TD2 Advanced Web Programming - Michaël NASS (SE2)

Exercice 1
This file must be loaded from a web server in order to load the JavaScript code. Otherwise, you will encounter the following errors :
- Origin null is not allowed by Access-Control-Allow-Origin (Status code: 0)
- Failed to load resource: Origin null is not allowed by Access-Control-Allow-Origin (Status code: 0)

We'll use the "npx http-server -a localhost" command to run a local server serving our code.

Exercice 2
4 JS files were loaded. It takes about 5ms in average to load a given light JS file. It took 60ms to load all 4 files. The browser downloaded all files in sequence, with an important delay between each download (presumably processing the last downloaded file).

Exercice 3
The browser looses a lot of time to download the files. Because the files are small and separated, the browser will encounter a lot of latency downloading each individual files. An efficient solution would be to bundle all files into a single file. That would reduce the overhead to downloading one file only, and makes the parsing easier for the browser. Tools exists to do that automatically.

Exercice 4
When NPM installs a package, it also installs all its dependencies in the "node_modules" folder. That is why there are way more packages in the "node_modules" folder than in the "package.json" file.

Exercice 5
One module has a severe vulnerability. The "npm audit" command will list all vulnerabilities in the project. Running that command, we learn that the "chart.js" package has a vulnerability. This vulnerability is fixed in version 2.9.4. The suggested fix is outside the dependencies range. Therefore, the terminal informs us that we can run the "npm audit fix —force" command to automatically fix the vulnerability.

Exercice 6
Webpack is used to tackle the problem of slow loading time for JS files. It is a bundler, meaning it will bundle all JS files into a single file. What it does is take all the JS files and their dependencies, and create an "index.js" file containing the code of all the JS files. This allows the browser to download only one file, and parse it only once, reducing therefore the overhead and delay caused by the sequence download.

Exercice 7
The "npm run build" command successfully created a "dist" folder containing the Webpack "index.js" file. This file contains the code of all the JS files inside eval() functions. Each eval() function matches the code of a JS file, on one line. The Webpack generated JS code is not quite human-readable, but will work exactly the same as the original JS files. In order to edit the code, we need to re-run the "npm run build" command after each change.

Exercice 8
We open the index-bundled.html file in the browser and see that the loading time is much faster. Specifically, it took 4ms to load the index.js file. The total time including the overhead is 28ms, which is way faster than the 60ms we had previously.

Exercice 9
We change the webpack.config.js file to edit the "mode" property to "production". Then, we re-run the "npm run build" command. In production mode, the Webpack generated JS code is minified, meaning it is compressed. It is even harder to read for a human, but it will still work exactly the same as the original JS files. The index.js file is only one line long.

Because the files are small, we don't see a big difference in loading time. However, in larger projects, you should see faster loading times in production mode.

Exercice 10
We will try to connect to our local server from a Windows 11 machine. The goal is to try to load the page using the Internet Explorer mode of the Edge browser.

We get an error saying: Syntax error in index.js (1,4)

According to the code, IE11 does not seem to accept the () => {} syntax, which is a well documented issue on the Internet.

This is a syntax that was introduced introduced in ES6, which is not supported by IE11. We therefore need to transpile the code to ES5.

Exercice 11
We will use Babel to transpile the code to ES5. We updated the webpack.config.js file to add the "babel-loader" package, with the file already provided. We import the babel runtime to the index.js file. Finally, we re-run the "npm run build" command.

This time, the Javascript code loads in IE11. But the app is yet to be functional.

Exercice 12
We import two additional required packages in the index.js file. The inserted lines are:
- import 'regenerator-runtime/runtime'
- import 'core-js/stable'
- import 'whatwg-fetch'

We re-run the "npm run build" command.
The app is now fully functional in IE11.

Exercice 13
There are two major issues with transpiling the code to ES5. First, the generated code is way heavier than the original code, leading to longer loading times and more resources used. This can more or less cancel the main benefit of using Webpack.

At some point, it is not worth it to transpile the code to ES5, specifically as browsers are evolving and most of them support the ES6 syntax. Internet Explorer is among the last browsers not supporting ES6. On top of that, Internet Explorer 11 (the last version, originally released in 2013!) is not supported by Microsoft anymore, and Microsoft pushes users to use Edge instead. Transpiling is becoming less and less relevant over time, at least when it comes to ES5.

Second, in addition to the loading time, transpiling the code to ES5 also increases the complexity of the code, even for the newer browsers. This can lead to slower execution time, and potentially more bugs as your code has been transformed. It can also make debugging harder, as you clearly won't recognise the code you wrote.

Note: Because this report was originally contained in an HTML file, I couldn’t take screenshots while making it but ensured the text is as descriptive as it can get.