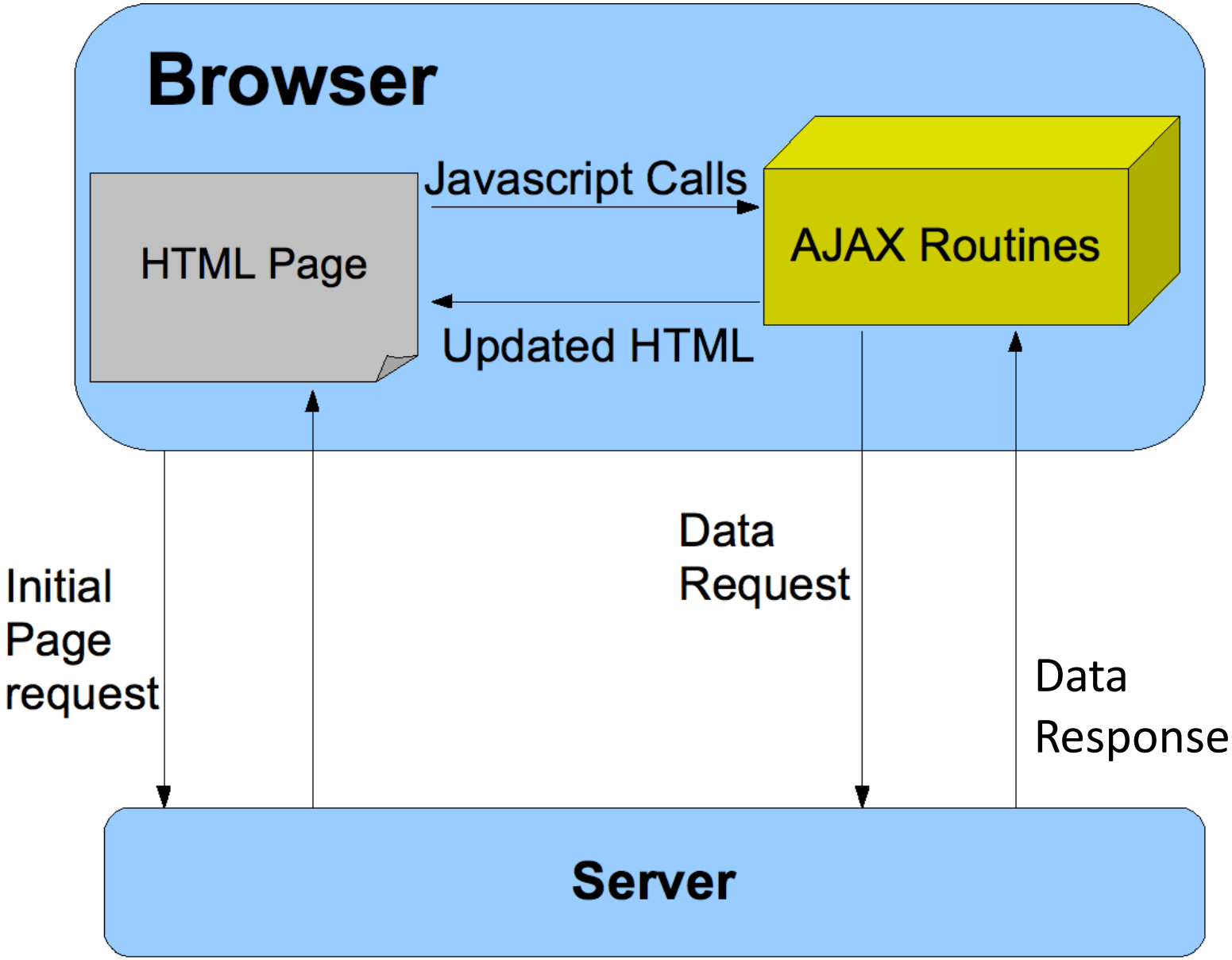

Traditional Page Lifecycle
Client
Server
Initial Request
HTML
Form POST
HTML
Page
Reload!


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

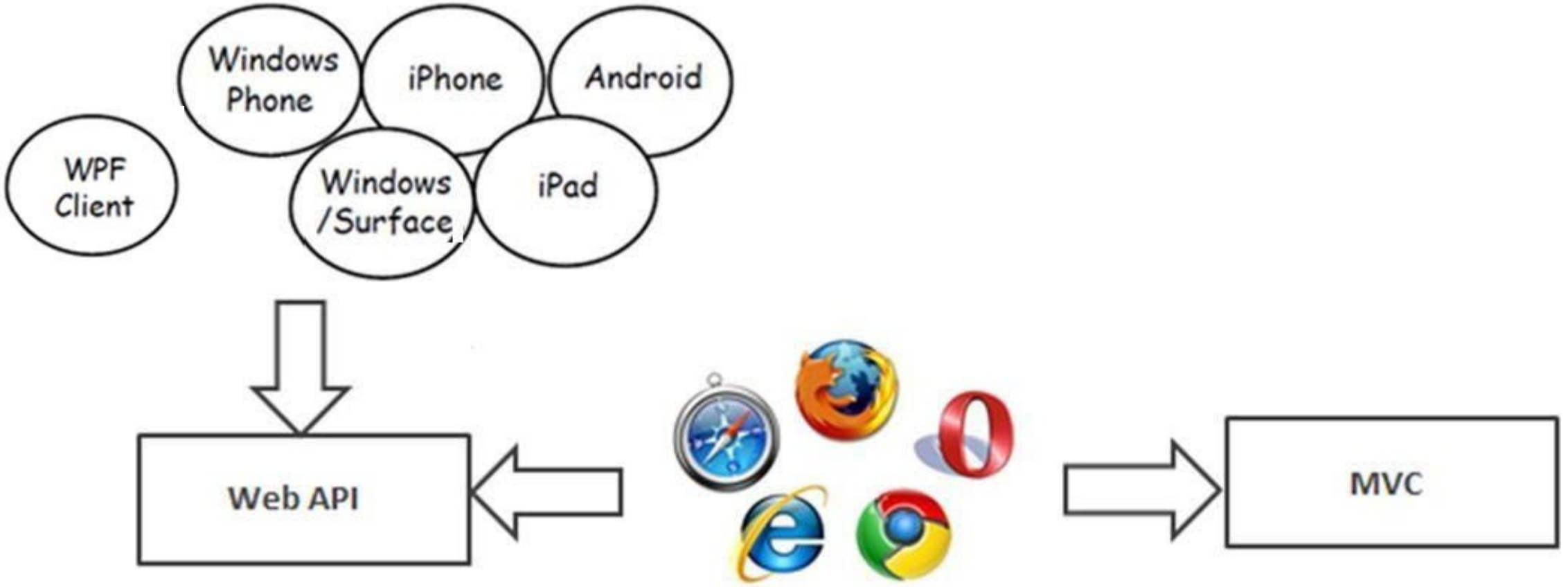

AJAX
Asynchronous Javascript And XML

Fordele ved AJAX AJAX:

- Opdaterer kun en lille del af websiden (af DOM'en), **uden full page reload**.
- Derfor normalt hurtige updateres
- Asynkront : client (browser) står ikke og venter på en server response.
- Eksempel: Se kode
- Install-Package jquery.unobstrusive.ajax.js


Browser
HTML Page
Javascript Calls
AJAX Routines
Updated HTML
Initial
Page
request
Data
Request
Data
Response
Server

MVC og Web API – andre use-cases



Opgaver