# Week 1

# Week 2

Architecture

* Blueprints, plans, patterns, standards (easy for others to understand)
* Think of maintenance, trade-offs, limitations (e.g. space)
* Power consumption, efficiency, interaction between components, consistency, bottlenecks, compatibility
* Metric: security, speed, reliability, modularity, scalability, resource management, UI friendly, cost (workforce, hosting, servers, database, licensing, downtime), robustness
* Can use multiple architectural patterns for one system (or incorporate aspects from multiple patterns)

Service Oriented Architecture (Microservice)

* Different services for different functionality
* Communicate through APIs
* Pro: modularity – don’t have to shut down everything to shut down one service
  + Separation of concerns
  + Easier to extend (don’t have to shut everything done

RESTful APIs

* Stateless → requests are independent, server does not send state of each request
  + ex) server responds pages of a book, can’t request “next page” → server doesn’t know previous stage

Web Storage

* Store object in file using serialization
  + File is string of characters
  + Convert object to JSON

Local Storage

* Store in browser side (session object, cookies, cache)

Origin, protocol, port number

# Week 3

Local Storage

* Web storage
  + Local storage
  + Session storage
* Cookies came before web storage
  + Limited size
  + Sent from client to website’s server with each request to that website
* Web storage depends on the browser → max of 10MB for all browsers
* Cookies are 4kB
* Cookies are popular for authentication
* Cookies can be HTTP only → not available programmatically to javascript → only available to the browser
* When you send a request, cookies are already included in the header
* Security: cookies or web storage
  + Consider who can access
  + Web storages are only available to the same origin (same protocol, same port number, same domain)

Term Project

* Can AI agents make phone calls on your behalf using your voice? (is this feasible for the project)

JavaScript Review

* Not defined and undefined are very different
* var is function scoped, let and const are block scoped
* 7 types: object, undefined, null, symbol, string, number, boolean [double check this]
* Undefined is if you define a variable and do not give it a value (because js can’t define a type based on value)
  + Can set something to null (means doesn’t have a valid value)
  + Bad practice to set something to undefined

Neither get nor post are more secure than the other

# 

# Week 4

API Server

* Restful
  + Low cost, simpler, more secure, robust
* Stateless
  + Request doesn’t depend on previous requests, all requests much be independent
* Process at client side to make the server more lightweight

Tips for job interviews

* Ask about what architecture they have used (because then you can share all your knowledge)
* Think out loud
* Put yourself in position of interviewers → try to come up with questions yourself
* Always doubt

Hosting

* Vercel, glowhost(?)

Server

* A computer that can be accessed at any time (by internet)
* Provides a service (e.g. email server)
* Server side scripting: javascript, elixir, php, c#
  + Javascript is good to focus on for job interview according to Amir and big companies won’t even care (except maybe SAP)

NodeJS vs JavaScript

* Can access files and database
* JavaScript can’t make requests → can’t connect to database
* NodeJS is non-blocking → while NodeJS is running, it can let other applications run as well → non-blocking means asynchronous (equivalent)

GET and POST

* Body of POST is not in the history of browser and doesn’t get cached
* You can technically use a GET request to store, it’s just kinda stupid
* Some things can only be done by POST because POST has no limit on data payload but GET has a limit (something below 1MB)

NodeJS

* require → imports all the exported functions from that module

Content-Type

* “text/html”

# 

# Week 5

Asynchronous programming

* How long will it take? No one knows
  + How long to wait until calling a timeout
* Real-time system vs regular system
  + Real-time: instantaneously showing data (instantaneous according to user), response time is calculated reliably, **predictable** and specific time constraint (even if that predictable is 2 minutes – so long)
* We should define timeouts → enforces predictability on unpredictable functions
* We should also be doing stuff at the same time??
* We need two types of responses for async actions
  + One for if there is a success
  + One for if there is a timeout
* In JavaScript, a function **always** returns something (returns undefined if no return value defined)
* Async actions returning a Promise (of type object)
  + Promises invented to deal with async behaviour of internet
* How to call function that returns a promise
  + **async** keyword for function definition → boxes return value into a promise
  + **.then** after the function call to handle return function (not that old)
  + **await** to wait for response (modern)
* There is a queue involved with async functions
* ACID = Atomicity, Consistency, Isolations, Durability
  + Important for payment transactions
* Memory management in JavaScript → garbage collection
* How to create memory leaks in JavaScript
  + Implicit global variables inside functions

Cracking code interviews

Term project → API service

* AI
  + How to train a model
  + For a business
* Security
* Education
  + Current education system is not great

# Week 6

*October 8, 2025*

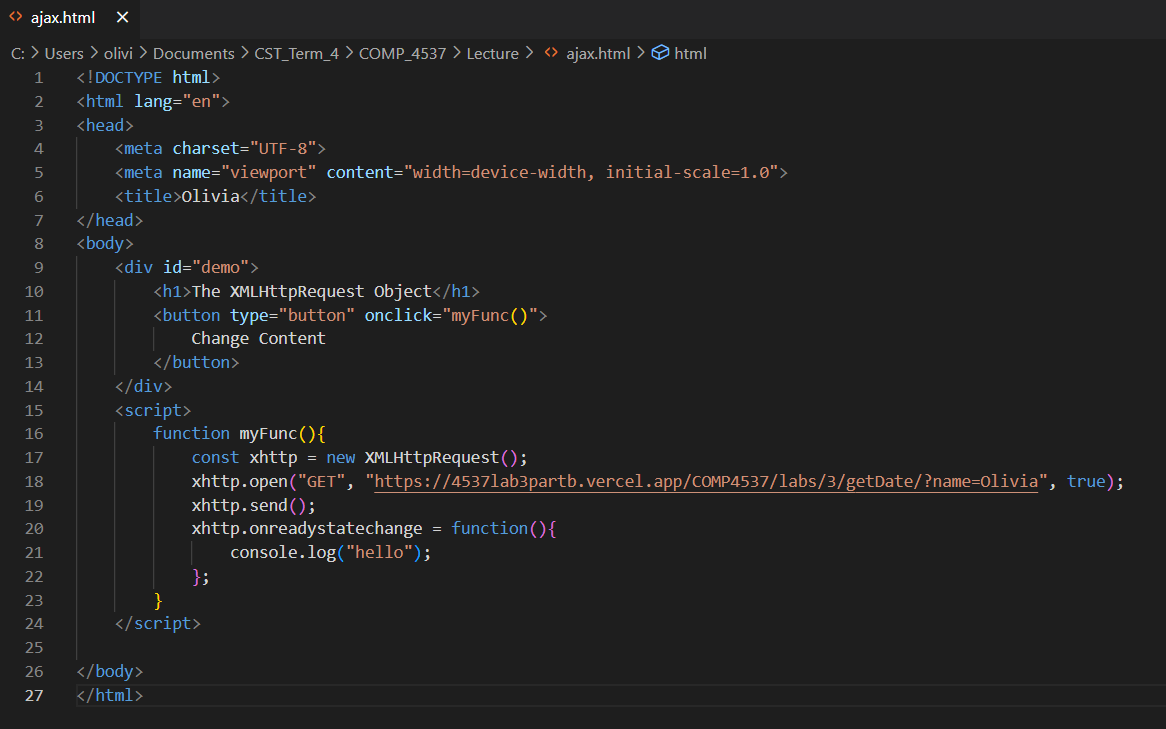
Sending data

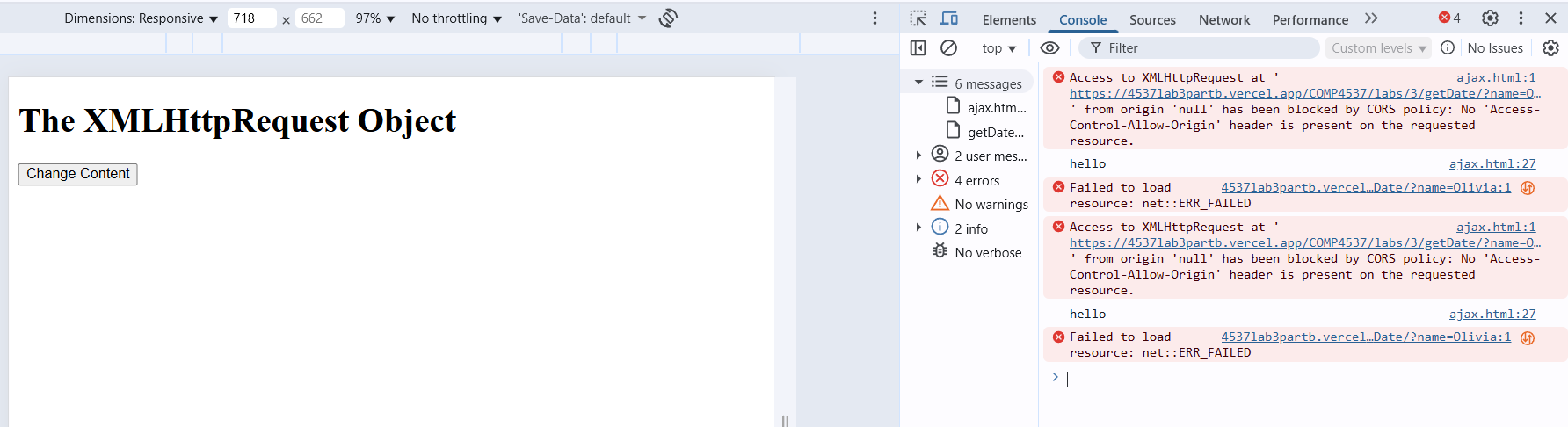
* Long distance (e.g. here to Japan), we would send data in serial rather than parallel for synchronization

AJAX

* Asynchronous javascript and XML
* Can be synchronous if you want it to, but that’s not what it’s meant for
  + Maybe you want to block a thread until a response is received
  + E.g. waiting for content to load on YouTube
  + E.g. promiseall (idk exactly)
* CORS (Cross origin resource sharing)
  + Setting at the server side → server will only provide services to specific origins, set by developer
  + Does not block request coming from address bar or PostMan, only from javascript
* Uses TCP (handshake)
* Web is in application layer

Class Example





Note for Book

* CORS regulations are set server side by the developer, they are executed (implemented, deployed) by the browser
* Browser knows because they are set in the header of the request
  + “Access-Control-Allow-Origin”:“\*”
* Prevent other origins from receiving service
* With options method we can read or retrieve the header response for the server

AI

* Learn how to fine-tune a model
* Learn how to train a model
* MCP

# 

# Week 7

*October 15, 2025*

Important note:

* Milestone 1 of the project is now moved a week later! Yay

GET vs POST

* Major difference: the size of the request is limited for GET request, but for POST there is no limitation (response sizes are not limited)

Promises

* A promise has 3 statuses: pending, fulfilled, rejected
  + Pending: once the promise has been made
  + Fulfilled and rejected are mutually exclusive, both are settled statuses
* Promise() is an object in JavaScript
  + res → for resolving the promise
  + rej → for rejecting the promise
  + res and rej are executed asynchronously and mutually exclusive
* Calling an async function involves two stages:
  + putting the function in the queue
  + fetching it an executing it
* What does Promise.resolve() do? It resolves a promise
* 3 main methods
  + .then
  + .catch → .catch(handleError) is the same as .then(null, handleError)
  + .finally

Example 7

* Why does 3 get executed first?

Summary of today’s discussion

1. The Promise is an object which has a method called then. When we create a function, we pass the executor function which takes 2 parameters: the resolve action function, the reject function action. The resolve and reject functions get executed asynchronously and are mutually exclusive (if resolve is executed then reject is not and vice versa)
2. There are default definitions for resolve and reject → we will get an “Uncaught reject/resolve” error message if we do not define ourselves

# 