jamk

Learning Diary

Modern Software Development (YTSP0100), Juho Pekki

Jari Pennanen

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

Contents

1 Introduction

I have made my living as a web programmer from late 90’s. First as an employee, and between 2007 - 2019 as a private entrepreneur. Since 2020 I’ve been partly unemployed, attended couple IT courses and lastly attended the Full Stack Software Development program at JAMK. It seems difficult to get a new job at this age (56), so my goal is simply to get back to the job market – hopefully with the help of this education.

I didn’t attend the live lessons this year, but attended some of them last year and watched the lesson recordings. While watching, made some notes in my own words (mainly topics 2 – 3 in this document). The recordings were also most helpful when completing the mini projects.

2 Frontend Frameworks

2.1 JavaScript

JavaScript (JS) is in fact a marketing name, and the actual name of the language is ECMAScript. JavaScript was founded in 1997 by Brendan Eich. JS was modernized in 2015 with the ECMAScript version 6. The new definition introduced classes, arrow functions, generators, promises and so on. Current (2022) version has features like private methods and fields, .at() for indexable values, module features, ?? operator and globalThis.

Knowing the ECMA standard is important for making code which is easily portable to different platforms.

2.2 TypeScript

Type Script is an extension of JavaScript, including, among the other things, type annotations and Generic classes. Generic class can contain any types of variables and the variables have to be defined when instantiating the class. This makes the class highly versatile.

Unlike JavaScript, TypeScript needs to be compiled before actually running it. Build tools include Browserify, Grunt, Gulp and Webpack.

2.3 Dart

Dart is a programming language being developed by Google. The same program code can be used to compile programs to multiple platforms like native code in Android and iOS, or JavaScript. Dart includes many features like interfaces, abstract classes, generics and type inference.

2.4 Web Client Frameworks

Most popular all-time frameworks are React.js (41%), jQuery (34%) and Angular (26%) and Express (23%), which is used in the server side. The most popular Front End frameworks today (2022) are listed below.

2.4.1 React

React is currently the most popular framework. React is designed by Facebook. It includes features like component based architecture, virtual DOM and it has it’s own HTML extension called JSX. React Native is a version of React that can be used to build native applications for Android and iOS.

2.4.2 Angular

Designed by Google, Angular is the 2nd popular frameworks in 2022. It is a complete rewrite of previously existed AngularJS. Angular recommends using TypeScript with it. Angular is used to build Single Page Applications (SPA’s). It includes features like templating, modularization ona RESTful API handling.

2.4.3 Vue

Vue is an open source framework and it’s the 3rd popular in 2022. It’s being developed by Evan You and a team of voluntary developers. It’s a model-view-viewmodel framework for building UI’s and SPA applications. Vue components extend basic HTML elements to encapsulate reusable code (Vue.js, 2022).

2.4.4 Ember

Ember is a newish alternative of frameworks. It follows Convention over Configuration (CoC), and the Don’t Repeat Yourself (DRY) principle. It focuses to building ”ambitious” web applications and foresights to future web standards like ES6+, Promises and Web Components. It has it’s own template syntax.

2.4.5 Svelte

Free, Open Source front end compiler dating back to 2016. Svelte compiles HTML code to VanillaJS code that manipulates DOM directly, thus reducing the transferred code and giving better client performance.

2.4.6 Meteor

Full Stack Framework. Supports Distributed Data Protocol (DDP) to allow data transfer in both directions. It does not require use of REST endpoints, it uses publication API instead. It can push data from server side when client side has been subscribed to listen. Meteor can be integrated with several frameworks including React, Vue or Angular.

2.4.7 SolidJS

SolidJS is rather new framework, open sourced 2018. It is similar to React using JSX template language. It is reactive, but does not have virtual DOM, so it’s a kind of lighweight version of React.

2.5 General Overview of Front End Development

2.5.1 W3C

W3C is the organization behind most Web standards. WebAssembly 1.0 (WASM), Web Assembly System Inteface (WASI) and Web Components to name a few.

2.5.2 Web Components

Web Components include custom elements, shadow DOM and HTML templates (template and slot elements). A template can consist of HTML, CSS and JS parts.

2.5.3 MVVM (Model-View-ViewModel) Architechtural Pattern

Graphical user interface (the view) is separated from the back end logic (the model) so that they are not directly dependent of each other. The viewmodel is more model than view, so that it contains data objects and UI related fields like button properties, text labels, selected values etc. For example React, Vue and Angular are JS implementations of viewmodel.

2.6 Features of modern Web application

- UI/UX
- Routing
- Data Fetching
- Rendering (static or dynamic content)
- Updating the application
- Scalability (when there is more traffic)
- Security
- Integration (3rd party services)
- Performance (optimize for front-end users)
- Automated testing (Unit testing, UI testing; automated tools like Selenium) or manual testing

2.7 Things to consider when deploying Web App

- CI/CD pipeline
- Storing app setup and data: configuration data, using SQL or noSQL, Cache database etc.
- Where to run application (Server, Cloud, Serverless)
- Scalability (increasing/decreasing users or team size)
- Updating application (building and maintaining app)

2.8 Excercises

Watched the recording ”Frontend exercise models (09.09.2022)” and noted that because I’m previous year’s student, didn’t have access to the repo with

excercises. Skipped them for now because the code seemed rather familiar and excercises are not mandatory.

3 Modern Back-End Development

Back end API’s and Back end frameworks (server side frameworks). Most popular back-end frameworks 2022 are Laravel, Django, Flask, ExpressJS, Ruby on Rails and Spring. Dated listing is available at https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2022/.

3.1 GraphQL

Supports both server and client out-of-the box. GraphQL query sent to the server returns only the fields required and nothing extra. The client can control exactly what data it gets from the server. This makes the queries fast and results predictible.

Apollo Server is an open-source, spec-compliant GraphQL server which is compatible with any GraphQL client. It is the best way to build a GraphQL API that can use data from any source.

Apollo Supergraph is ”the key idea of the Supergraph is the graph of graphs. It’s how these individual graphs that people have been building come together into a new layer of the stack — a different way of building applications” (Apollo GraphQL releases its Supergraph, 2022).

3.2 Serverless and Function as a Service (FAAS)

Serverless refers to a service where server is fully virtualized from the user’s viewpoint. FAAS is a subset of that, a function that ”lives” in a cloud and gets triggered by some event. FAAS providers include for example Amazon AWS, Microsoft Azure and Google Cloud. FAAS can be implemented also locally for example with node.js, faasjs or OpenFAAS Cli.

3.3 WebSocket

The WebSocket API makes it possible to open a two-way communication channel between user’s browser and server. You can send requests to server and receive responses without having to poll the server for reply. The WebSocket call address starts with ws:// (connects on http) or wss:// (connects on https). Either client or server can close the WebSocket connection.

3.4 Socket.io (with WebSocket)

Socket.io is node.js based. With it, server can push messages to client. It’s a bi-directional channel between client and server.

3.5 Progressive Web Application (PWA)

Progressive Web Applications are built and enhanced with modern APIs to deliver enhanced capabilities, reliability and installability while reaching anyone, anywhere, or any device with single codebase.

”A progressive web application (PWA), commonly known as a progressive web app, is a type of application software delivered through the web, built using common web technologies including HTML, CSS, JavaScript, and WebAssembly. It is intended to work on any platform with a standards-compliant browser, including desktop and mobile devices. (Progressive web app, 2022)”

3.6 Reactive Programming

Reactive Programming is a design paradigm that relies on asynchronous programming logic to handle real-time updates in otherwise static content. It provides automated data streams to handle data updates.

A Stream is sequence of ongoing events ordered in time. Streams can emit three different things: a value, an error or a ”completed” signal.

Events are captured asyncronously. The defined functions are observers or handlers. The stream is the subject, or observable beign observed.

3.7 ReactiveX (Reactive Extensions)

Reactive programming deals with data flow and automatically propagates changes via the data flow. This paradigm is implemented via Reactive Extensions.

”ReactiveX (also known as Reactive Extensions) is a software library originally created by Microsoft that allows imperative programming languages to operate on sequences of data regardless of whether the data is synchronous or asynchronous. It provides a set of sequence operators that operate on each item in the sequence. It is an implementation of reactive programming and provides a blueprint for the tools to be implemented in multiple programming languages. (ReactiveX, 2022)”

3.8 BaconJS

A small functional reactive programming lib for JavaScript. You can for example replace nested for-loops with functional programming concepts like map and filter. It supports event streams. Can be used in reactive back-end development.

3.9 FastAPI

FastAPI is a modern, high-performance web framework for building APIs with Python 3.6+ based on standard Python type hints.

3.10 Server-Side languages (2022)

The most used language is SQL, followed by Java, Python, JavaScript and C#. Current ranking can be seen for example in https://spectrum.ieee.org/top-programming-languages-2022.

3.10.1 JavaScript (with node.js)

Breakthrought for JS as back-end programming language was 2009 when Ryan Dahl created Node.js using Chrome’s JavaScript runtime v8 and c++ libraries. It includes Event Loop and Asynchronous, is suited for event driven and I/O heavy applications like Web applications, Online games, IoT apps, real-time streaming etc. Lots of npm packages can be found in https://www.npmjs.com/.

3.10.2 Express

De facto standard backend JavaScript framework, including middleware, routing, template support etc. Also supports MVC pattern with View system supporting multiple templating engines.

3.10.3 Meteor

Framework for building full-stack apps. For front-end can use Meteor’s own template engine or some other supported framework like Angular or React. Has integrated JavaScript stack to integrate other technologies like MongoDB and React easily.

3.10.4 Koa

Aimed to Web apps and APIs. It has lightweight core and better performance compared to express.js. It offers integrable packages (middleware). Supports async/await an other modern ES features.

3.10.5 Gatsby

Gatsby.js is a server-side React framework that is focused on generating static pages and content from various external sources at build time. It has GrapQL data layer support. Includes also client side support.

3.10.6 Other backend JavaScript frameworks

- Nuxt.js (used with VUE)
- Nest.js (used with Angular)

- Fastify
- Hapi
- Deno (Build in Rust, supports TypeScript and JS)

3.11 Server-Side Languages

3.11.1 GoLang

Go is statically typed, compiled programming language, syntatically similar to C, but with some extra features. Go is a language for creating simple but efficient Web servers and Web services. It provides a built in HTTP package.

3.11.2 Rust

”Modern C language”, developed at Mozilla 2006. Rust is a multi-paradigm, general-purposet programming language. Rust emphasizes performance, type safety and concurrency. Rust enforces memory safety (all references have to point to valid memory).

3.11.3 Python (based frameworks)

Django (2005), a high-level Python Web framework which is based on **MVC** (**Model-View-Controller**) architecture. It consists of Object Relational Mapper (**ORM**), Relational Database (**Model**), a system fro processing HTTP requests with a web template system (**View**) and a requlat expression based URL dispatcher (**Controller**).

Flask (2010), A micro framework written in Python that does not require particula tools or libraries. It supports extensions to add features, unit testing and Restful API routes dispatching. It has built-in web server and debugger.

3.11.4 Kotlin

Kotlin is a cross-platform, statically typed, general purpose programming language with type inference. It mainly targets to JVM (Java Virtual Machine),

but compiles to JavaScript or native code as well. It is possible to develop Android applications with Kotlin.

3.11.5 Java

Java comes with two versions, Enterprise Edition (EE) and Standard Edition (SE). There are several Java Frameworks like Spring Framework, Vaadin, JPA, Play, Java Server Faces and Vert.X. There’re also lot of Open Source Java tools available.

3.11.6 Groovy

Groovy is compatible with Java Virtual Framework (JVM). Grails, Scala and Play Frameworks have been programmed with Groovy.

3.11.7 Ruby

Ruby on Rails is an open source web app development framework based on Ruby and implies DRY (Don’t Repeat Yourself) and MVC (Mode-View-Controller). It’s best features include easy compilation, test and debug, code re-use, large community, fewer development times and ActiveRecord for ORM implementation.

3.11.8 C# (”c-sharp”)

Developed by Microsoft. ASP.NET is a web framework built on C#. ASP.NET Web Api is a set of components which are built on top of ASP.NET MVC runtime. Web API automatically handles the transport details fo HTTP and exposes the HTTP programming model.

3.11.9 PHP

PHP dates back to 90’s but is still popular. It has lot of modern frameworks like Laravel, Symfony, Phalcon, Yii and Slim (micro-framework).

3.12 RESTful Service

In REST architectural style data and functionality are considered resources and accessed via URIs (Uniform Resource Identificator). Resources are accessed using a simple, well-defined set of operations. The resources are usually available in several formats like HTML, XML, plain text, PDF, JSON, JPEG etc. There are standardized interfaces and protocols to exchange representations of resources between server and client, typically HTTP, but REST does not mandate it.

Metadata about the resource is used to control caching, detect transmission errors, negotiate the representation format, and perform authentication or access control.

All interactions between client and server must be stateless.

Also watched the video ”Restful service example with testing”. Testing this way was new for me. The mentioned Postman is more familiar.

4 Mini Project 1 - Framework

4.1 The assignment

”...(create) demo application (You can select a topic freely) Simple "tutorial app" or more complicated one. Complicated app if you want to develop your skills in this exercise. Presentation slides to present the most important parts and features of the selected Framework and your Demo App..”

4.2 Selecting the framework

I selected GraphQL because:

1) GraphQL is gaining popularity fast and seems like a worthy competitor against REST API, which is currently the dominant method. GraphQL can handle

both back end and front end of the service, whereas REST only handles the back end.

2) I have some experience with GraphQL. Couple years ago I worked 4 months as a developer in a company which used GraphQL in it’s mobile application.

4.3 GraphQL Servers comparison

Most popular GraphQL servers in order of popularity:

1) Express GraphQL
2) Apollo
3) Yoga

First tested Apollo, because it is one of the most popular and seems more versatile than Express GraphQL and is well supported. (Most Popular GraphQL Servers, 2020; The most popular GraphQL Servers and Clients for Node.js, 2021; How to Choose the Best GraphQL Server for Your Next Project, 2021)

4.4 Installing Apollo Server and setting up tutorial app

Followed the instructions at ”Get started with Apollo Server (2022)”:

Step 1: Create a new project
Step 2: Install dependencies
Step 3: Define your GraphQL schema
Step 4: Define your data set
Step 5: Define a resolver
Step 6: Create an instance of ApolloServer
Step 7: Start the server
Step 8: Execute your first query

Created project folder. Then installed Apollo by applying this command, which also inits the application by adding package.json file and node_modules subfolder:

$yarn add @apollo/server graphql

After that followed the steps 3 – 8 of JavaScript version of the tutorial. However, when starting the server got an error:

$ node index.js

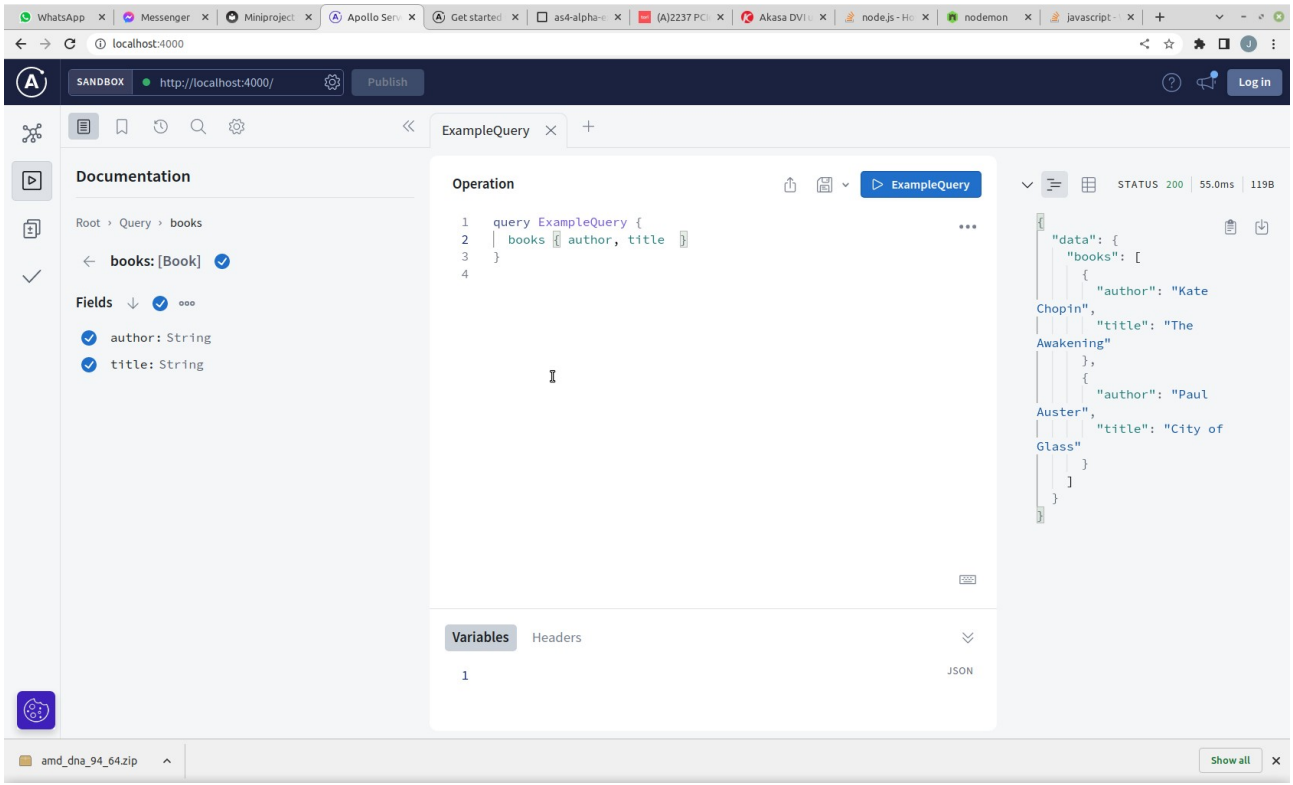
/home/jp/Desktop/FullStack/Modern Software Development/GraphQL/index.js:1

import { ApolloServer } from '@apollo/server';

^^^^^^

SyntaxError: Cannot use import statement outside a module

Managed to fix the error by renaming index.js to index.mjs e.g. changing JavaScript file to ES6 module file, which has a bit different syntax and which was actually used in the example. (”What is the difference between .js and .mjs files?”, 2022)



Apollo Server's demo page after installation

In addition, added nodemon to the project for automatic restarts after code changes.

$yarn add nodemon

#package.json

```
  "dependencies": {
     ...
     "nodemon": "^2.0.20"
  },
 "scripts": {
     "start": "nodemon index.mjs"
  }
```

Using Apollo for the assignment seemed like a good idea at this point.

4.5 Creating repository in GitLab

After setting up the Apollo demo needed to push the code to JAMK LabraNet GitLab server. Created project ”Modern Software Development” via web interface. Tried to add my local files to repo with the instructions that appeared on page after creating the project, but **following 2 lines caused errors**:

…

git init –initial-branch=main

git remote add origin git@gitlab.labranet.jamk.fi:P1408/YTSP0100.git

…

So it looks like the instructions on GitLab web UI are outdated. Found working instructions from here and got the job done: https://gitlab.labranet.jamk.fi/AD1223/ohjelmoinnin_perusteet/-/blob/main/README.md

4.6 Building the application

4.6.1 Watching Video: GraphQL Example (Juha Peltomäki)

- In the video we’re using JavaScript to execute GraphQL. The examples are based on express-graphql package.
- It’s best to start learning with minimial schema to get the server up and running, and add features later.
- When using GraphiQL (graphical UI in browser), it will show hints in query builder about available parameters
- In the end of the video is an example of adding data with GraphQL (=mutation). See https://gitlab.labranet.jamk.fi/YTSP0100/js-exercises/-/blob/main/graphql-samples/lang_codes-graphql-services.js

After watching the lecture video decided to abandon using Apollo in my assignment and use Express GraphQL instead, because it seems simplier and there are great example codes in course repository.

4.6.2 Designing Schema

During my studies I’ve created a React Native mobile application whose purpose is to do some things in mobile phone in a simplified way and supported by speech recognition. Currently the app does not have a backend,

but logging how users use the application would be helpful for further development. So I'm going to create the backend with database here. First thing to do is to design the database structure.

I want to keep the users anonymous for security reasons, but the application needs to be able to combine requests from same origin. The used device can be identified with DeviceID which is unique for all mobile phones and which is available through React Native programming. Another option would be to label each copy of the application with unique id.

Here's a simple database design for the assignment:

Device
---
device_id* (varchar) // all mobile phones have individual id
created_at (timestamp)

Log

Id* (autonumber)
device_id
data (text) // Any data in dedicated log format
created_at (timestamp)

4.6.3 Building the app

Created a simple GraphQL back end based on tutorialspoint.com example (https://www.tutorialspoint.com/graphql/graphql_quick_guide.htm) and it works so that Devices and Logs can be created and queried. Editing log files is usually not required. A flat-file database system notarealdb was used for database because it was fast and simple to implement. It can be replaced with more serious DB later if needed.

Then started to implement the front end, using my previously-existing mobile application created in React Native. However, the old code didn’t compile any more, possibly due some update to some Node package. Here’re the errors:

$ yarn react-native run-android

...

WARNING:: Please remove usages of `jcenter()` Maven repository from your build scripts and migrate your build to other Maven repositories.

This repository is deprecated and it will be shut down in the future.

...

WARNING:: The specified Android SDK Build Tools version (26.0.3) is ignored, as it is below the minimum supported version (30.0.2) for Android Gradle Plugin 4.2.0.

Android SDK Build Tools 30.0.2 will be used.

To suppress this warning, remove "buildToolsVersion '26.0.3'" from your build.gradle file, as each version of the Android Gradle Plugin now has a default version of the build tools.

...

FAILURE: Build failed with an exception.

** What went wrong:*

Execution failed for task ':mobile-assistant:checkDebugAarMetadata'.

...

BUILD FAILED in 13s

After several hours of try-and-error found the solution. Needed to add these lines to build.gradle:

---

def REACT_NATIVE_VERSION = new File(['node', '--print',"JSON.parse(require('fs').readFileSync(require.resolve('react-native/package.json'), 'utf-8')).version"].execute(null, rootDir).text.trim())

```
allprojects {
    configurations.all {
        resolutionStrategy {
            // Remove this override in 0.66, as a proper fix is included in react-ative itself.
            force "com.facebook.react:react-native:" + REACT_NATIVE_VERSION
        }
    }
}
---
```

(https://github.com/facebook/react-native/issues/35229)

In general, the countless Node package versions and settings in build.gradle are endless source of compatibility problems and I've spent lot of hours debugging them. After getting some package to work another then doesn't. Usually it's a good idea to use newest versions of everything but that has also failed in some occasions and it is very frustrating.

ERROR Invariant Violation: Could not find "client" in the context or passed in as an option. Wrap the root component in an <ApolloProvider>, or pass an ApolloClient instance in via options.

The key fixing this was using the client object directly.
https://stackoverflow.com/questions/55201963/graphql-mutation-invariant-violation-must-contain-a-query-definition

WARN Possible Unhandled Promise Rejection (id: 4):
ApolloError: Response not successful: Received status code 400

Very difficult to debug because there is no proper error message – the message just tells that the GraphQL query did not succeed on server.

Tried to tackle the error by testing several logging methods on server side, morganBody being the best so far. It displays detailed information about the request but not the GraphQL response. After comparing the requests from Apollo cient and Graphiql UI the request seemed structurally different, which led me to think that the problem arises form incompatibility betwneen Apollo client and Express Graphql server. So the easiest fix seemed to be installing Apollo also on server side, which I then did. Still had some errors like following which proved to be a bug in GraphQL:

```
 ---
[nodemon] starting `node server.js`
./node_modules/graphql/jsutils/devAssert.js:12
    throw new Error(message);
    ^
Error: Body must be a string. Received: undefined.
---
```

(https://github.com/nestjs/graphql/issues/742)

Finally, after lot of try-and-error got it working.

5 Mini Project 2 - Design an RESTful Web service

Installed Swagger Editor in my laptop and implemented the design in it. The resulting YAML file can be found in my GitLab repository folder Assignment-2. (https://gitlab.labranet.jamk.fi/P1408/YTSP0100/-/blob/main/Assignment-2/appointments-1.0.yaml)

6 Mini Project 3 – CI/CD

6.1 CI/CD Pipeline

Followed the lecture’s video and did the example labranet-ci.yml. Asked Pekki to enable ”msd” runner to run it. However, an error occurred when running:

Error response from daemon: Get https://gitlab.labranet.jamk.fi:4567/v2/: unauthorized: HTTP Basic: Access denied. The provided password or token is incorrect or your account has 2FA enabled and you must use a personal access token instead of a password. See https://gitlab.labranet.jamk.fi/help/user/profile/account/two_factor_authentication#troubleshooting

---

Checked from User Settings / Account that 2FA (2-factor authentication) is not enabled, so it was not that. After some debugging found out that I had a typo in gitlab-ci.yml where I had $CI_ $CI_REGISTRY-USER instead of $CI_REGISTRY_USER, which then worked and got the example to run succesfully.

My assignment resides in Assignment-3 folder, so need to update some paths. Changed Settings / CI/CD / CI/CD configuration file: Assingment-3/gitlab-ci.yml

6.2 Kubernetes

Installed Google Cloud SDK and KubeCtl to my laptop. Then followed the instructions in Kubernetes 2021 video, until got the following error:

---

jp@Silver:~/.../Assignment-3$ g
cloud container clusters get-credentials cluster-1 --zone europe-north1-c
Fetching cluster endpoint and auth data.
CRITICAL: ACTION REQUIRED: gke-gcloud-auth-plugin, which is needed for continued use of kubectl, was not found or is not executable. Install gke-gcloud-auth-plugin for use with kubectl by following https://cloud.google.com/blog/products/containers-kubernetes/kubectl-auth-changes-in-gke
kubeconfig entry generated for cluster-1.

Fixed the error by installing the required plugin with command:
gcloud components install gke-gcloud-auth-plugin

(https://stackoverflow.com/questions/64313277/gitlab-ci-cd-job-showing-error-could-not-read-json-file-when-im-tring-to-setu)

Another error:

- kubectl create secret docker-registry jamkgitlab --docker-server=gitlab.labranet.jamk.fi:4567 --docker-username=$GITLAB_USER_LOGIN --docker-password=$GITLAB_TOKEN –docker-email=$GITLAB_USER_EMAIL

error: either --from-file or the combination of --docker-username, --docker-password and --docker-server is required
---
Fixed this by changing the predefined variable names as follows:
- kubectl create secret docker-registry jamkgitlab --docker-server=gitlab.labranet.jamk.fi:4567 --docker-username=$CI_REGISTRY_USER --docker-password=$CI_BUILD_TOKEN –docker-email=$CI_USER_EMAIL

(https://stackoverflow.com/questions/69790883/strange-error-while-creating-imagepullsecret-with-kubectl-in-gitlab-ci-job)

A bit unclear step was *”# how to apply a new service&deployment?”*
- In video ”Docker (part II, 26.11.2022.)” it is mentioned that this can be solved on one line. Then figured out that this could be a call to deployment file created in step 4 and the line may just be a reference to that file. Did that and managed to get the job to run successfully – hopefully as it was supposed to.

7 Final comments

Preveiously I had some experience in GraphQL, only minor experience in containers and none in CI/CD automation. It is well possible that I will run to these topics later in worklife so the course seemed necessary to me and was actually interesting. I want to thank Juho Pekki and Juha Peltomäki about the well-thought course materials which were easy to follow.

References

Vue.js (2022). Retrieved November 3, 2022, from https://en.wikipedia.org/wiki/Vue.js.

Apollo GraphQL releases its Supergraph (2022). Retrieved November 7, 2022, from https://techcrunch.com/2022/05/18/apollo-graphql-launches-its-supergraph/

Progressive Web App (2022). Retrieved November 9, 2022, from https://en.wikipedia.org/wiki/Progressive_web_app

ReactiveX (2022). Retrieved November 9, 2022, from https://en.wikipedia.org/wiki/ReactiveX

GraphQL Basics: Pros and Cons (n.d.). Retrieved November 20, 2022 from https://aloa.co/blog/graphql-basics-pros-and-cons

Most Popular GraphQL Servers (2020). Retrieved November 15, 2022, from https://blog.graphqleditor.com/graphql-servers

How to Choose the Best GraphQL Server for Your Next Project (2021). Retrieved November 18, 2022, from https://spin.atomicobject.com/2021/11/19/choosing-best-graphql-server/

The most popular GraphQL Servers and Clients for Node.js (2021). Retrieved November 18, 2022, from https://smartive.ch/blog/the-most-popular-graphql-servers-and-clients-for-node-js

Get started with Apollo Server (n.d.). Retrieved November 20, 2022, from https://www.apollographql.com/docs/apollo-server/getting-started/

GraphQL vs REST – Difference Between APIs (2022). Retrieved November 20, 2022, from https://www.guru99.com/graphql-vs-rest-apis.html

GraphQL Vs. REST: All That You Must Know (n.d.). Retrieved November 20, 2022, from https://www.wallarm.com/what/graphql-vs-rest-all-that-you-must-know

What is the difference between .js and .mjs files? (2022). Retrieved November 21, 2022, from https://stackoverflow.com/questions/57492546/what-is-the-difference-between-js-and-mjs-files

React Native Get Unique ID of Device (2022). Retrieved November 26, 2022, from https://aboutreact.com/react-native-get-unique-id-of-device/