Fachhochschule
Nordwestschweiz

Application Security (apsi)

Lecture at FHNW

Lecture 12, 2020

Arno Wagner, **Michael Schläpfer**, Rolf Wagner

<arno@wagner.name>, <{michael.schlaepfer,rolf.wagner}@fort-it.ch>

Agenda

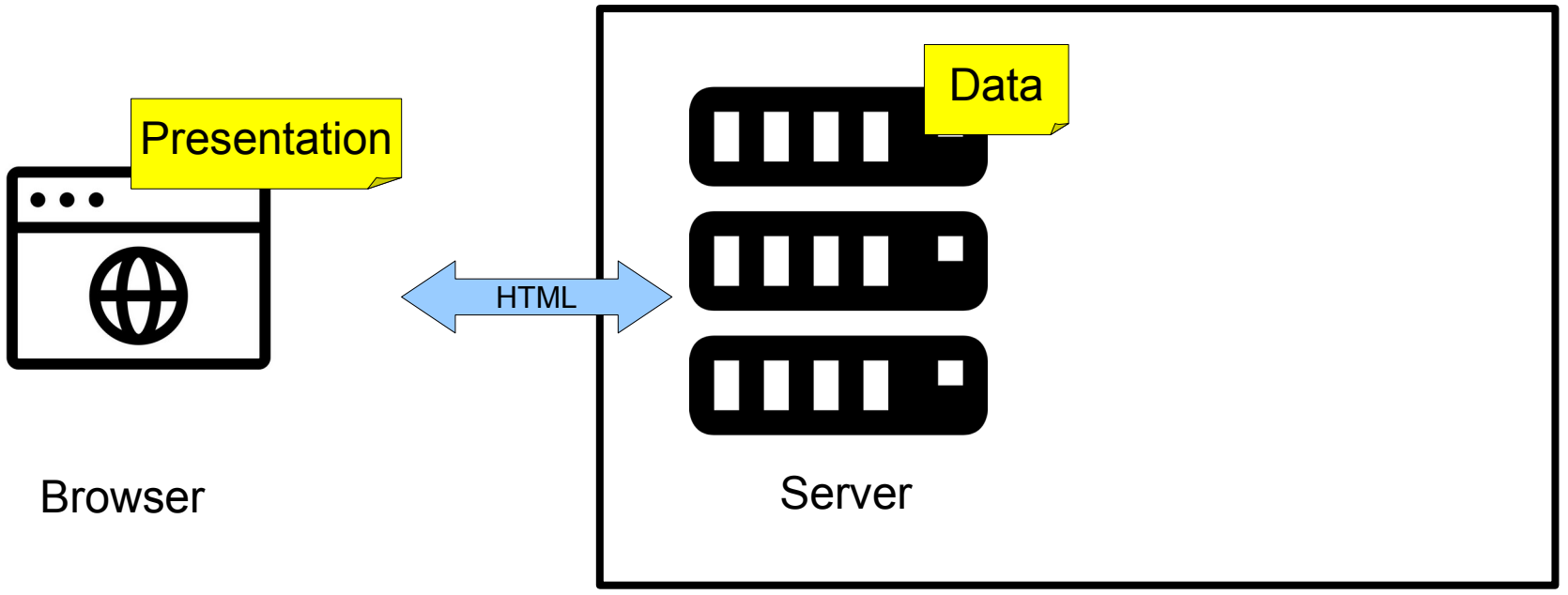
- **Modern Web Technology**
- CORS vs. Content Security Policy (CSP)
- Webframeworks
- Example: MEAN Stack

Welche Sprachen haben Sie schon für das Erstellen von Web Apps verwendet?

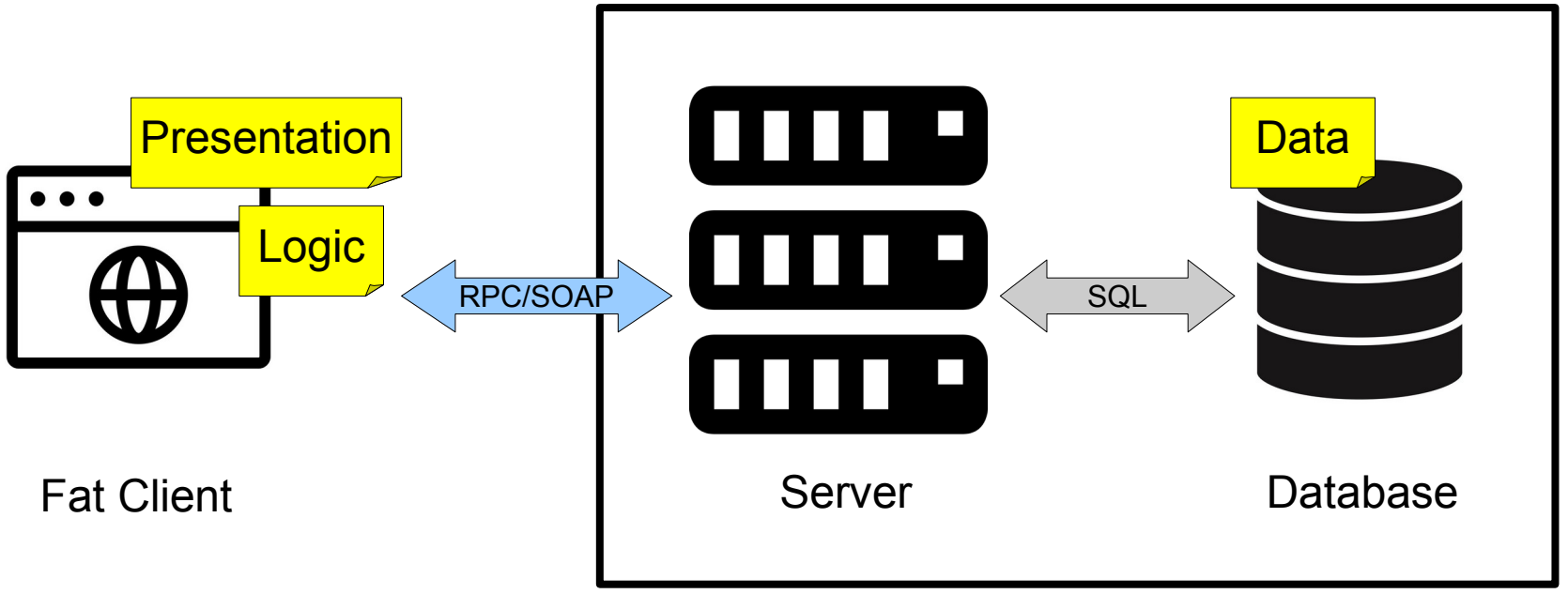
www.menti.com

Between 70% and 80% of all mobile and cloud breaches are due to misconfigurations.
– Gartner (2019)

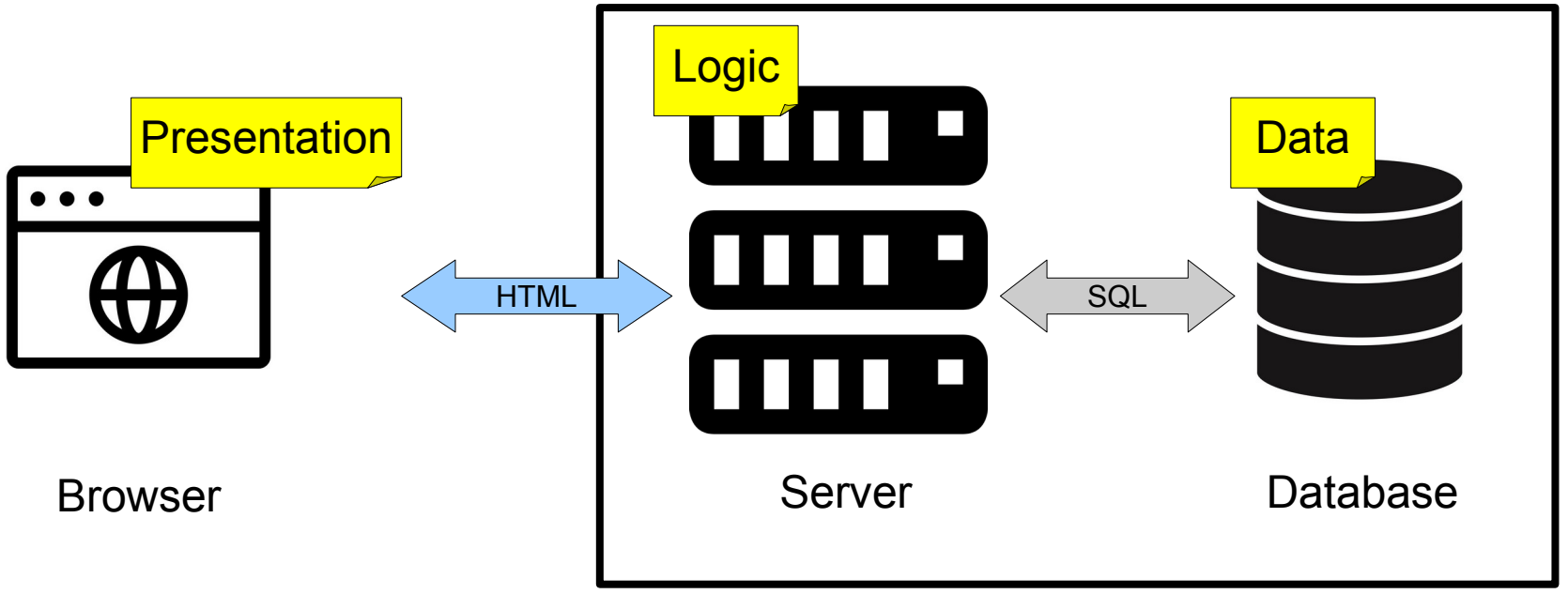
History on Web Technology (~1990)



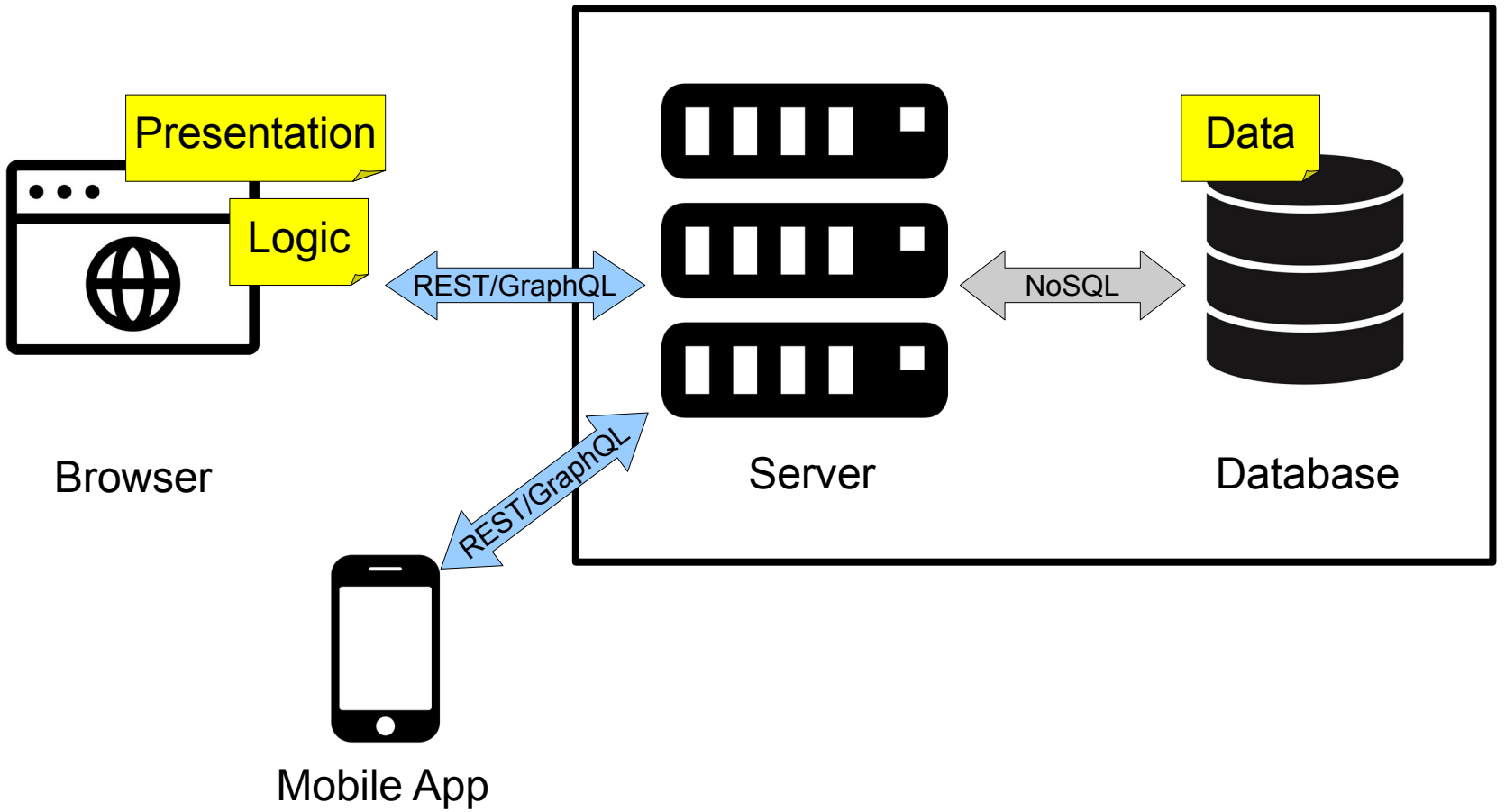
History on Web Technology (~2000)



History on Web Technology (~2010)



History on Web Technology (~2020)



Pitfalls

What could possibly go wrong when we put the logic in the browser?

- Secrets must not be stored in the application (browser)
- Authentication flows must be validated by the backend (e.g., challenge/ response)
- Every API call's authorization must be validated (principle of complete mediation/ Zero Trust)
- The application's logic cannot be trusted, since the adversary is in control of it's execution (WAFs are useless with respect to ensuring Layer-7-logic)

Agenda

- Modern Web Technology
- **CORS vs. Content Security Policy (CSP)**
- Webframeworks
- Example: MEAN Stack

Recap on Same Origin Policy (SOP) and its Limits

Sometimes, SOP is not enough.

Example:

- A web-page may want to embed content from a different server, and request this via a script on client-side
- SOP prevents scripts from accessing a non same-origin page

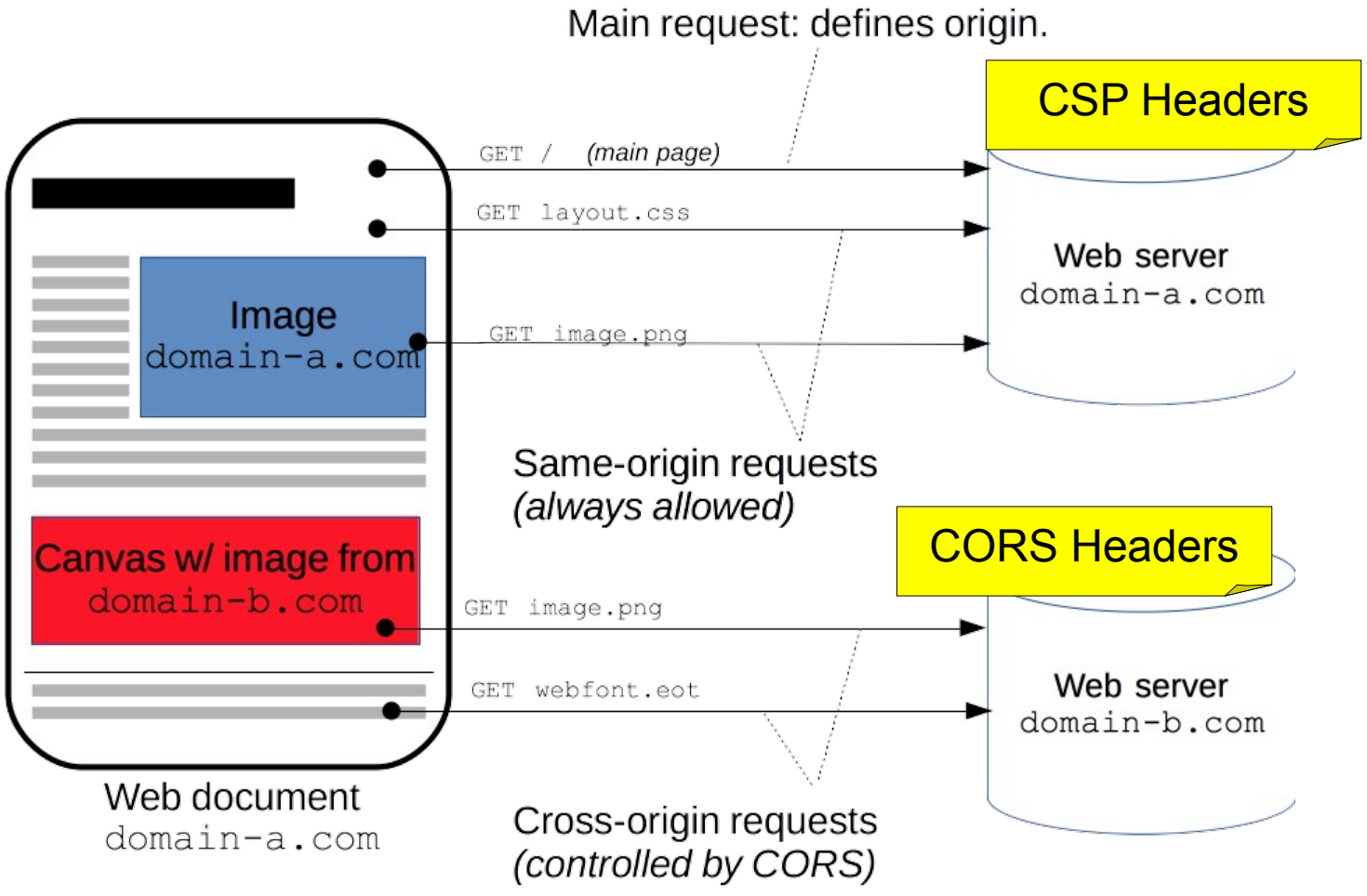
Solution (one of several): Cross-origin resource sharing (CORS)

Original page is from server A and the request from script goes to B

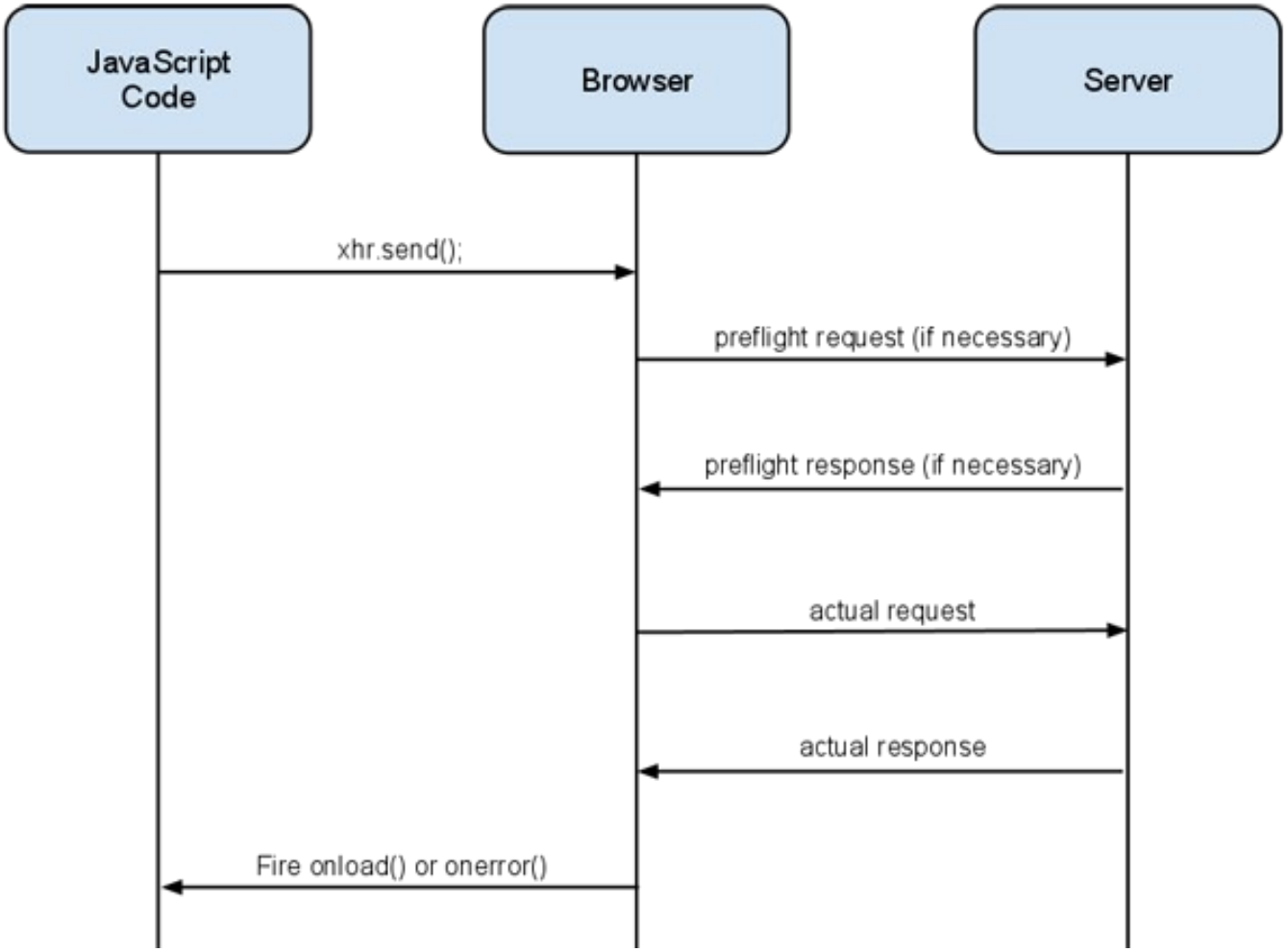
1) Browser asks B via OPTION request for access, stating a as Origin
2) B allows it via "Access-Control-Allow-Origin: A"
3) Request to B is made

Note: "Access-Control-Allow-Origin: *" is for "everybody may access this"

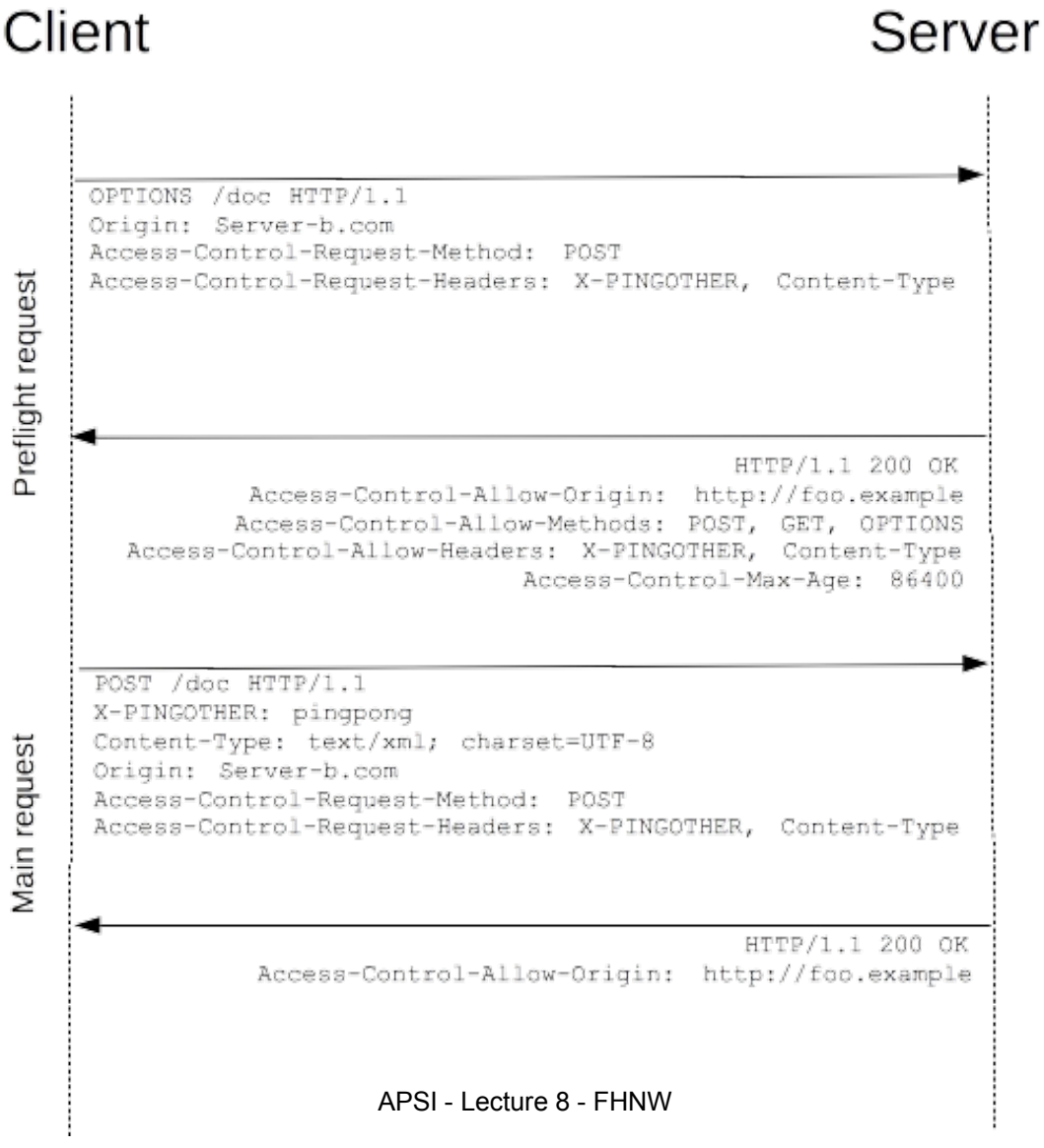
CORS and CSP Overview



CORS Preflight Requests



CORS Example (slightly abbreviated)



CORS for Multiple Domains

- CORS headers are set for one domain only
- If multiple domains shall be allowed to access the ressources, the CORS headers must be set dynamically
- Example (.htaccess) to allow loading of external fonts:

```
<FilesMatch "\.(ttf|otf|eot|woff|woff2)$">
    <IfModule mod_headers.c>
        SetEnvIf Origin "http(s)?://(www\.)?(google.com|staging.google.com|
development.google.com|otherdomain.example|dev02.otherdomain.example)$"
AccessControlAllowOrigin=$0
        Header add Access-Control-Allow-Origin %{AccessControlAllowOrigin}e
env=AccessControlAllowOrigin
        Header merge Vary Origin
    </IfModule>
</FilesMatch>
```

- Since the header depends on the Origin, it must not be cached for other domains: Vary HTTP header

Content Security Policy (CSP) Headers

- Proposed in 2004 by Robert Hansen (Content Restrictions)
- New set of HTTP response headers set by the web application
- Helps reducing XSS attacks, clickjacking, and code injection attacks
- Allows fine-grained control on resources to be loaded and executed in the browser: Reference
- Too restrictive for legacy applications (substantial refactoring needed)
- Examples:

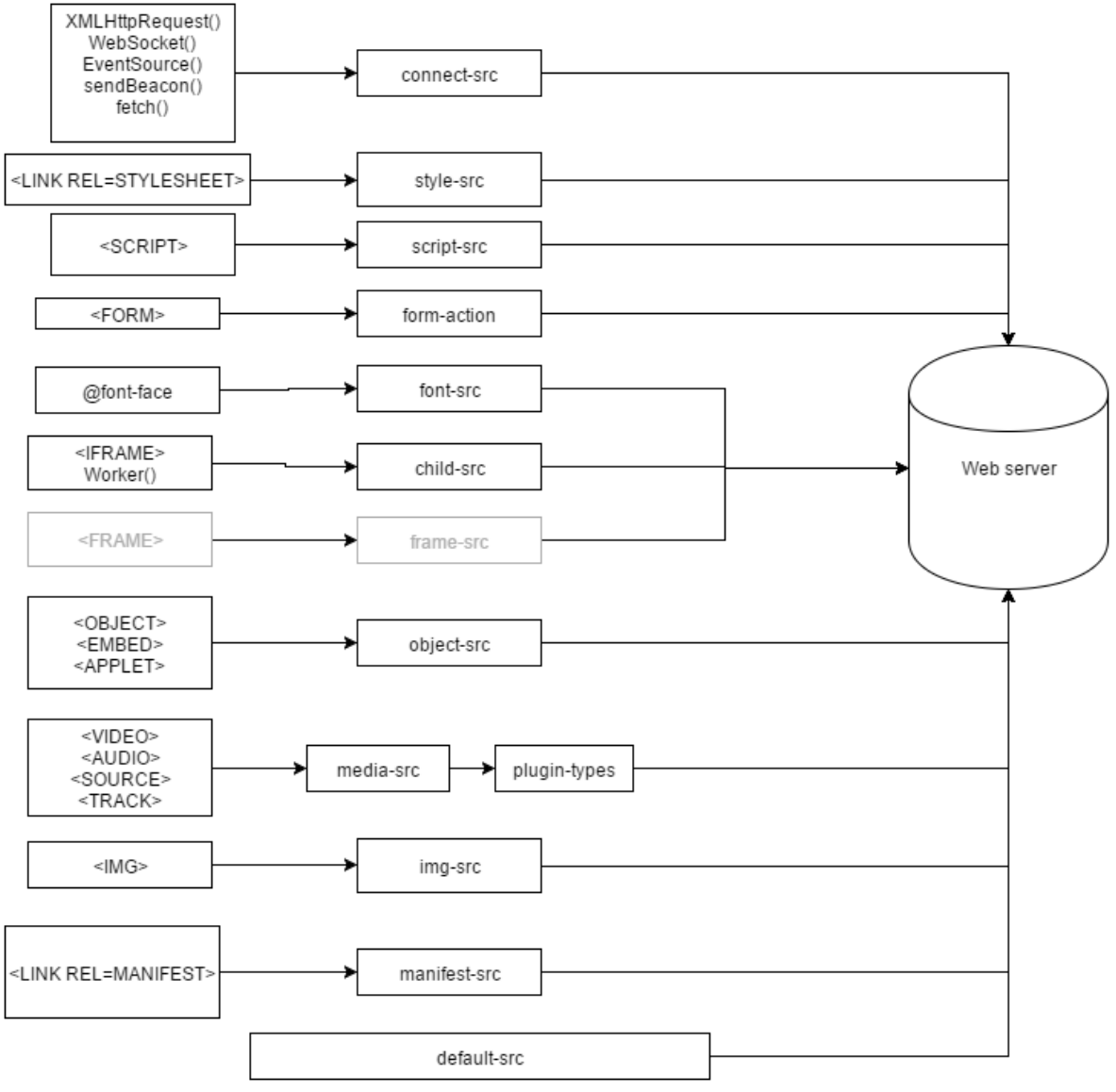
```
Content-Security-Policy "default-src 'none'; script-src 'self'; connect-src
'self'; img-src 'self'; style-src 'self';"

Content-Security-Policy "script-src 'self' www.google-analytics.com
ajax.googleapis.com;"
```

- Can be set in the HTML code directly (<head>):

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self';
script-src 'self'">
```

Overview of Common CSP Options



Agenda

- Modern Web Technology
- CORS vs. Content Security Policy (CSP)
- **Webframeworks**
- Example: MEAN Stack

Webframeworks

Webframeworks are useful for developing web applications. Amongst others, the most common reasons are:

- Standardized development
- Automation (e.g., CI/CD, build pipelines, automated testing)
- Cross-browser compatibility
- Reusable components (many available in repositories)
- Lower engineering skills needed for development –> Full stack devs
- Faster development with less resources
- Shorter time-to-market
- Cheaper development

Risks with webframeworks

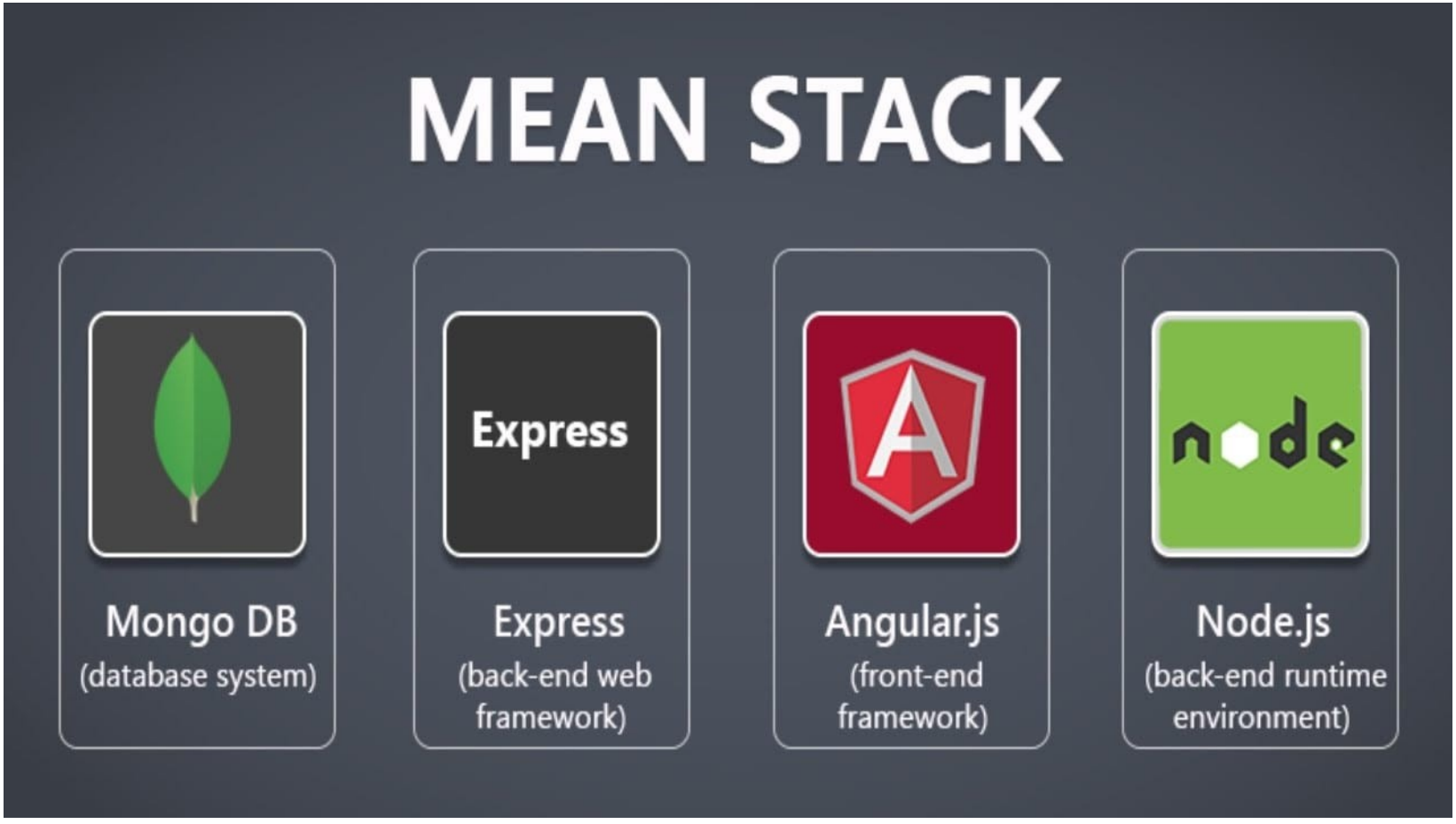
The advantages of webframeworks come with corresponding risks. Amongst other these are:

- Developers are becoming just (framework) users
- Missing knowledge of actual implementation
- You get what you pay for
- Trust in repositories and out-of-the-box components
- Maintenance and security fixing depends on others/unknown
- Own software quality depends on others/unknown (rotten code, remember?)
- Commonly used components are more prone to be attacked
- Vulnerabilities in components a multiplicative risk

Agenda

- Modern Web Technology
- CORS vs. Content Security Policy (CSP)
- Webframeworks
- **Example: MEAN Stack**

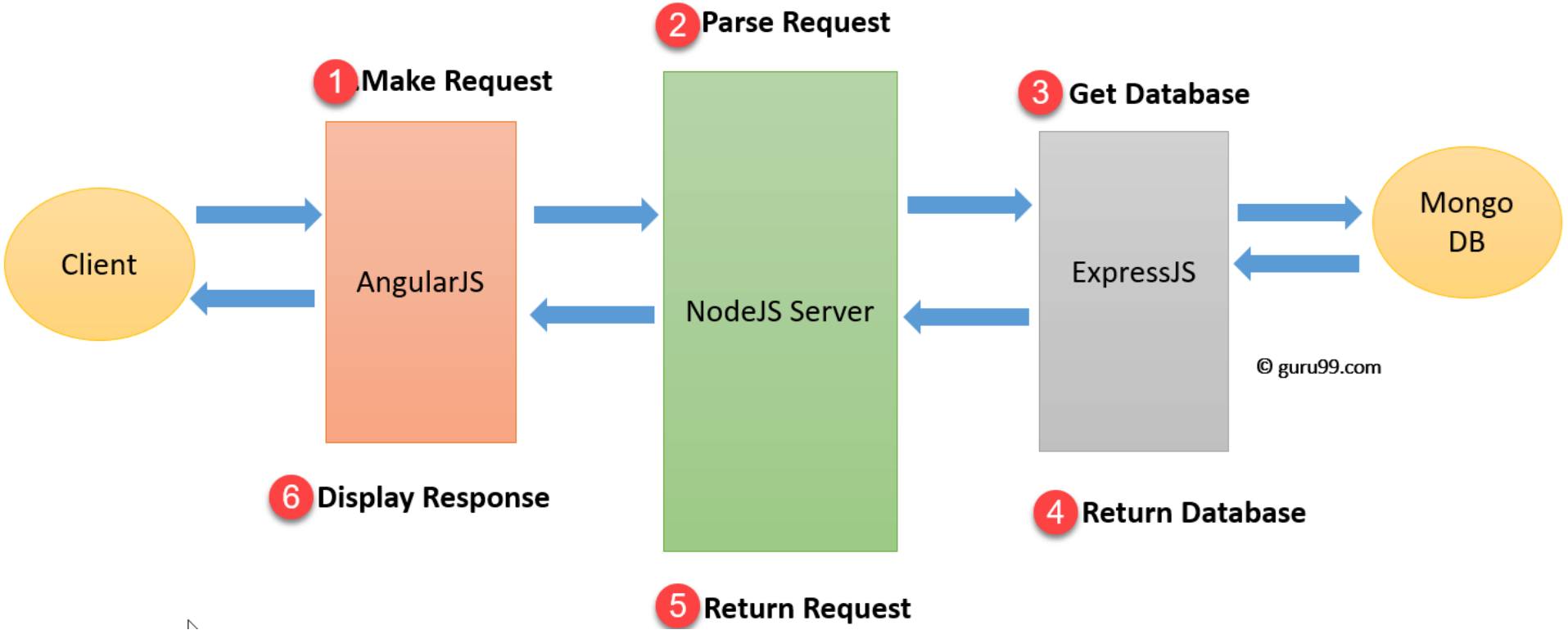
Examples of webframeworks



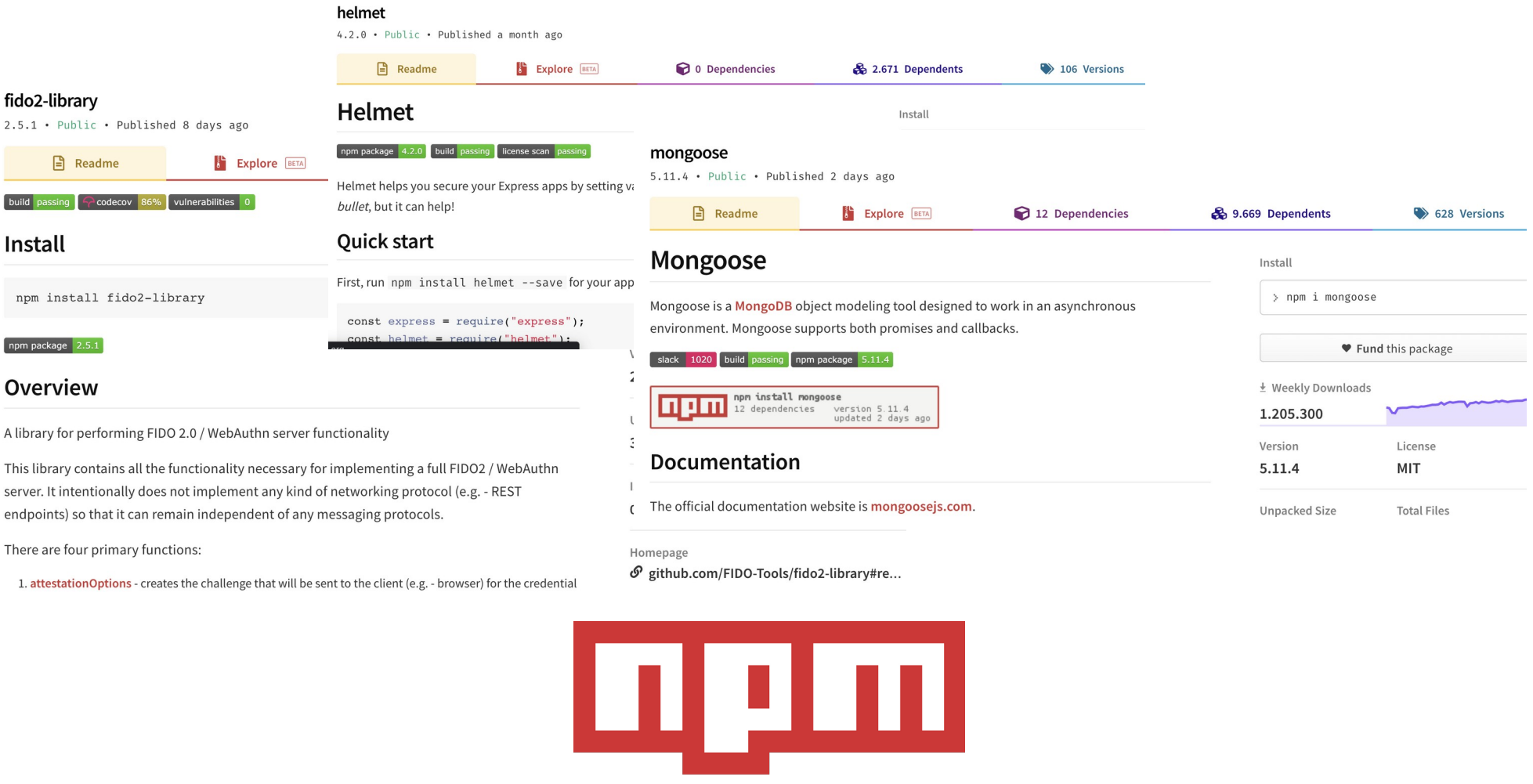
Mit welchen Komponenten des MEAN Stack haben Sie bereits Erfahrung?

www.menti.com

MEAN Stack - Overview



Example: npm Repository ("Node Package Manager")



MEAN Stack - MongoDB

- Disable JavaScript execution
- Use a framework to access your database (e.g., mongoose) but know how it works
- Properly set up Access Control (i.e., separate credentials, RBAC)
- Restrict and limit access to the database
- Encrypt data (at rest, in transit, additionally sensitive data)
- Auditing and logs
- As usual: stay up-to-date with security fixes

MEAN Stack - Express

- Always use TLS
- Use “Helmet” for setting proper security HTTP headers (e.g., CSP): https://helmetjs.github.io/
- Make sure all dependencies are up-to-date and secure (npm audit)

MEAN Stack - Angular

- Use interpolation ({{ }}) to safely encode and escape HTML/CSS expressions withing templates
- Prevent using [innerHTML]
- Prevent using templates concatenated with potential user input
- Do not manipulate the DOM on your own (e.g., node.appendChild() or using the document object), instead use Angular's APIs to manipulate the DOM
- Hold all packages up-to-date and regularly scan for vulnerabilities (npm audit)

MEAN Stack – Node.js

- Use parameterized inputs only to prevent injection attacks
- Sanitize all user input to prevent XXS (Cross-site-scripting) attacks
- Use MFA to prevent automated attacks
- Discard sensitive data after use
- Patch old XML processors to prevent XXE (XML external entity) attacks
- Enforce access control on every request
- Keep all packages up-to-date
- Regularly scan your application for vulnerabilities (npm audit)
- Enable auditing and logging

Take Home Message

- Webframeworks help to develop web applicationns fast, reuse existing work, and avoid common development errors (also security errors)
- Ease of use comes with a price:
  - You have to trust the framework
  - You have to trust the components' developers and the repositories
  - Maintenance and security fixing of the used components is not assured
  - Not the same deep understanding of how things work

Exercise and Project

- Part of next week's project will be to successfully go through the following tutorial:
  https://www.javatpoint.com/mean-stack
- Based on the tutorial we will additionally harden the application (infos will follow next week in the intro (**mandatory 30min lecture from 1215-1245**)
- As today's exercise and preparation for next week:
  - Setup your local development environment:
    https://www.javatpoint.com/setup-components-of-mean-stack
  - Setup a free MongoDB in the cloud (MongoDB Atlas):
    https://www.javatpoint.com/setup-mongodb-in-mean-stack
  - Make yourself familiar with the MEAN stack's project structure:
    https://www.javatpoint.com/exploring-the-mean-stack-project-structure