Continuous Integration

The main purpose of this exercise is for students to setup a continuous integration (CI) pipeline for their code repository. For this purpose we will be using Circle CI. We will also setup a simple application programming interface (API), using Express.js and test our API using an additional testing library suitable for this purpose, SuperTest. API tests are different from unit tests which we have been focusing on, they are on the integration level, where the API needs to be up and running, when unit tests don't have that requirement.

Part 01: Prepare our project

Start with last week's lab exercise; you should have a git project, with the NPM package.json file and a struture similar to:

```
Week09
  |
  | - .gitignore
  | - app.js
  | - node_modules/
  |     | ... (all the packages)
  | package.json
  | package-lock.json
  | - src/
  |     | greeting.js
  |     | greeting.test.js
  |
```

Using Express.js, we want to take this structure and create a very simple API that has a single endpoint that returns the greeting with the name given, i.e. when a user opens up the web server and goes to the URL [http://localhost:3000/greeting/Diana] they should see something like `{ greeting: "Hello, Diana!" }`.

Writing the API tests

But first, write the test, so once you manage to implement this, you should have a green test. For the test, we'll use SuperTest.

1. Install SuperTest (npm package name: supertest)
2. Create a new file under src `src/api.test.js`

```
// src/api.test.js
const request = require("supertest");
const api = require("./api");
```

```
describe("GET /greeting/:name endpoint", () => {
  it("should return a 200 OK status code", async () => {
    const res = await request(api).get("/greeting/_");
    expect(res.status).toBe(200);
  });
  it("should return the greeting in a object", async () => {
    const res = await request(api).get("/greeting/Diana");
    expect(res.body.greeting).toBe("Hello, Diana!");
  });
});
```

A few things to explain before we continue. First two lines are just imports (nothing new), first importing the SuperTest testing library, and then the `greeting.js` module that we already have. Then we have the `describe(...)` part. In Jest this is used to group together related tests, this part is optional, but as tests grow in number this becomes more convenient.

Then in our test, we use `async` decorator, because the `request` (SuperTest) function call takes time to complete and without us explicitly saying `async` and `await` the execution of the the test would continue before `request` would finish and our test would simply fail (always). We won't spend a lot of time on this, but there are multiple videos and articles on async and await in Javascript (a personal favourite is this Fun Fun Function video on it).

And finally we expect the `res.status` (res being the object returned from SuperTest's request) to be 200 (HTTP status code for OK).

Now that we have a test for our API endpoint we need to make the actual `src/api.js` file. Looking at the tests to guide you in your implementation.

1. Install Express.js as a dependency (npm pagckage name: express)
2. Change `app.js` to use the Express.js dependency, and define an "app".
3. Create a `get` route, using a string (where the string is the name), which calls greeting.js and returns a JSON object with the greeting value and message. See more on [Express routes][express-routes].

```
// src/api.js
const express = require("express");
const app = express();
const greeting = require("./greeting");

app.get("/greeting/:name", (req, res) => {
  // Fill in your code that:
  // 1. set's the status code to 200
  // 2. and returns an object with the greeting

  // Hint: `req.params.name`
});

module.exports = app;
```

Once you have a running API test, you need to change `app.js` to the following:

```
const app = require("./src/api");

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {
  console.log("Server running on port " + PORT);
});
```

Verify everything works with `npm start`, check out your browser: http://localhost:3000/greeting/your_name

Part 02: Continuous Integration with Circle CI

The objective is that the students experiment with setting up a continuous integration pipeline; from a code change to running tests automatically and getting feedback. In addition deploying a working artifact to a staging-like environment is a logical next step.

1. Setup Circle CI for the project

Go to Circle CI

1. Sign up with GitHub
2. Add the repository to Cirlce CI as a project, by going to `https://circleci.com/add-projects/gh/**YOUR_USERNAME**`
3. Click "Set up project", and follow the steps given.
4. You will need to add a folder `.circleci` in the root of the project, and create a `config.yml` file.

Once you have Circle CI setup, changes made to the repository will result in tests being run. Think of adding a status badge for your README.md file.