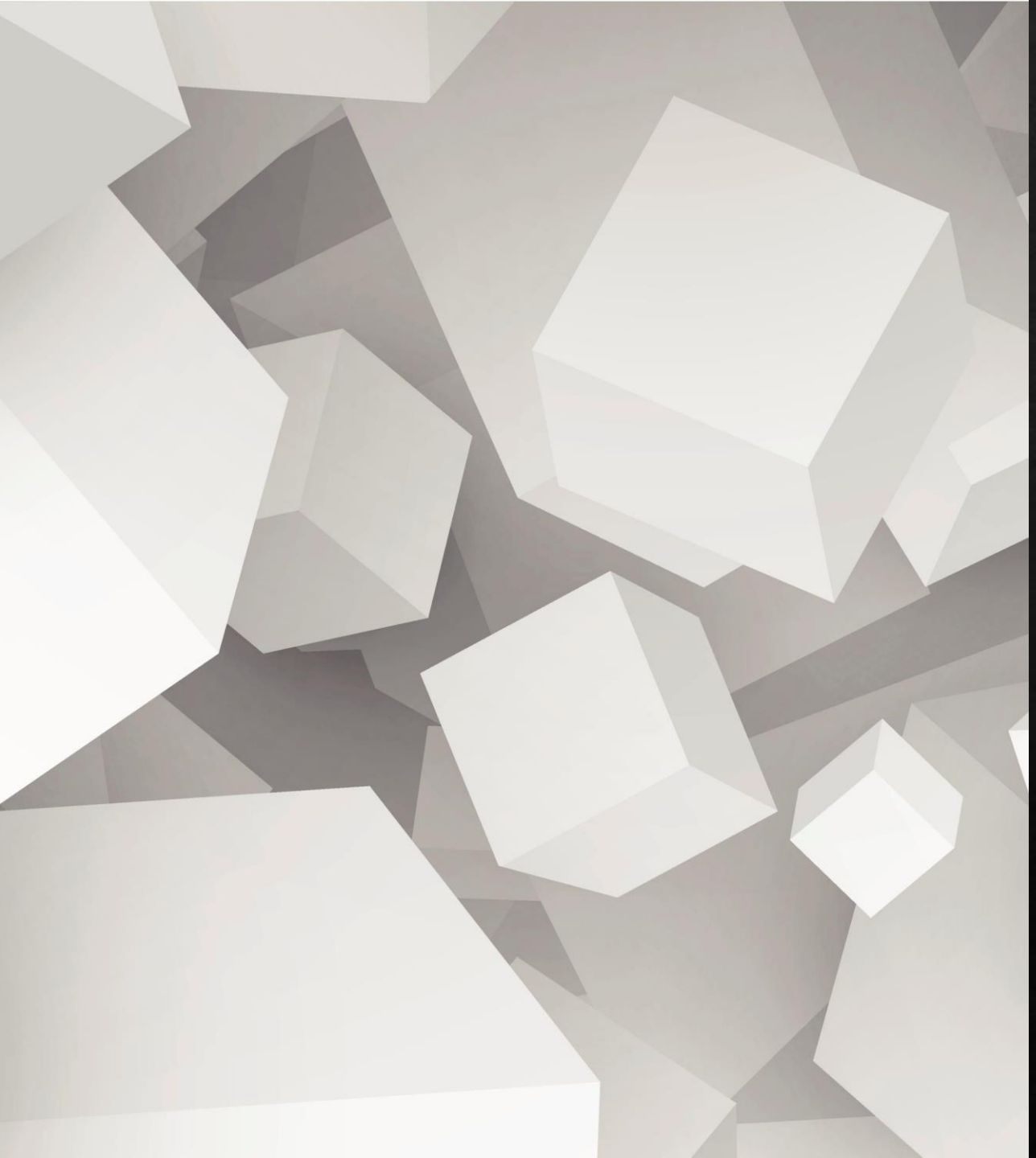
C.O.R.S

Web Data Cluster
ICT50220 Adv programming



What we're covering

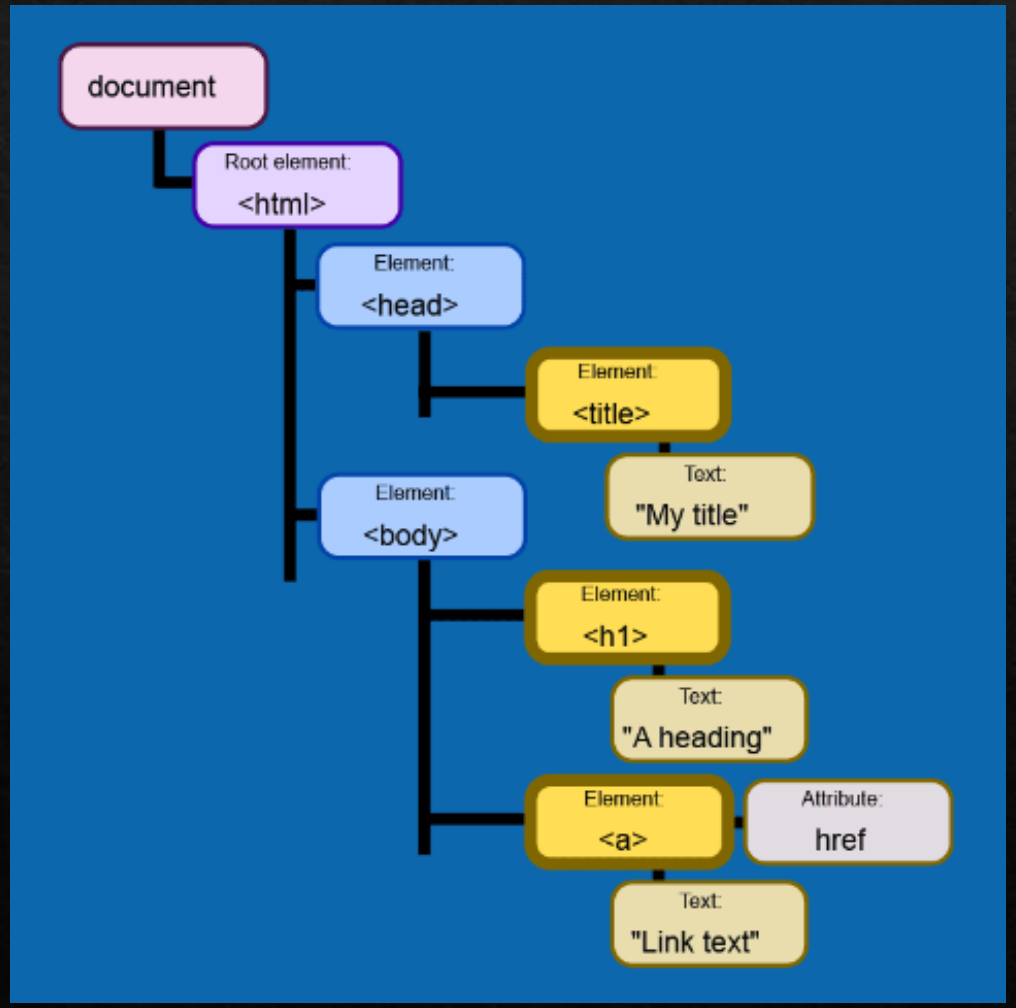
- CORS – What it is.
- OPTIONS – Pre-flight
- HEAD
- Fiddler Application
- ASP.NET Core CORS
- Testing CORS

Brief Internet History

- When the HTTP protocol was first created it was very simple and websites were only static pre-generated sites.
- As time passed the HTTP protocols were improved to support new technologies.
- Two of the most important early improvements were the addition of the JavaScript language and the Document Object Model (DOM) API.
- The DOM API is the system we use when writing HTML to outline and develop web pages.

Brief Internet History

- The DOM API is the system we use when writing HTML to outline and develop web pages.
- This system allowed for more dynamic pages and interactions between systems.
- This however caused issues when people realised that they could embed commands within a page's HTML to cause actions on another site.
- This became a common tools for cyber criminals to use against websites and users.



Same Origin Policy

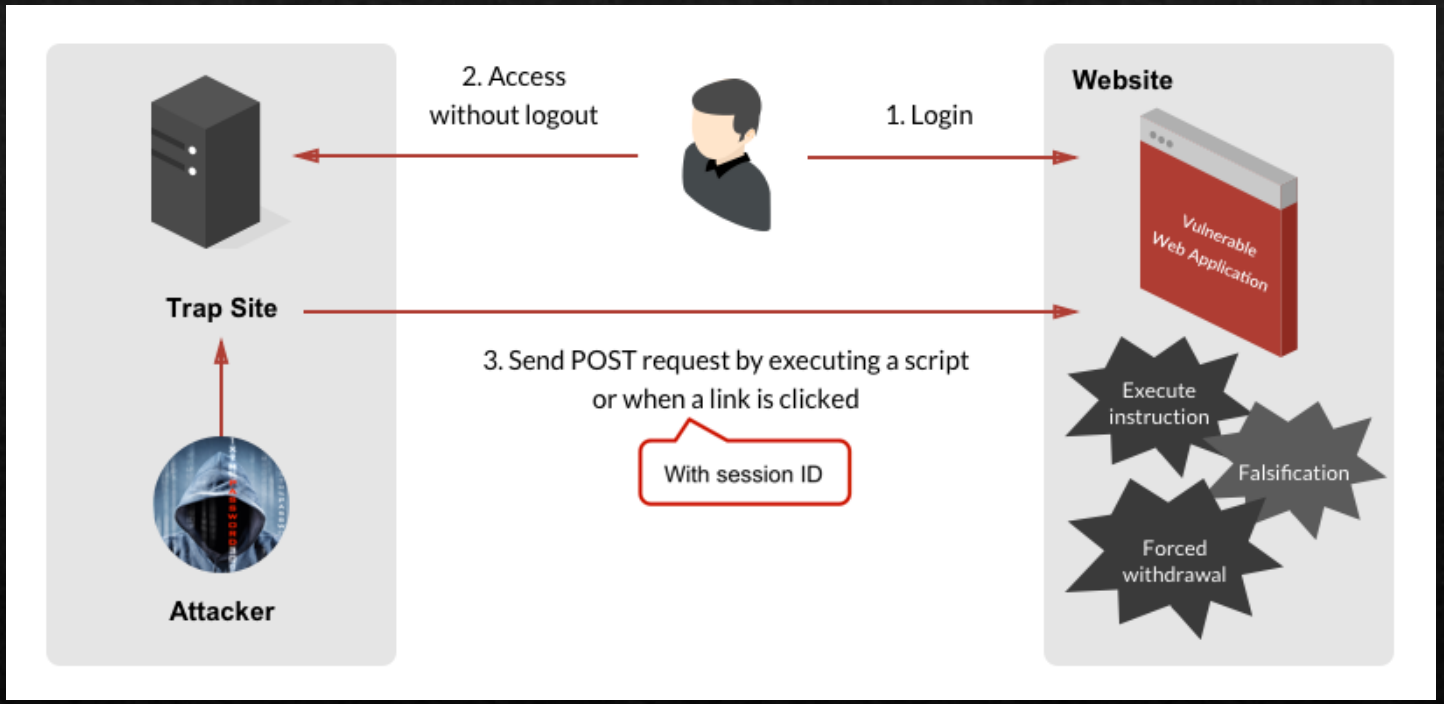
- To combat this, many browser developers began adding a same origin policy system to their browsers.
- This stopped any scripts from one site being used to execute functionality on another site.
- However, this created many limitations in what sites could do and caused issues where cross site interactions were wanted(Social Media etc).

What is CORS?

- CORS stands for Cross-Origin-Resource-Sharing
- Designed as a workaround for the Same-Origin-Policy to allow flexibility of interactions between sites.
- Designed to control interactions between browsers and APIs.
- It is designed to stop allow authorised browsers access to the APIs resources via scripts if they are allowed to do so.
- Allows for configuring what Origins, Methods and Headers are allowed to be used.
- Configured on the Server (Web API), then enforced in the browser.
- Does not affect direct server to server interactions(there are other systems for this).
- Useful to prevent attacks such as Cross-Site Request Forgery (CSRF) or DDOS(Distributed Denial Of Service) Attacks.

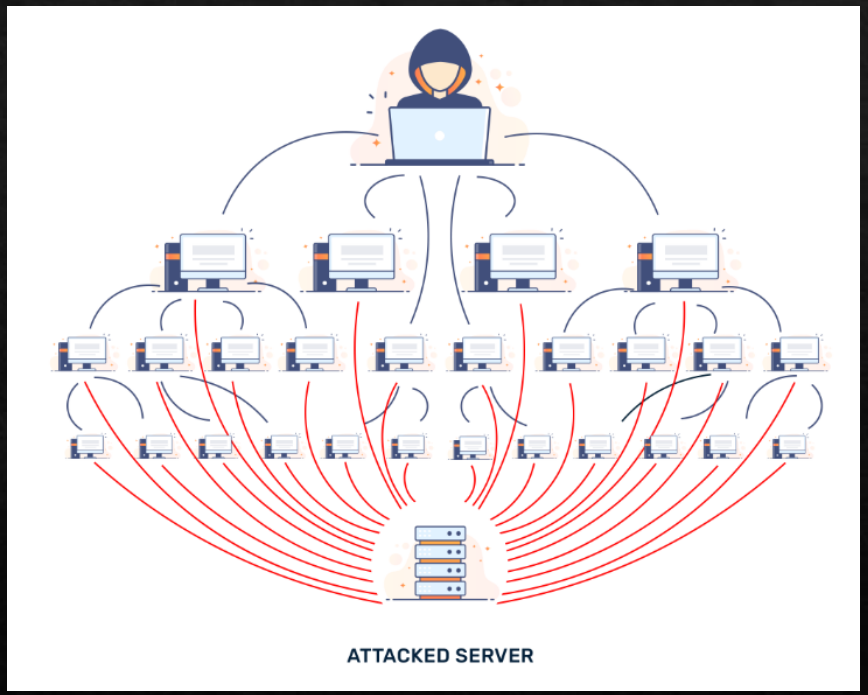
Cross Site Request Forgery

- CSRF is when a user is tricked into accessing a fake site which looks like the real one or embeds the real one inside itself.
- When you log into the site it logs you into your account and then executes a script from your browser which has been embedded in the fake page.
- At the same time, it sends a HTTP request to the API of the site to execute additional commands without your knowledge.
- This can be used to steal money, lock you out of your accounts or other harmful actions.



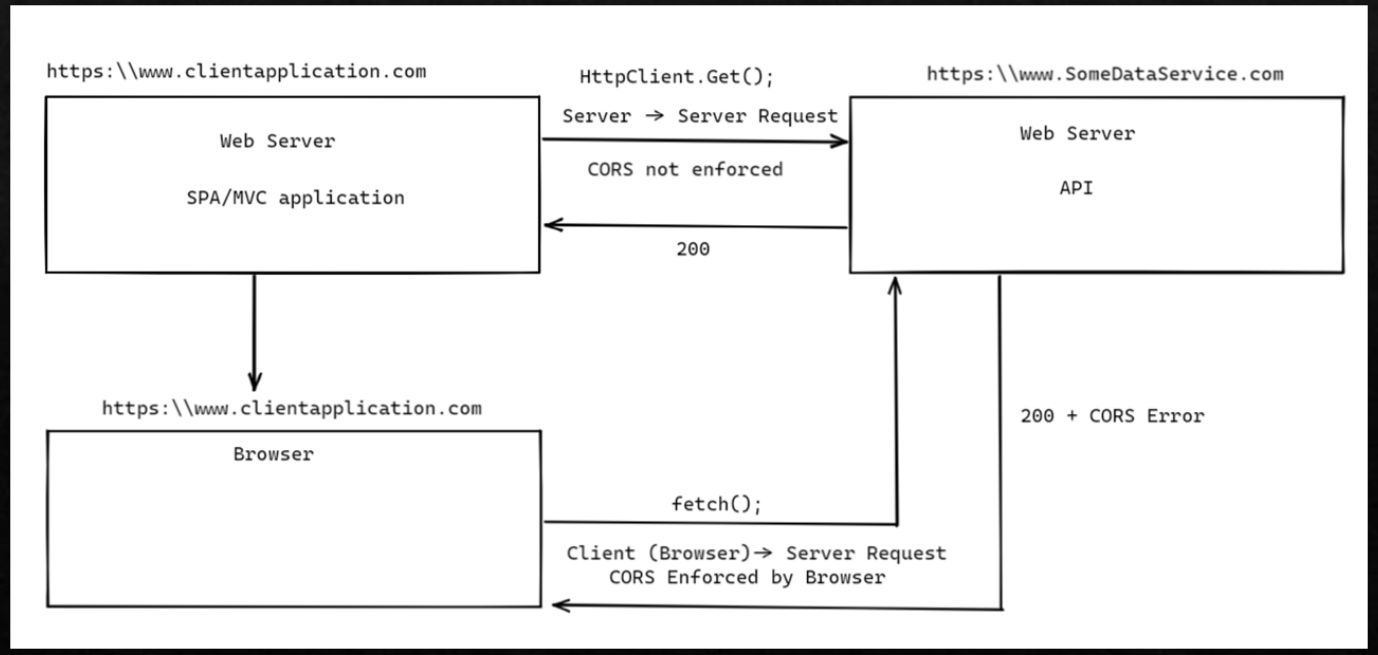
DDOS Attack

- In a DDOS attack the goal is to send as many request as possible to a server that it can't withstand the workload and it fails.
- In modern distributed attacks, instead of attacking from one machine, hackers try to get many unknowing machines to contribute to the attack.
- This can be done by setting up scripts embedded within fake URL links and webpages to be shared around the internet which then trigger the attack scripts from within user's browsers as they are shared across the web.



What CORS does.

- CORS creates a whitelist of allowed domains/webservers that can execute these requests and then denies everyone else.
- This way we can allow requests from authorised sites, but if someone tries to embed additional code to be executed by the user's browser on a fake site, it will get rejected.
- It is an additional layer of security to help protect your API and its users.



CORS Continued

- CORS can be used to set the following:
  - Allowed Origin (where the request is coming from)
    - Can be set to a wildcard '*'
  - Allowed Methods (GET, PUT, POST, DELETE)
  - Allowed Headers – restricting the use of some headers
- All can be set to wildcards, or allowed globally
  - This allows any client-side application access to your API
  - Should only be allowed in a public facing API

CORS Pre-Flight

- Pre-flight requests are sent by the browser when the intended request is not *simple*
  - A simple request can be defined as using the following methods:
    - GET, HEAD, POST
  - A simple request will also only use the following headers:
    - Accept, Accept-Language, Content-Language, Content-Type
- Some Browsers will consider all requests as non-simple and do pre-flight checks on all requests.
- Any request sent outside of these parameters will trigger a pre-flight request.
- The pre-flight request is intended to first check the servers CORS configuration, sending an OPTIONS request.
- This is then used to determine whether the request will be accepted before attempting to send it.

OPTIONS request

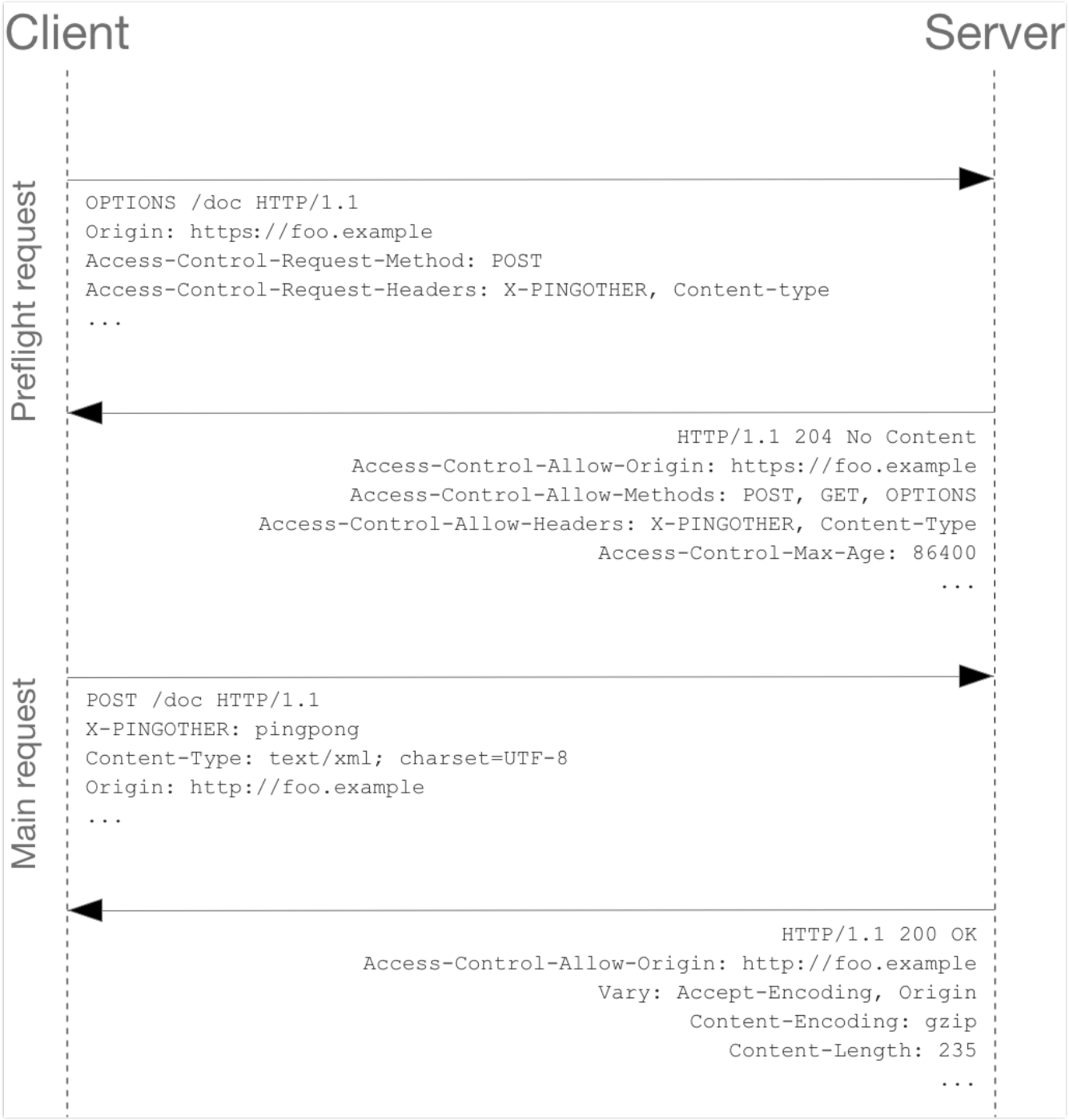
- The OPTIONS method is used in a pre-flight request to send the intended Method, Origin and Headers to the server to confirm these are compatible
- If the OPTIONS request fails, the pre-flight fails and the intended request is not sent
- The OPTIONS request is sent with a range of headers that describe the required configuration:
  - Origin: https:\\exampleorigin.com
    - The domain of the requesting application
  - Access-Control-Request-Method: POST
    - The method that will be utilised with the actual request
  - Access-Control-Request-Headers: [Content-Type, X-SpecialHeader, X-OtherSpecialHeader]
    - All of the non-simple headers that will be included in the request

OPTIONS response

- The OPTIONS response returns a 204 (No Content) containing headers that describe the Server configuration:
  - Access-Control-Allow-Origin: (origin name or *)
  - Access-Control-Allow-Credentials: true/false
  - Access-Control-Allow-Methods: [POST, GET, OPTIONS]
  - Access-Control-Allow-Headers: [Content-Type, X-SpecialHeader, X-OtherSpecialHeader]
- If the response lines up with the request, the required Request is sent to the server, otherwise a CORS error is displayed in the browser's console

Pre-Flight Example

- If the request is not simple, a preflight request will be sent via an options request outlining the headers and request type it intends to send..
- The pre-flight response will tell the browser whether the upcoming request will be accepted or not.
- If the options response confirms the request will be accepted , it will be sent.



HEAD Method

- The HEAD method is used to send a request to an endpoint to request information about the potential response
- This can be used to check the size of the response to a GET request without sending the GET request
- Useful for taking action if content is in the wrong format, or is too large to be handled by the client
- Will not return content to the client
- Can be added to an endpoint in ASP.NET Core with the [HttpHEAD] attribute
- Primarily used to get metadata about an endpoint to determine how it can be used.

CORS in ASP.NET Web API

- CORS can be configured in the Program.cs class, as a service and then injected into the pipeline
- The example below will create a policy called "GeneralOrigin" that allows any method and header, as long as it comes from https://www.google.com or any sub-page on this domain

```
builder.Services.AddCors(options => { options.AddPolicy("GooglePolicy", d =>
    {
        d.WithOrigins("https://www.google.com", "https://www.google.com.au");
        d.AllowAnyHeader();
        d.WithMethods("GET", "PUT", "POST", "DELETE");
    });
});
```

```
app.UseRouting();

app.UseCors("GeneralOrigin");

app.UseAuthorization();
```

Testing CORS

- Activity - Using the 'NoteAPI' developed in class:
  - Add Cors, providing a single allowed origin
  - Use fetch or XMLHTTP from any (not whitelisted) website to send a GET request to the API endpoint
    - This will send a request from the origin of the browser tab

```
// GET
await fetch("https://localhost:44369/api/note/");
```

  - Observe the response and error message
  - Repeat the process from an origin that is listed as allowed
  - Review the difference in the browsers console.

Testing CORS with OPTION

- Activity - Using the 'NoteAPI' developed in class:
  - Open Fiddler and ensure there are no filters applied
  - Use fetch or XMLHTTP from any (not whitelisted) website to send a PUT (or non-simple GET/POST) request to the API endpoint
  - Review the two request and response pairs in Fiddler

```
// PUT
await fetch("https://localhost:44369/api/note/15",{
method: 'PUT',
    body: JSON.stringify({
        noteId: 15,
        title: "newTitle",
        body: "newBody",
        dateCreated: "2022-08-09T12:29:01.9517508",
        authorId: 6,
    }),
    headers: {"Content-Type": "application/json"}
})
```

Challenge

- Set up CORS to only allow GET, OPTIONS, PUT, POST and DELETE requests from all origins, including the following custom headers: x-sent-by, x-client-name
- Test this setup using the fetch command to the right

```
await fetch("https://localhost:44369/api/note/15",{
method: 'PUT',
    body: JSON.stringify({
        noteId: 15,
        title: "newTitle",
        body: "newBody",
        dateCreated: "2022-08-10T12:29:01.9517508",
        authorId: 6,
    }),
    headers: {"x-sent-by": "yourNameHere",
              "x-client-name": "chrome browser",
              "content-type": "application/json"}
})
```

Fiddler – Viewing HTTP Traffic

- Fiddler can be used to track all requests and responses on a system
- Useful for debugging the request -> response action between two system
- Fiddler Classic can be used for free if a valid email address is used:
  - https://temp-mail.org/en/ (for a temporary email address)
  - https://www.telerik.com/download/fiddler/fiddler4 (download Fiddler)
- Fiddler is similar to the 'Network' tab in Chrome dev tools, but works for your entire PC, not just a single browser tab