Experis Academy notes

By Thomas “Noit” Andersen

[**1. Introduction**](#_fb1tos6oaqhj) **3**

[**Tools**](#_qh3uqc9q5lwx) **4**

[**Language Specific**](#_72xsuys8lpyd) **4**

[JavaScript](#_9iic6ksr1rm0) 4

[Closures](#_mgdx99jcck7f) 5

[Higher Order Function (HOF)](#_cjdgwx8igp97) 5

[Foreach in JavaScript (c# foreach)](#_9al77unayjqt) 5

[Callbacks, Promises, Async](#_4dek068lsvd) 6

[Bind](#_bliejov8n1xd) 6

[Appending HTML to the DOM](#_fihlb4dznttg) 6

[Functional Constructors](#_a6vgws8kl56u) 7

[Prototype (objects)](#_o8ypj8bh1ijq) 7

[Destructuring](#_41sk60uuatfe) 9

[Spread operator](#_res4rt5bqc3g) 9

[Node.js, NPM, Express](#_1cy4rxy3a845) 9

[Vue (JS framework)](#_f7p04eejonoo) 9

[Setup](#_p1icgnbv4qx3) 9

[Basics of Vue](#_rg100ms2b62y) 10

[State Management (Emit, Props)](#_444oxe2j3n28) 10

[Vuex (State Management for Medium-Large projects)](#_62oy2zbbtpi8) 11

[Word definitions](#_rtbnt6u872yz) 12

[How to use it](#_mv729sp7j0pp) 12

[Router](#_2csvaucvrc2s) 13

[React (JS framework)](#_k4ve6klg4i2v) 13

[Angular (JS framework)](#_kvquhpm9dun) 13

[**General Purpose**](#_oj30qtocj7d4) **13**

[Git](#_aawc8pxvqqkd) 14

[Docker](#_b2j401bxwm3k) 14

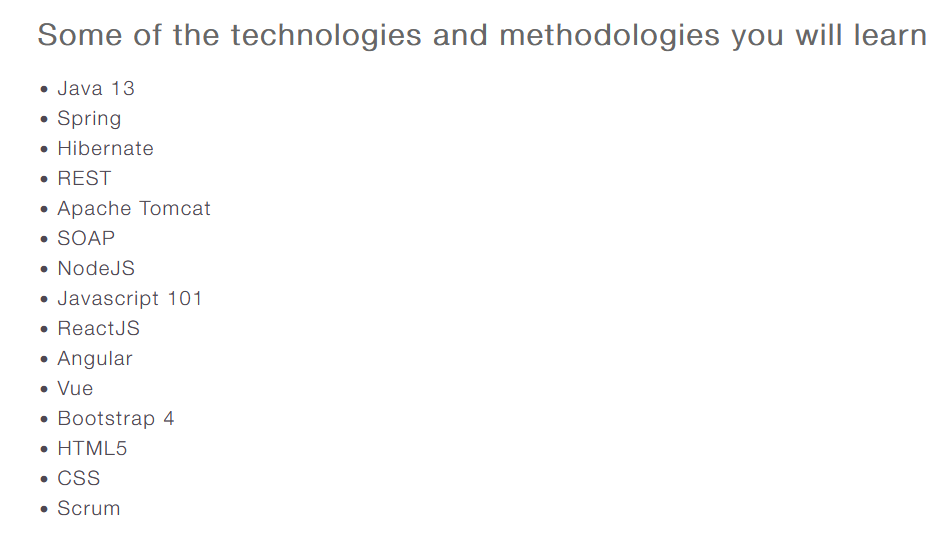
[**Bilag**](#_zckrrp1lwzux) **14**

[Schedule (typical day)](#_zi4gp5m1uuri) 14

# 1. Introduction

This document contains notes from my time at Experis Academy. My job title is “Software Udvikler”, however the first 3 months will pass with making me a “Fullstack Java Developer”. The first 3 months will be intense learning with Noroff - a Norwegian technology school - and preparing for interviews with Experis Academys partners, which for me will most likely be DXC Technology.

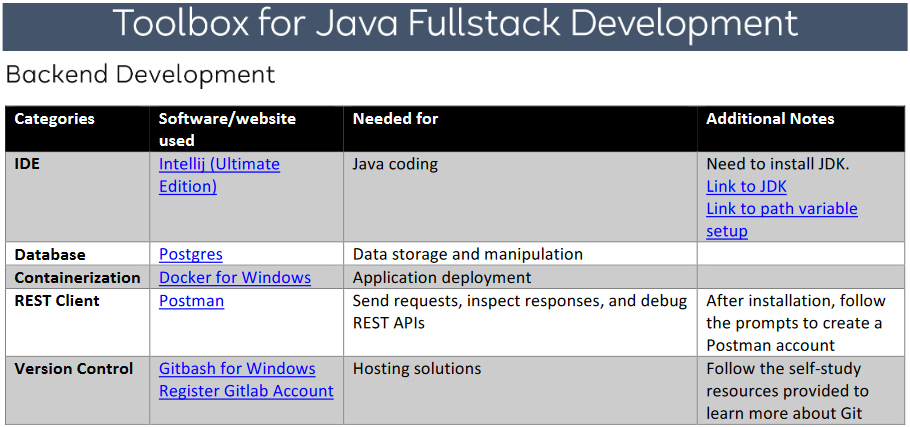
Some of the technologies we will work with in the first 3 months are:

  
Source: <https://experisacademy.dk/Fullstack_Javautvikler>

Note that I will mark keywords with **bold.**

# Tools

I will in this section summarize tools relevant to this document.



Source: <https://lms.noroff.no/pluginfile.php/185579/mod_resource/content/6/Java%20Fullstack%20Development%20Setup.pdf>

Aside from the above I have also used:

* Figma.com (design tool for wireframes and digital prototypes.
* <https://coolors.co/> (helps you with selecting colors for your app)
* <https://devhints.io/> (cheatsheets for multiple languges/frameworks)

# Language Specific

The “Language Specific” section will include any language specific teachings I deem worthy of noting down. This will focus mostly on Java, but also include parts about JavaScript and its frameworks.

## JavaScript

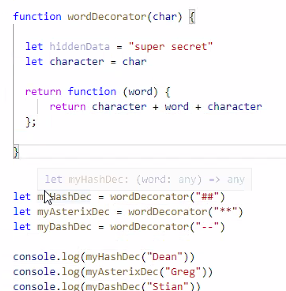
This is what we learned in the Noroff JavaScript Fundamentals.

1. Write simple statements using variables, values and operators
2. Modularize your code with functions
3. Use objects to store key-value pairs
4. Use collections to store and retrieve items in memory
5. Use conditional statements to choose which parts of your code to execute
6. Use looping constructs to iterate through data in collections
7. Use higher order functions to take other functions as argument and return functions as the result
8. Dynamically add, remove, change or read HTML elements
9. Use prototypes to emulate object oriented concepts in JavaScript
10. Manipulate the DOM using browser APIs
11. Use design patterns to solve common problems in a consistent manner
12. Use ES6 abstractions to write elegant and concise JavaScript code

* Serve HTML, CSS and JavaScript files using Express.js and Node.js

### Closures

A closure is **a function that returns a function.** It’s main use is to ‘hide’ parts of the code. Used in the Module design pattern[[1]](#footnote-1).

  
Source: class demo lesson 1.2

### Higher Order Function (HOF)

A HOF takes one or more function parameters. This means you can extend/change the functionality of the HOF depending on the function(s) that is passed as an argument.

All the array dot functions are HOFs. Such as **array.map**, **array.filter**, **array.reduce**, **array.forEach**.

### Foreach in JavaScript (equivalent to C# foreach)

**fo**r let number **of** numberArray{

// do something for every number in numberArray

}

### Callbacks, Promises, Async

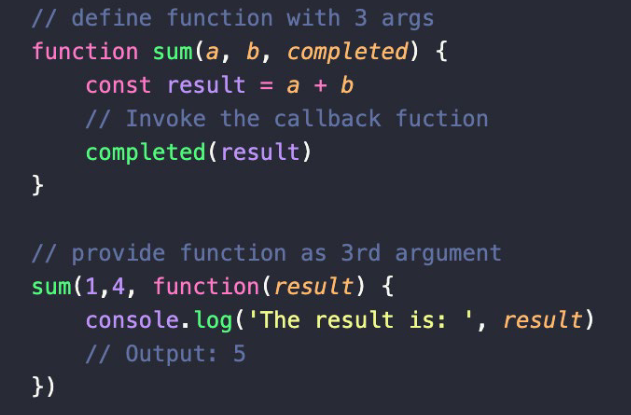
A callback is a scheduled function that runs after another function has completed.

Callbacks are used for running things asynchronously.

“We can pass a function as a parameter and when "result" is ready the callback function is run.

• We still must wait, but we can do things in the meanwhile.

• We can wait for multiple things asynchronously”[[2]](#footnote-2)



Source: Lesson 1.3\_Fundamental JS Coding Constructs.pdf

Callbacks are the first way to make JS code run “asynchronous”. Another way is using Promises.

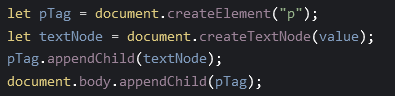
A Promise is a type of object that allows us to request information and handle the result once the information is available. To use a Promise you need to use either async or .then and .catch blocks. Promises are typically used for API calls.

### Bind

Bind is used to specify which context something is run in. This could be used, for example, to specify what the **this** keyword is outside its original context.

### Appending HTML to the DOM

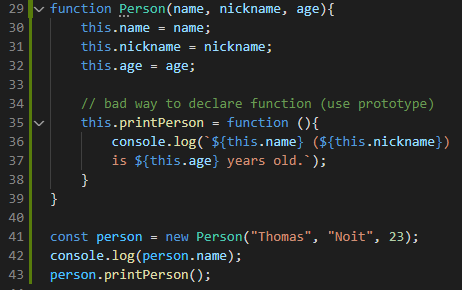
To add HTML elements to the DOM without messing with the HTML file you can do this in JS:



### Functional Constructors

Functional constructors try to similate a class constructor known in other languages, such as C#. JavaScript is not made for OOP, but functional constructors works towards OOP.

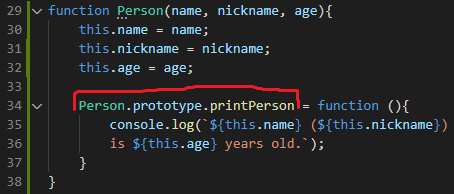
Example of functional constructor:

  
Functional constructor (and how to use it on line 40+)

The “bad way” mentioned in the example duplicates the code in the function to all **new Person** declarations. To get around this you use the prototype object.

### Prototype (objects)

In JavaScript all objects receive a prototype object on creation. The situation mentioned in [Functional Constructors](#_a6vgws8kl56u), where a functions code is duplicated in ALL **new** objects can be remedied by using prototype like below:



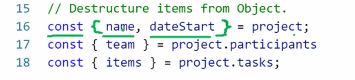
In the example above, you share the function code across the constructors without duplicating it.

When you console log an object like this, the method is attached to the prototype in the object. If you do it without prototype it takes up a lot of memory.

### [Destructuring](https://dmitripavlutin.com/javascript-object-destructuring/)

Destructuring is a way to copy (or reference) select items out of an object into a new variable.

The syntax waries depending on what you’re destructuring. An example can be seen below (functional constructor destructuring):



### Spread operator

const newArray = […myArray]; // clones myArray items into newArray

It can also be used to copy values from an object into a new object with more values.

### Node.js, NPM, Express

Extremely quick overview of Node.js, NPM and Express.

Node.js is a different environment than the browser, where you can run JS code. It needs to be installed on your pc to be usable. When code is run here, the code does not have access to prototype, window object etc. from the browser. Node.js is backend javascript.

NPM and Express use Node,js to function.

NPM stands for Node Packet Manager. It is used to retrieve and use code made by others. One such package is Express.

Express allows for quicker development of JavaScript APIs.

## Vue (JS framework)

Vue is one of the major JavaScript frameworks, as of 2022. This chapter will go through a common setup as well as some important functions in Vue, such Components Vue.

### Setup

I will cover how to install Vue here and name a few packages commonly used to extend Vue beyond its default functionality.

To install Vue you need Node.js and NPM.

Thereafter you just write: **“npm install -g @vue/cli”**

This install Vue and its CLI (command line interface). An alternative to Vue-cli is Vite.

You can create a project with Vite with: “**npm init vite@latest”**

Commonly used packages used are:

* **Vue Router (Version 4+ is needed to use Vue 3)**
* **Vuex 4 (state mangement)**

### Basics of Vue

Generate a project with Vue-Cli or Vite.

In that project, main.js is the JavaScript entrypoint, where Vue is setup to run.

Routes and other global stuff can be put here.

Then you components. Vue components have the extension “**.vue”.**

Inside the components you can use the“magic” of Vue.



### State Management (Emit, Props)

emits are for sending data upstream (up to parent compoents)

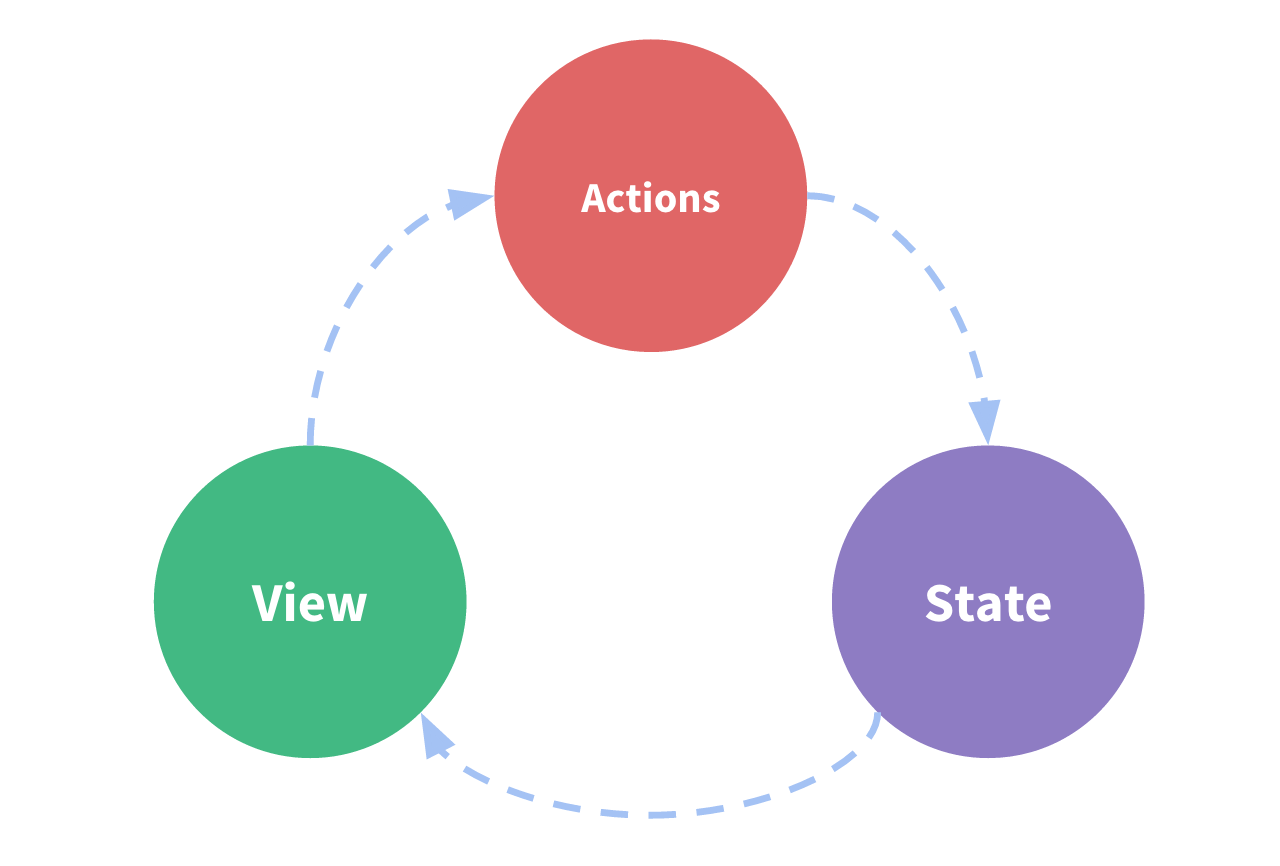
props are for sending data downstream (down to child components)

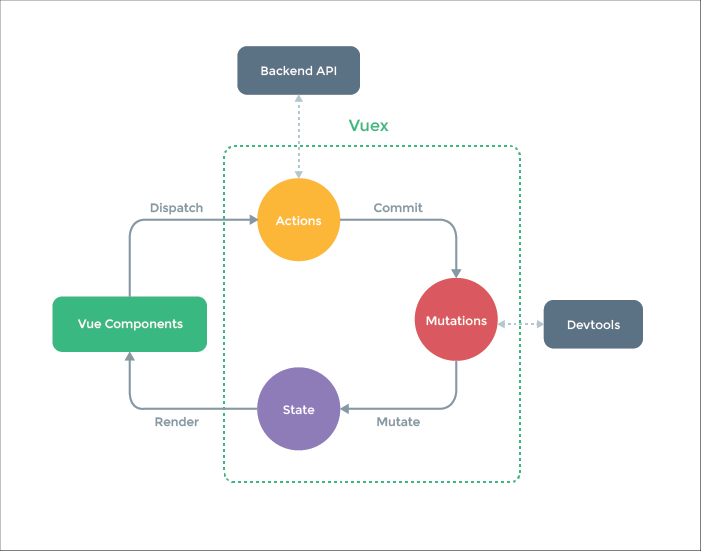
It is done through defineEmits and defineProps methods, which are available in all Vue components.

### Vuex (State Management for Medium-Large projects)

Vuex is a npm package that implements the Flux pattern to handle state management for Vue. It solves the “prop drilling” problem and ensures that the state can only be mutated predictably.

Vuex helps us deal with shared state management with the cost of more concepts and boilerplate. It's a trade-off between short term and long term productivity. It is generally used for medium or large projects.

  
Source: <https://next.vuex.vuejs.org/#what-is-a-state-management-pattern>



Source: <https://next.vuex.vuejs.org/#what-is-a-state-management-pattern>

#### Word definitions

* **Getter** (**computed** properties for the **store**)
* **Mutation** (synchronous functions that changes the state in the Vuex store. Never async)
* **Action** (runs multiple mutation functions. Can be async.)

#### How to use it

**Getter**



Getter example. Source: <https://next.vuex.vuejs.org/guide/getters.html>

INSERT Getter example (used in a component)

**Mutation**

INDSÆT 2 EKSEMPELER

**Action**

INDSÆT EKSEMPELER

### Router

Routing in Vue needs the extension **Vue Router**. It is setup in the javascript entrypoint (commonly main.js).

## React (JS framework)

## Angular (JS framework)

# General Purpose

The“General Purpose” section will include any general purpose information/teachings, such as version control with Git, that is not related to a specific programming language.

## Git

<https://www.conventionalcommits.org/en/v1.0.0/> A convention for git commit messages.

## Docker

This chapter is missing.

## Security Theory (OAuth2)

This section will detail theory about handling login, more specifically with OAuth2.

When dealing with user data you need to do it in a secure way. One of the easiest ways to do it securely is using OAuth2.

### OAuth2 terminology

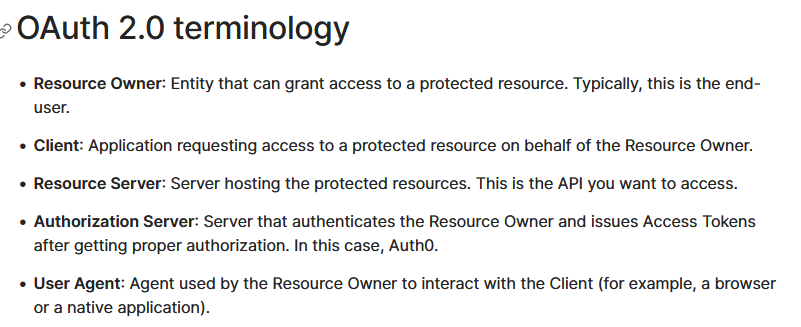
OAuth2 is a standard protocol that details how to **authorize[[3]](#footnote-3)** someone or something in a secure way. To explain how this works a few key terms need to be defined:

The **Issuer** handles the login & data, so we – the developers – don’t have to worry about it, aside from some configuration. An Issuer could for example be: Keycloak, Google, GitHub and many others.

Then we have the **RP**, which stands for **Resource Provider** or **Reliant Party** (they are synonymous). This could, for example, be a REST API. The RP accepts requests from the **Client**, which it checks with the **Issuer** to ensure the client is authorized access a resource.

Lastly we have the **Client**, which is the place we need to login to, for example a web application.

These terms are also known under other names and are used to describe **OAuth2 flows**:



Source: <https://auth0.com/docs/get-started/authentication-and-authorization-flow/which-oauth-2-0-flow-should-i-use>

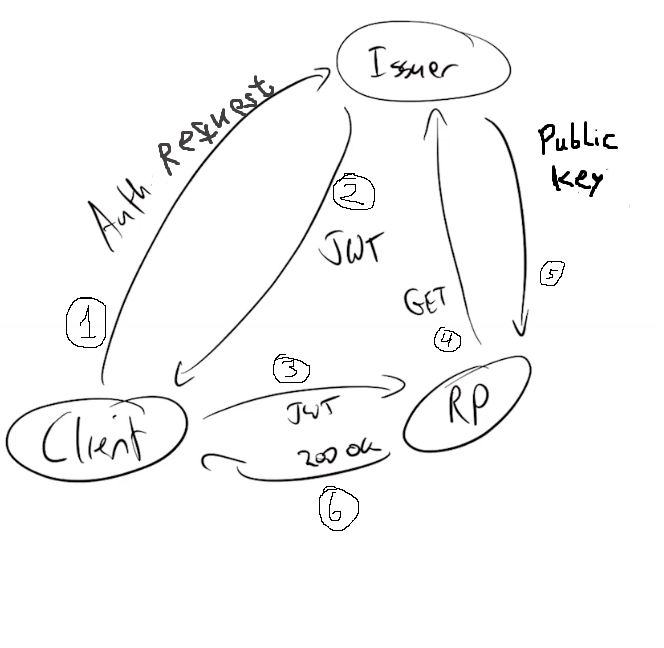
With terminology out of the way we can detail two of the most common OAuth2 flows:

### OAuth2 Flows

An OAuth2 flow describes how communication between the different parties (Issuer, Client, RP) happens. There are many different flows, but here I will describe two of the most common flows.

#### Implicit flow (less secure, but simpler)

I will here show a diagram of how the implicit flow works:

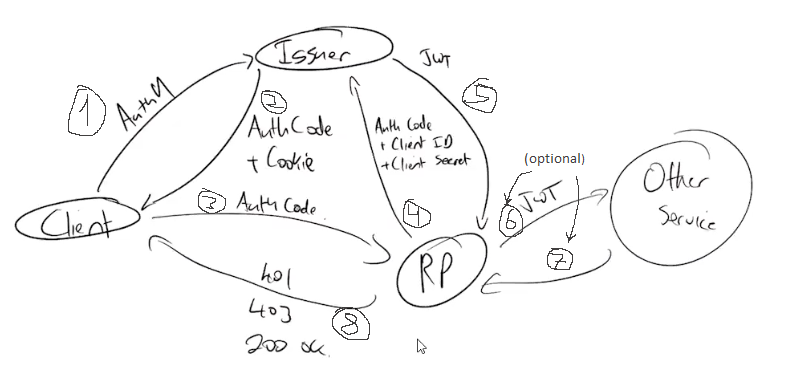


The JWT, which the client receives, can be stolen by “hacker observers” and use it to impersonate the owner of the JWT. This is what makes the Implicit Flow less secure.

#### Authentication code flow (more secure, but more configuration)

This flow exists in variants. To use it in a SPA (Single Page Application) you need Authorization Code Flow with Proof Key for Code Exchange (PKCE).

I will now show a diagram of how the auth. code flow works:



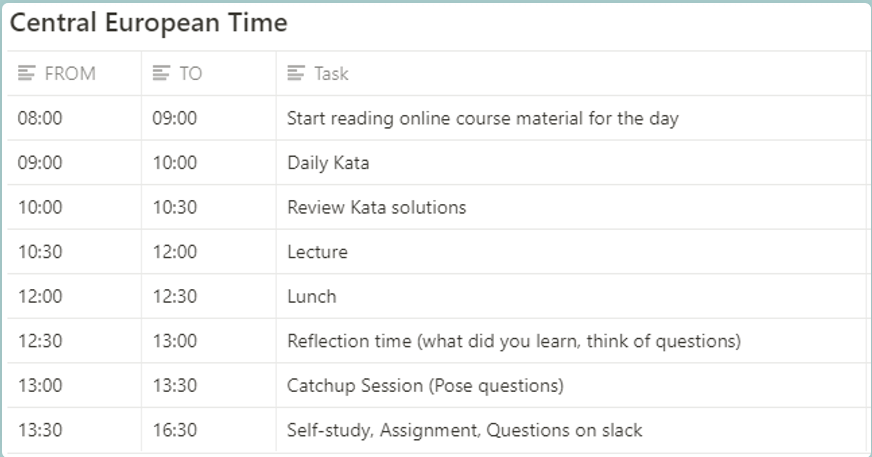
The main difference between the Implicit Flow and the Auth. Code Flow is that the JWT is not exposed on the Client, but instead uses an Auth code + a cookie.

### OIDC (OpenID Connect)

OIDC is built on top of OAuth2 to provide Authentication. OIDC provides access to basic user information, without the need to store it. It embeds an Id token into a JWT token with the user info requested.

# Bilag

## Schedule (typical day)



1. <https://javascript.plainenglish.io/data-hiding-with-javascript-module-pattern-62b71520bddd> [↑](#footnote-ref-1)
2. Lesson 1.3\_Fundamental JS Coding Constructs.pdf [↑](#footnote-ref-2)
3. Authorize meaning: you are authorized to do something. Contrary to Authentication = you are who you say you are [↑](#footnote-ref-3)