Lab Week10 Description

For this week's exercise, we will build a web interface on top of the api we created in last week. We will use webpack to bundle our project with all of its dependencies. Then we will deploy it with Heroku.

1. First, let's install webpack with the command **`npm install --save-dev webpack webpack-cli`**.

2. Now, let's update our folder structure as following:

```
|   .gitignore
|   app.js
|   server.js
|   package.json
|   src/
|     | client/
|     |   | index.js
|     | server/
|     |   | api.js
|     |   | api.test.js
|     | logic/
|     |   | greeting.js
|     |   | greeting.test.js
|   dist/
|     | index.html
```

3. We want to create a dist folder to separate the "source" (/src) code from our "distribution" (/dist) code. The "distribution" code is the minimized and optimized output of our build process that will eventually be loaded in the browser. In the dist folder, create an index.html file:

```
<!doctype html>
<html>
  <head>
    <title>Getting Started</title>
  </head>
  <body>
    <script src="./src/index.js"></script>
  </body>
</html>
```

4. Now, let us create the index.js file in the client which will generate some html for our web application.

```
function component() {
  let element = document.createElement('div');

  element.innerHTML = _.join(['Hello', 'there!'], ' ');
```

```
  return element;
}

document.body.appendChild(component());
```

Install lodash with **`npm install --save lodash`** to bundle the lodash dependency within index.js. Then, import the global _ variable with lodash to be able to use _ in index.js.

5. Now, let's add a webpack.config.js file to the root of the project. This config enables webpack to bundle together and create the main.js file to contain every production dependency it needs. The file should look like this (for now):

```
const path = require('path');
module.exports = {
  entry: './src/client/index.js',
  output: {
    filename: 'main.js',
    path: path.resolve(__dirname, 'dist')
  }
}
```

6. Let's add a shortcut to our package.json script `"build": "webpack"` Now, run `npm run build` and webpack should build successfully.

7. Let's add a html webpack plugin to our Webpack config file. It will by default generate its own html file, even though we already have one in the dist folder. You need to npm install with **`npm install --save-dev html-webpack-plugin`**, declare the HtmlWebpackPlugin in the top of the config file and add the following changes to your webpack config object:

```
plugins: [
    new HtmlWebpackPlugin({
        title: 'Greeting page'
    })
],
```

In general, it's a good practice to clean the `/dist` folder before each build, so that only files used in your next build will be generated. Lets npm install a webpack clean plugin: **`npm install --save-dev clean-webpack-plugin`**

Then declare the plugin within your webpack config object the same way as with the html webpack plugin, where the folder to clean is `dist`.

8. We will have to run our server as well as building the application, and we are going to use webpack devServer to be able to have our server running. Install webpack devServer with **`npm install --save-dev webpack-dev-server`**. Add devServer config to your webpack.config object:

```
devServer: {
    port: 3000,
    open: true,
    proxy: {
      "/api": "http://localhost:8080"
    }
},
```

    - Then update the your server.js to use port number 8080.
    - Run the node server and build the project with **`npm run build`**.
    - Now you can access your web application in localhost:3000.
9. Now you should see a static `Hello there!` message on localhost. The next step is to consume the api in our client index file and display a greeting in our web app. We leave that implementation up to you but here's a hint: Use fetch on the endpoint to get the greeting.

Deployment with Heroku

Heroku offers a great step-by-step introduction for deploying node applications on Heroku, if you get stuck somewhere, head over there to check that out.

Deploying Node apps on Heroku

1. Sign up for free on Heroku.com.
2. Setup the Heroku Toolbelt
3. Sign in using the Heroku Toolbelt
4. Using the same repo as in the first part of the assignment; in the root, create a new Heroku app: heroku apps:create
5. Add a script to your project's package.json for Heroku to run when deploying: `json "heroku-postbuild": "webpack -p"`
6. Commit and push to Heroku: `git commit -m "Added Heroku app"; git push heroku master`
7. You should now have a running application somewhere on herokuapp.