

Bachelor of Computer and Information Sciences**Contemporary Issues in Software Engineering
Semester 2, 2024****ASSIGNMENT 1A: Worksheet 2 (20% of Ass1A)***APIs with Nest JS and Node, Jest, GitHub Actions***Deliverables and Due dates:**

You are required to complete the Worksheet and keep evidence as you do it by **selectively** taking screenshots of your work, as well as explanations.

Each worksheet should ideally be checked off by the TA by the end of the week's tutorial

This worksheet should be Checked off and Uploaded on CANVAS ideally by end of Tutorial Week 2 – all four worksheets for Assignment 1a are due before week 6, and the knowledge develops cumulatively so don't leave it to the end – that will also make it hard for the TA's to mark and give you feedback.

Introduction

This worksheet will involve the following steps to get you to understand how node.js and nest.js can be used to a backend with API end points. Also testing a React component locally and manually is introduced. GitHub Actions, as the start of an automated Continuous Integration pipeline, are introduced to automate running the test in the code base after a pull request is received.

Overview

1. Learn about TypeScript (<https://www.typescriptlang.org/docs/handbook/intro.html>)
2. Create a backend Nest web server running locally on node.js
3. Create some RESTful API end points to Read (CRUD) data in the backend (from a database eventually)
4. Create a react front end
5. Run some simple jest tests for the front end
6. Automate the tests running for a pull request using GitHub actions

Create a web server using Nest.js running on Node.js

1. Open VS Code and open the CISE_REPOS folder and open a terminal window. Check terminal is in the CISE_REPOS folder. In VS Code create a new folder inside the CISE_REPOS folder called "worksheet2" to store the content of this worksheet. In terminal change to being in this new folder (use the `>cd` command).
2. A traditional way to initialise a NodeJS project is by using the command `>npm init`. This will ask for a number of prompts – you can press `<return>` to accept the defaults. I changed the "entry point" to `server.js` and put my name as the author. A `package.json` file will be created.

```
PS C:\Users\jc\Desktop\TA\ense701_2024\CISE_Repos\worksheet2> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

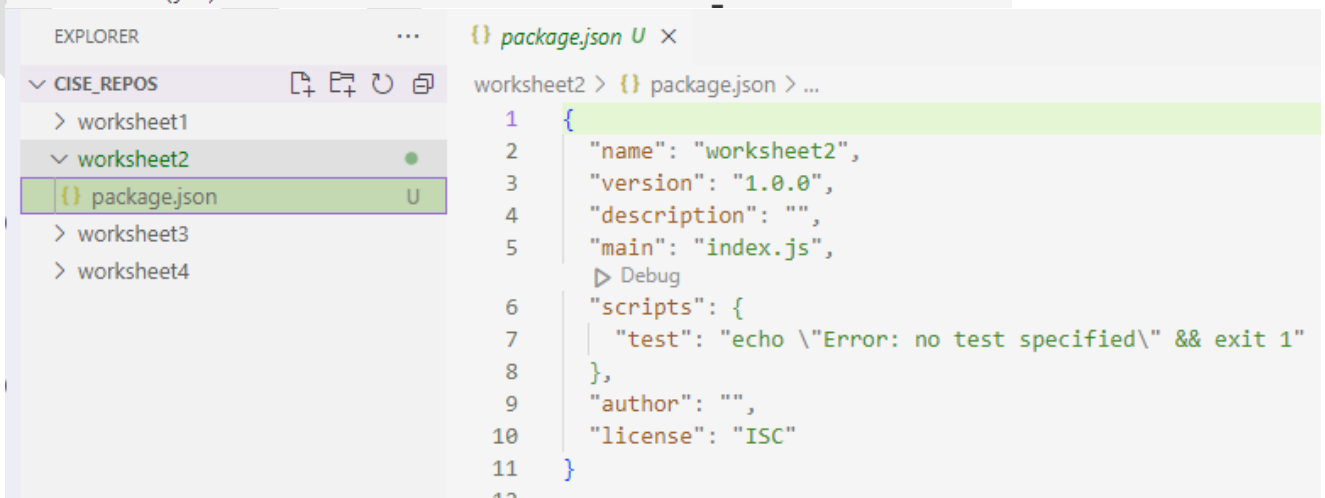
See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (worksheet2) worksheet2
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\jc\Desktop\TA\ense701_2024\CISE_Repos\worksheet2\package.json:

{
  "name": "worksheet2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes)
```



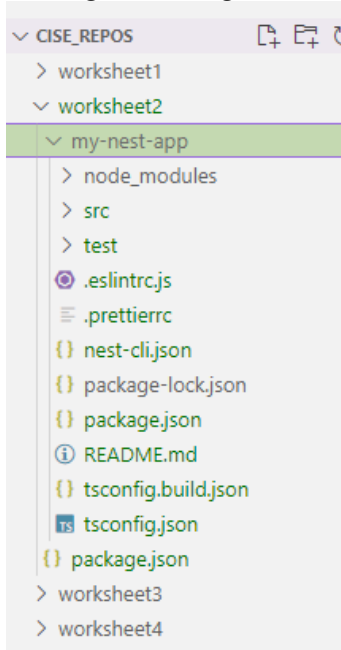
```
1 {
2   "name": "worksheet2",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "",
10  "license": "ISC"
11 }
```

3. However, NestJS also provide an init command for a quick setup. So we can ignore the "npm init" but use "nest new".
4. Ensure you have nest installed with: `>nest -version`

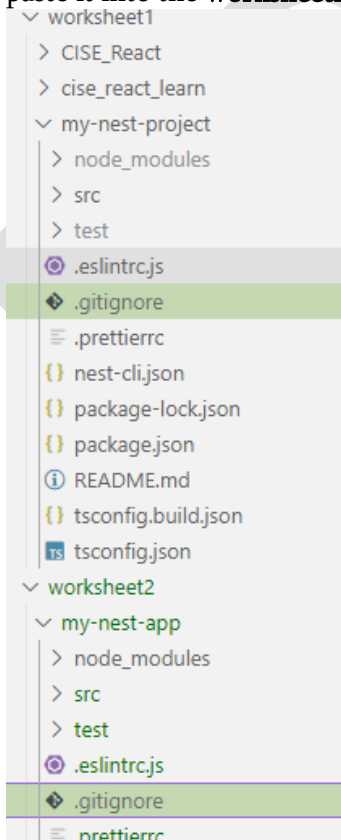
If you need to install nest again place this command into the terminal `>npm i -g @nestjs/cli`.

Now run the command `>nest new my-nest-app --skip-git`
(If you ever forget the commands of nest you can always enter `>nest --help`)

I am skipping the git initialization here using the tag `--skip-git`. We do not need to set this up as we did this last week. You should now have a repo with last week's and this week's work so far looking something similar to this:



Because we skipped the git set up step. We need to manually add the `.gitignore` file. (read more about why we need [this file here](#)) Lets copy the one in week 1 “**worksheet1/my-nest-project**” and paste it into the **worksheet2/my-nest-app** directory:



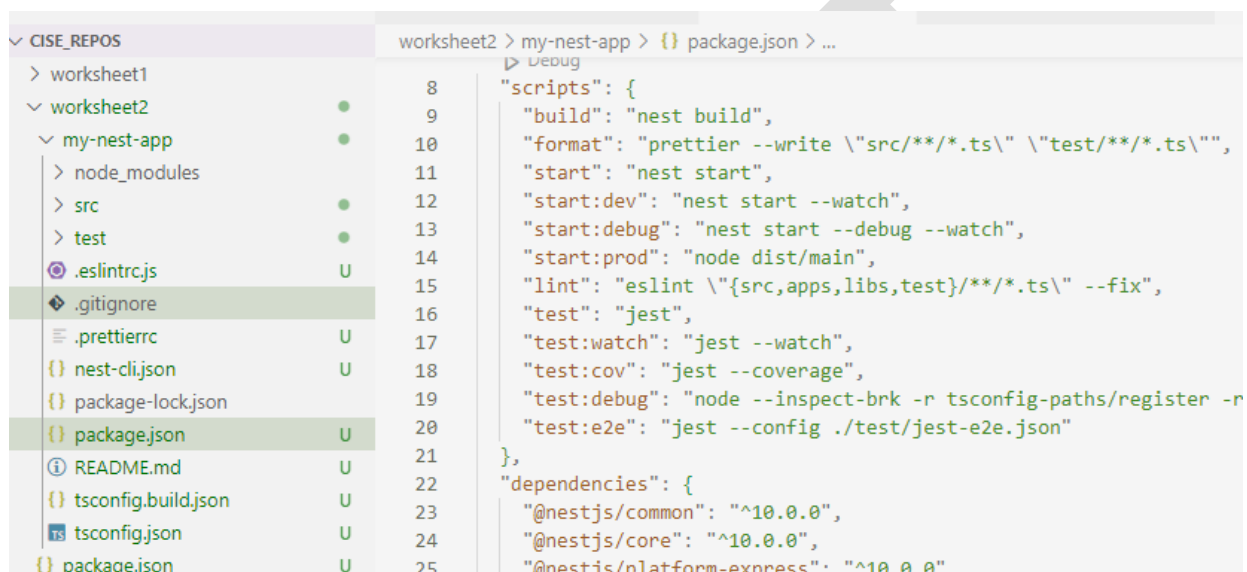
- It's **important** you familiarize yourself with the file structure of the newly generated nest application. Take a look around, open files – explore before continuing! A lot of these files we do not need to edit yet until later in the course so don't be afraid.

- After you have finished exploring, ensure you are in the correct directory:

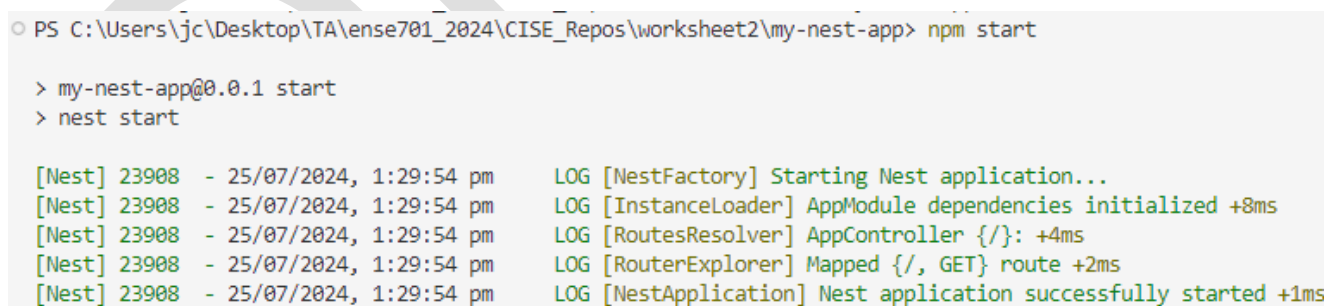
```
> cd .\my-nest-app\
```

SIDE-NOTE: scripts in the package.json file

- Open up the package.json file. Observe all the scripts that have been generated. Try running a few and see what happens with: `>npm run {NAME_OF_THE_SCRIPT_HERE}`




Now you can type `>npm run start` (or `"npm start"` for short) to start the server. Pressing ctrl-c in terminal will stop the server.



Create some API end points and routes

8. Now we need to add an API end point. If we want to receive (or send, or modify) data from our backend or database we need an API endpoint, which is the route that serves up the data. Open up `src/app.controller.ts` (read more about [controllers here](#))



```

EXPLORER
  CISE_REPOS
    > worksheet1
    > worksheet2
      > my-nest-app
        > dist
        > node_modules
        > src
          TS app.controller.spec.ts
          TS app.controller.ts
          TS app.module.ts
          TS app.service.ts
          TS main.ts
          > test

TS app.controller.ts
1  import { Controller, Get } from '@nestjs/common';
2  import { AppService } from './app.service';
3
4  @Controller()
5  export class AppController {
6    constructor(private readonly appService: AppService) {}
7
8    @Get()
9    getHello(): string {
10     return this.appService.getHello();
11   }
12 }
13
  
```

Fig 5

NOTE: We can actually right now see this - ensure the application is running with `>npm start` And then open up a browser and type into the url: <http://localhost:3000/> You should see something on the page!

SIDE NOTE: Http methods for RESTful services

We can use GET (read data from the database), POST (create new data in the database) PUT (update a particular database document) or DELETE (delete a particular database document). Read more about them here: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

SIDE-NOTE: Arrow function notation in javascript / typescript

The javascript arrow notation ("`=>`") is used here to write a function. This is a more concise way of writing js functions and have some interesting properties. To read more about the arrow function read <https://javascript.info/arrow-functions-basics> The excerpt below gives you the idea using the equivalent functional notation, but there is more to it, so it is worth reading about it.

```

1  let sum = (a, b) => a + b;
2
3  /* This arrow function is a shorter form of:
4
5  let sum = function(a, b) {
6    return a + b;
7  };
8  */
9
10 alert( sum(1, 2) ); // 3
  
```

As you can, see `(a, b) => a + b` means a function that accepts two arguments named `a` and `b`. Upon the execution, it evaluates the expression `a + b` and returns the result.

- If we have only one argument, then parentheses around parameters can be omitted, making that even shorter.

Fig 6

Create API end points that read a data array in a file in the backend.

9. Now let's create another API. Firstly, we need some dummy data to use (since we have no database yet). We will create a typescript file with the data in an array and assign it to an object "articles". It will be similar to the database we need with information on articles about TDD and some claims and some evidence which an analyst has extracted from reading the articles. Create a folder called "dummydata"
10. Create a file called "articles.ts" in this dummydata folder. The file should now live: src/dummydata/articles.ts - Copy the code shown in Fig 7.
11. You can read about typescript arrays here <https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#arrays>

```
export const ARTICLES: any[] = [
  {
    _id: "1",
    title: "An experimental evaluation of test driven development vs. test-last development with industry professionals",
    authors: "Munir, H., Wnuk, K., Petersen, K., Moayyed, M.",
    source: "EASE",
    pubyear: "2014",
    doi: "https://doi.org/10.1145/2601248.2601267",
    claim_evidence: [["code improvement", "strong support"], ["product improvement", "weak against"], ["team improvement", "none"]],
  },
  {
    _id: "2",
    title: "Realizing quality improvement through test driven development: results and experiences of four industrial teams",
    authors: "Nagappan, N., Maximilien, E. M., Bhat, T., Williams, L.",
    source: "Empirical Software Engineering, 13(3), 289–302",
    pubyear: "2008",
    doi: "https://doi.org/10.1007/s10664-008-9062-z",
    claim_evidence: [["code improvement", "weak support"], ["product improvement", "weak against"], ["team improvement", "low support"]],
  },
  {
    _id: "3",
    title: "Does Test-Driven Development Really Improve Software Design Quality?",
    authors: "Janzen, D. S.",
    source: "Software, IEEE, 25(2) 77-84",
    pubyear: "2008",
    doi: "",
    claim_evidence: [["code improvement", "strong support"], ["product improvement", "weak support"], ["team improvement", "none"]],
  },
  {
    _id: "4",
    title: "A Comparative Case Study on the Impact of Test-Driven Development on Program Design and Test Coverage",
    authors: "Siniaalto, M., Abrahamsson, P.",
    source: "ArXiv.Org, cs.SE, arXiv:1711.05082-284",
    pubyear: "2017",
    doi: "https://doi.org/10.1109/esem.2007.35",
    claim_evidence: [["code improvement", "weak support"], ["product improvement", "weak support"], ["team improvement", "none"]],
  },
];
```

Fig 7

12. Import the file for our code in app.controller.ts with the line:

```
import { ARTICLES } from './dummydata/articles';
```

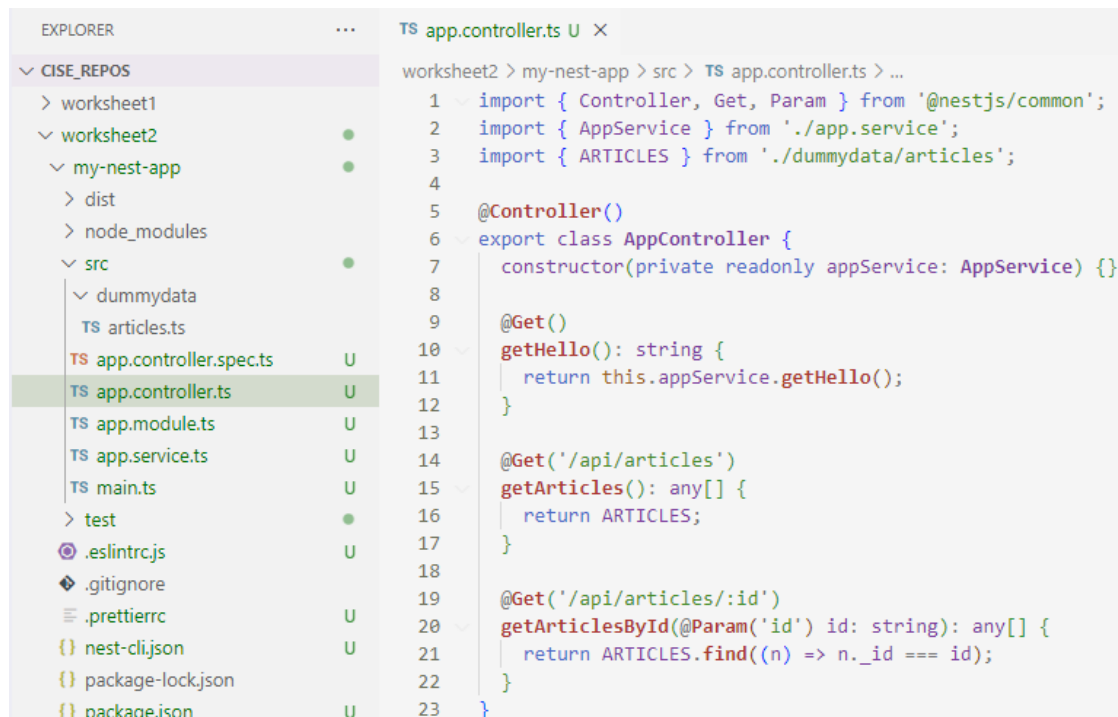
13. To show all the articles let's use the end point "/api/articles". Add another endpoint (see Fig 8).



Fig 8

Now **stop and restart** (*control + c in the terminal*) your server and go to <http://localhost:3000/api/articles> on your web browser. You should see all articles.

14. To show just a single article using its "id" we can use the find() array method to find the single article with the "id" specified in the path. See Fig 9 for what to add. The console.log will print out the "id" on the screen as you change it in the URL in your browser. You can delete this line later. [Read about the @Param decorator here](#)



```

EXPLORER
CISE_REPOS
  worksheet1
  worksheet2
  my-nest-app
    dist
    node_modules
    src
      dummydata
        TS articles.ts
        TS app.controller.spec.ts
        TS app.controller.ts
        TS app.module.ts
        TS app.service.ts
        TS main.ts
    test
    .eslinttrc.js
    .gitignore
    .prettierrc
    nest-cli.json
    package-lock.json
    package.json

TS app.controller.ts
1 import { Controller, Get, Param } from '@nestjs/common';
2 import { AppService } from './app.service';
3 import { ARTICLES } from './dummydata/articles';
4
5 @Controller()
6 export class AppController {
7   constructor(private readonly appService: AppService) {}
8
9   @Get()
10  getHello(): string {
11    return this.appService.getHello();
12  }
13
14  @Get('/api/articles')
15  getArticles(): any[] {
16    return ARTICLES;
17  }
18
19  @Get('/api/articles/:id')
20  getArticlesById(@Param('id') id: string): any[] {
21    return ARTICLES.find((n) => n._id === id);
22  }
23 }

```

Fig 9

Now restart the server and type in the URL <http://localhost:3000/api/articles/1> to see the first article. You can change which article is displayed in the browser by changing the id number in the URL.

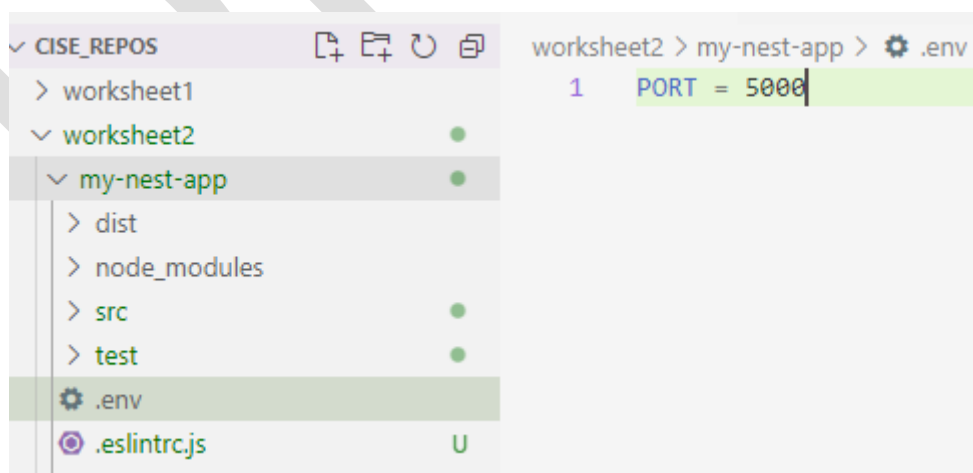
SIDE-NOTE: .env file

We are going to tidy up hard coding the port number (5000) in our code by using a configuration file “.env” to store the port number.

15. Create a file in the **Worksheet2/my-nest-app** folder called “.env”

16. Install the package: **>npm i dotenv**

17. In the .env file add the line **PORT = 5000**



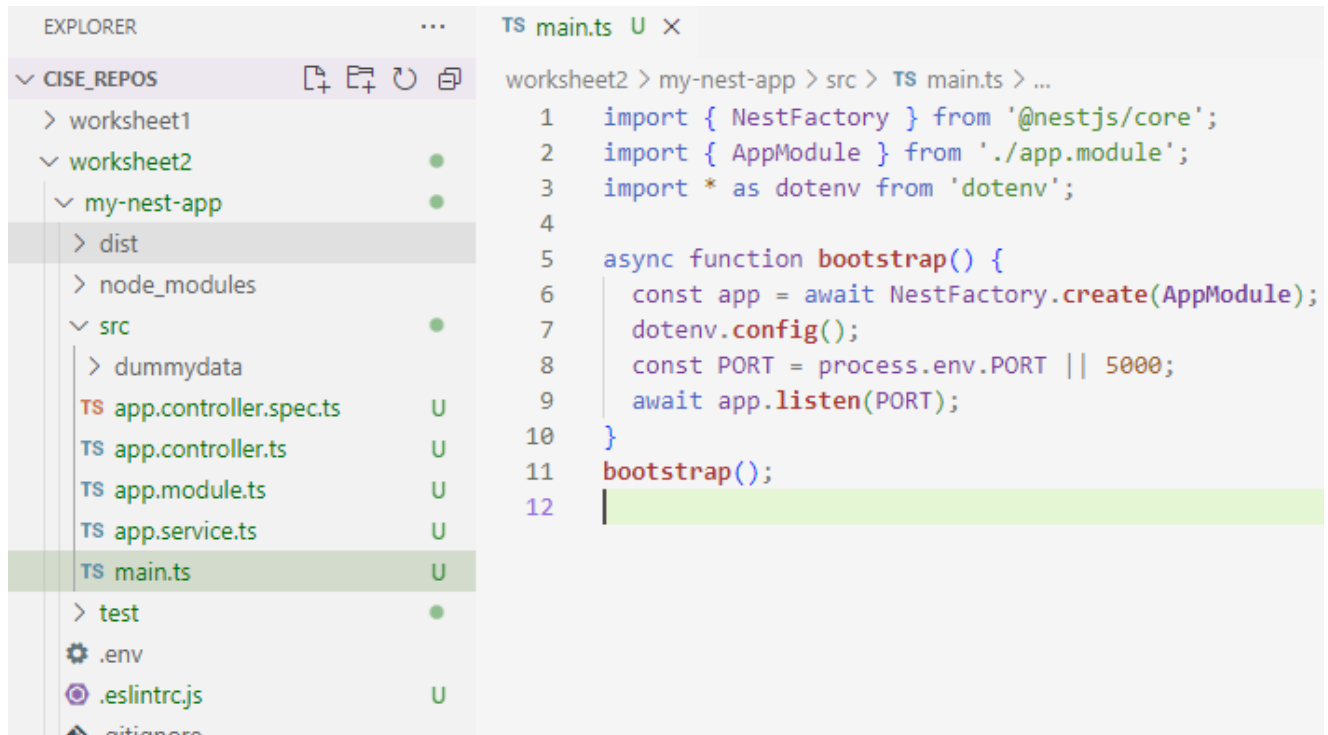
```

CISE_REPOS
  worksheet1
  worksheet2
  my-nest-app
    dist
    node_modules
    src
    test
    .env
    .eslinttrc.js
    .gitignore
    .prettierrc
    nest-cli.json
    package-lock.json
    package.json

.env
1 PORT = 5000

```

18. In **main.ts** add the lines shown in Fig 10.



The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar showing a project structure. The 'src' directory is expanded, listing several TypeScript files. The file 'main.ts' is selected and highlighted in green. On the right is the editor window showing the content of 'main.ts'. The code is a TypeScript file that imports NestFactory and AppModule, configures dotenv, and defines an async bootstrap function that creates the application and listens on a port. The port is defined as process.env.PORT or 5000.

```
EXPLORER
CISE_REPOS
  worksheet1
  worksheet2
    my-nest-app
      dist
      node_modules
      src
        dummydata
        TS app.controller.spec.ts
        TS app.controller.ts
        TS app.module.ts
        TS app.service.ts
        TS main.ts
      test
      .env
      .eslinttrc.js
      .gitignore

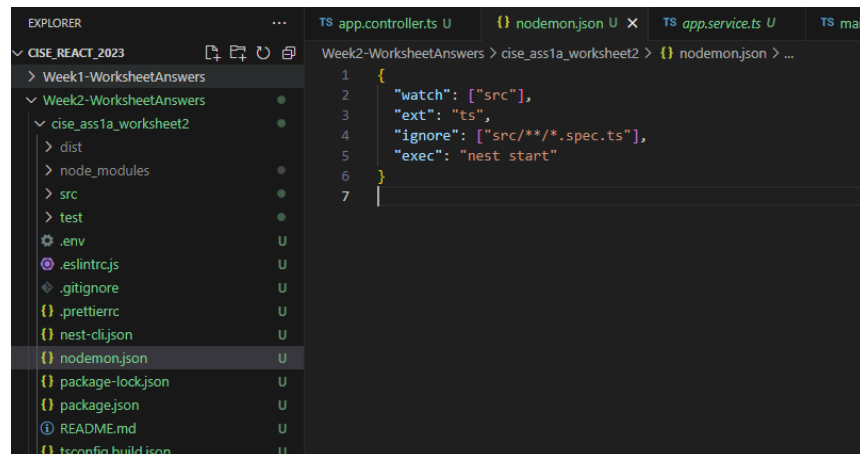
TS main.ts
worksheet2 > my-nest-app > src > TS main.ts > ...
1  import { NestFactory } from '@nestjs/core';
2  import { AppModule } from './app.module';
3  import * as dotenv from 'dotenv';
4
5  async function bootstrap() {
6    const app = await NestFactory.create(AppModule);
7    dotenv.config();
8    const PORT = process.env.PORT || 5000;
9    await app.listen(PORT);
10 }
11 bootstrap();
12
```

Fig 10

You can see the port will either use the port number. In the .env file or if that is not available it will use port 5000. ("||" means "OR").

SIDE-NOTE: install nodemon package

19. To save having to stop and restart the server continually, we can use a package called “nodemon” (node monitor). `> npm i nodemon` This will automatically restart our web server whenever we change a file. This will add nodemon to our package.json dependencies. Create a nodemon.json file and add the following:



```

1 {
2   "watch": ["src"],
3   "ext": "ts",
4   "ignore": ["src/**/*.spec.ts"],
5   "exec": "nest start"
6 }
7

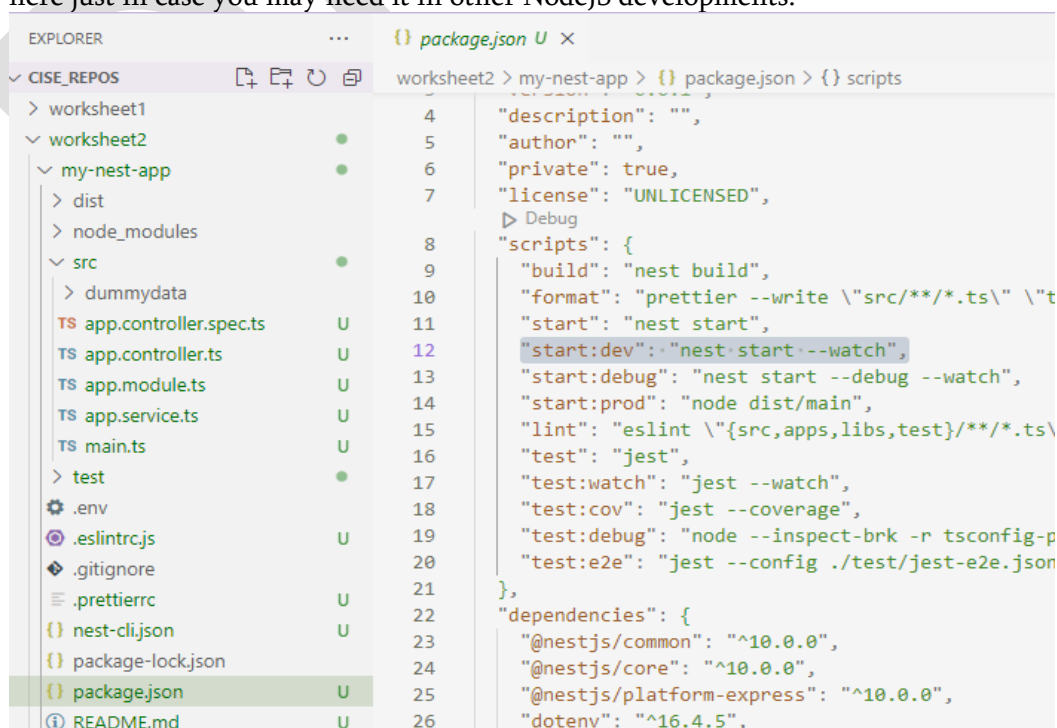
```

You also need to change the start script in package.json”.

```
"start": "nodemon",
```

Now when you “npm start” your server will automatically restart if you change any file. So you can see what happens in your browser immediately after change. Try it! (e.g. change the root “/” API message)

NOTE: NestJS has a built-in script called “start:dev”, which works basically the same way as nodemon. If you run “npm run start:dev” then it will start in the developing mode, all changes to the source files will be watched and the app will be restarted automatically. We still put nodemon here just in case you may need it in other NodeJS developments.



```

4   "description": "",
5   "author": "",
6   "private": true,
7   "license": "UNLICENSED",
8   "scripts": {
9     "build": "nest build",
10    "format": "prettier --write 'src/**/*.ts' 'test/**/*.ts'",
11    "start": "nest start",
12    "start:dev": "nest start --watch",
13    "start:debug": "nest start --debug --watch",
14    "start:prod": "node dist/main",
15    "lint": "eslint '{src,apps,libs,test}/**/*.ts'",
16    "test": "jest",
17    "test:watch": "jest --watch",
18    "test:cov": "jest --coverage",
19    "test:debug": "node --inspect-brk -r tsconfig-paths/register ts-node src/index.ts",
20    "test:e2e": "jest --config ./test/jest-e2e.json",
21  },
22  "dependencies": {
23    "@nestjs/common": "^10.0.0",
24    "@nestjs/core": "^10.0.0",
25    "@nestjs/platform-express": "^10.0.0",
26    "dotenv": "^16.4.5",

```

SIDE-NOTE: Postman (extension)

20. Postman should be installed on your machine also, for later use. It is software that allows you to test any API using the POST, GET, PUT, DELETE, http commands. Try installing it and use it to test your localhost:5000 API etc.

Create a Front End using React.js

21. In terminal in VS Code move to the root directory for this worksheet (use `>cd ..` to move up one level)
22. Similar to the first worksheet, create a react app called “my-react-app”

`>npx create-react-app my-react-app --template typescript`

Note: --template typescript flag will set this react application to use typescript instead of javascript

Now remove this inside the package.json from the my-react-app directory – we already have this configured we do not need it again:

```

1  {
2    "name": "my-react-app",
3    "version": "0.1.0",
4    "private": true,
5    "dependencies": {
6      "@testing-library/jest-dom": "^5.17.",
7      "@testing-library/react": "^13.4.0",
8      "@testing-library/user-event": "^13.
9      "@types/jest": "^27.5.2",
10     "@types/node": "^16.18.104",
11     "@types/react": "^18.3.3",
12     "@types/react-dom": "^18.3.0",
13     "react": "^18.3.1",
14     "react-dom": "^18.3.1",
15     "react-scripts": "5.0.1",
16     "typescript": "^4.9.5",
17     "web-vitals": "^2.1.4"
18   },
19   "scripts": {
20     "start": "react-scripts start",
21     "build": "react-scripts build",
22     "test": "react-scripts test",
23     "eject": "react-scripts eject"
24   },
25   "eslintConfig": {
26     "extends": [
27       "react-app",
28       "react-app/jest"
29     ]
30   },
31   "browserslist": {
32     "production": [
33       ">0.2%",
34       "not dead",
35       "not op_mini all"
36     ],
37     "development": [

```

“create-react app” is an officially supported way to create **Single-Page Applications** (read about this here <https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps>).

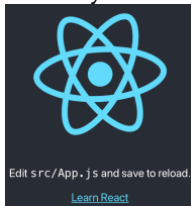
“create-react-app” will install many packages and set up some specific scripts. Some of the files are specified in the dependencies section of the specified in the *package.json* file (in the **frontend folder** – **there is another package.json in the root folder that relates to the backend**). Have a look at the *package.json* file. You can see all the other packages installed (these were installed automatically by create-react-app (**>npm install** will install all the dependencies in the package.json file manually – sometimes you need to do this after cloning another repository)

The create-react-app has installed jest (a testing package), as well as a linter (eslint), and babel and lots of other files.

Note: If you want to know more about the format of this package.json file, then look at <https://nodejs.dev/learn/the-package-json-guide>

>npm start will run the “script labelled “start” in the *package.json* file (which runs a script in react-script library called “start” installed as a dependency).

When you do this it should open a browser to **localhost:3000** and you should see this:



This should look familiar!

SIDE-NOTE about React.js

The App.js file is being rendered on the Document Object Mode (DOM) representing the web page, with the help of the index.js file in the src directory. All the javascript used to manipulate the DOM is in the App.js file, written in a language similar to html, called jsx. React is a package used to create User Interfaces (UIs) using jsx which is changed to javascript at runtime, but we don't need to worry about that. React has some benefits over vanilla javascript including:

- (a) it uses components that can be re-used,
- (b) it is easy to write in jsx,
- (c) it manipulates a virtual DOM in memory then updates only the changes to the real DOM. This means it is very fast since it rarely needs to refresh a page (from the server) in the browser (a single-page can be made to look like many pages).

Clean up the created react files to start your own app

(Remember to “>cd” so you are working in the frontend directory in terminal).

23. In the App.tsx file delete everything between <header> tags, including the tags, so there is only the <div> tags. This is NOT html, but is jsx. Note that the “class” in html is now “className” in jsx to point to a css file to apply.

```
import './App.css';

const App = () =>
  <div className="App">
    <h1>Welcome to CISE – the home of learning and fun</h1>
  </div>

export default App;
```

Create a very simple interactive app

24. Cut and paste the following code into App.tsx, replacing all the code that was there. Can you guess what this will do?

```
import React, { Component } from "react";

interface State {
  count: number;
}

class App extends Component<{}, State> {
  constructor(props: {}) {
    super(props);
    this.state = {
      count: 0,
    };
  }

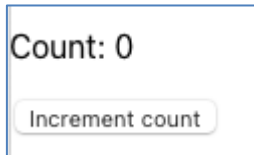
  makeIncrementer = (amount: number) => () =>
    this.setState((prevState: State) => ({
      count: prevState.count + amount,
    }));

  increment = this.makeIncrementer(1);

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button className="increment" onClick={this.increment}>
          Increment count
        </button>
      </div>
    );
  }
}
```

```
}
export default App;
```

25. Stage and commit this file in your local repo. (Remember: *git add* and *git commit -m "first commit"*)
26. You should see something like this in your browser



Run a test locally and manually

Normally, we'd need to install and configure Jest before writing any tests, but we do not need to do this since "create-react-app" has Jest already installed. In fact "create-react-app" creates a single test in the *src/App.test.tsx* file. It tests that the App.tsx component can render without crashing. Let's run that test manually and locally:

From the frontend folder....

27. `>npm test` will run jest behind the scenes and uses the "test" script in the *package.json* file. It will run the test watcher in an interactive mode. By default, it will run tests related to files changed since the last commit.
28. Change the *App.test.tsx* file to the following:

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

it('renders without crashing', () => {
  const div = document.createElement('div');
  ReactDOM.render(<App />, div);
  ReactDOM.unmountComponentAtNode(div);
});
```

29. Try running the test with `>npm test`
You should get 1 test passed out of 1

```
PASS src/App.test.js
  ✓ renders without crashing (15 ms)

Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 1.856 s
Ran all test suites.

Watch Usage: Press w to show more.[]
```

(you can press ctrl-c to exit this)

30. Add this dummy test just below the first one *in App.test.tsx* file (it doesn't actually check functionality!)

```
describe('Addition', () => {
  it('knows that 2 and 2 make 4', () => {
    expect(2 + 2).toBe(4);
  });
});
```

Can you understand what this test is testing? Notice the syntax! The thing to test is wrapped in a call to the “expect()” function, before calling “matcher” function on it “toBe()” in this case. It checks that the value provided equals the value that the code within the expect() function produces.

31. After saving and staging and committing the changes **run both tests** with `>npm test` and you should see both tests passing:

```
PASS src/App.test.js
  ✓ renders without crashing (15 ms)
  Addition
    ✓ knows that 2 and 2 make 4 (1 ms)

Test Suites: 1 passed, 1 total
Tests: 2 passed, 2 total
Snapshots: 0 total
Time: 1.56 s, estimated 2 s
Ran all test suites.
Watch Usage: Press w to show more.
```

(you can press ctrl-c to exit this)

32. Stage and commit all changes and push the local repo to GitHub
33. Check that the second test fails when it should (to convince yourself it won't just always pass!) by changing the “`.toBe(4)`” to “`.toBe(5)`”. (ie it will check $2 + 2 = 5$, which should fail)
Save changes and run both tests again locally.
You should see that the first test passes as before, while the second one fails, with the reason for the failure displayed.

Note: You can read more about the testing in React with ReactTestUtils at

<https://reactjs.org/docs/test-utils.html>

<https://reactjs.org/docs/testing.html>

“ReactTestUtils makes it easy to test React components in the testing framework of your choice. At Facebook we use [Jest](#) for painless JavaScript/Typescript testing. Learn how to get started with Jest through the Jest website’s [React Tutorial](#).

We recommend using [React Testing Library](#) which is designed to enable and encourage writing tests that use your components as the end users do.

A good tutorial is at freecodecamp <https://www.freecodecamp.org/news/testing-react-hooks/>

We will return to testing in a later tutorial.

SIDE-NOTE Build the App locally and manually

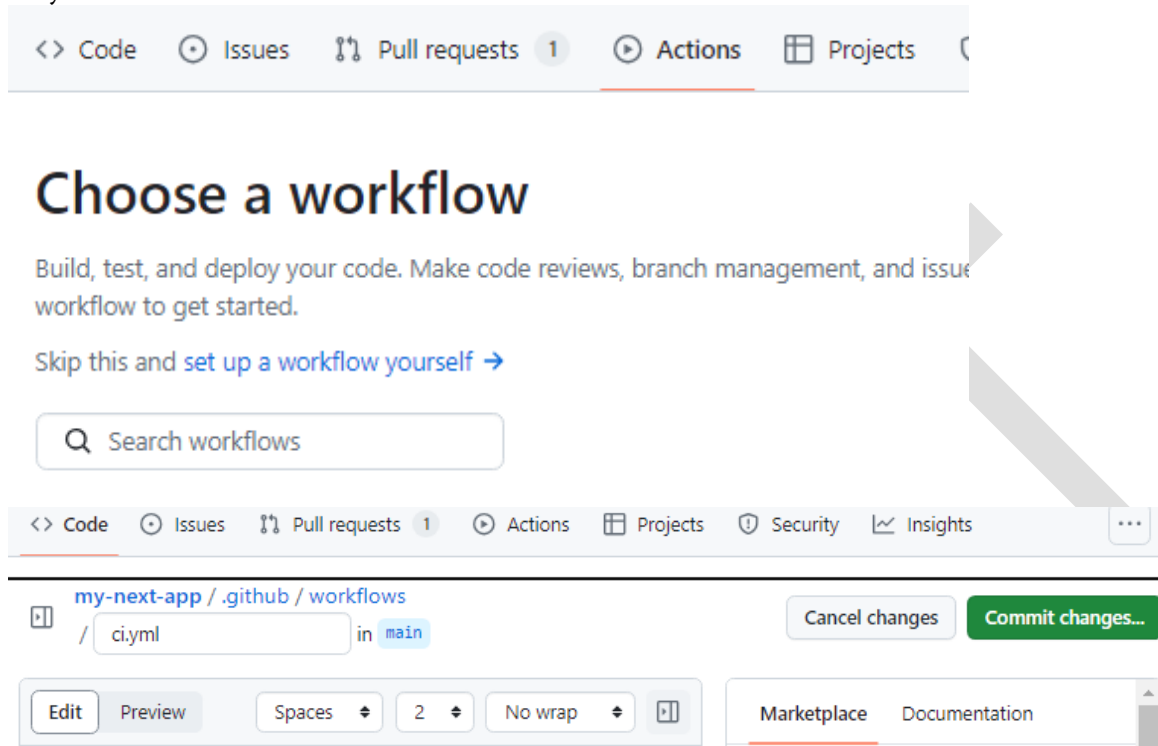
Builds the app for production to the *build* folder (check that it is created). It correctly bundles React in **production mode** and optimizes the build for the best performance.

DRAFT

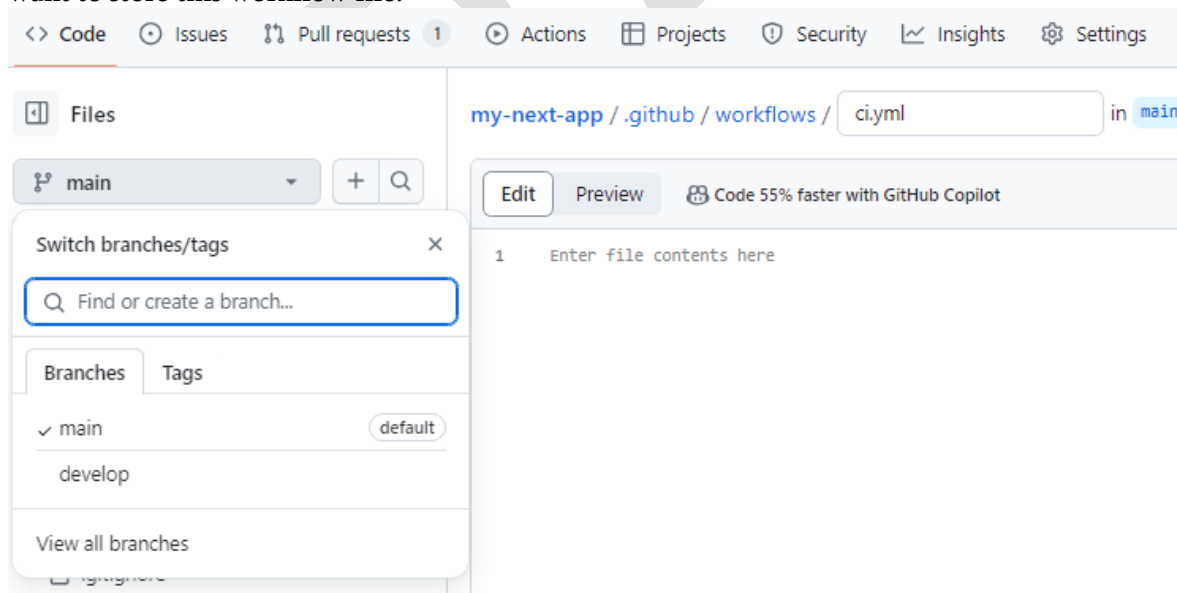
Automating the running tests using GitHub Actions

We need to create a workflow (a yaml file in the folder `.github\workflows`) that will define what triggers to wait for (like “pull request”) and what actions to take (like “run all tests”)

You can do this most easily from GitHub. Select “Actions” and create your own workflow, name it “ci.yml”.



Click the little icon on the left side, you can expand the file tree. In here you can select which branch you want to store this workflow file.



Type the content in to the ci.yml:

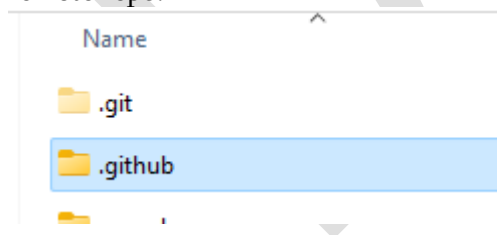
```
1  name: CI_action
2
3  on:
4    push:
5      branches: [main]
6    pull_request:
7      branches: [main]
8
9  jobs:
10   build:
11     runs-on: ubuntu-latest
12
13     strategy:
14       matrix:
15         node-version: [12.x]
16
17     steps:
18       - name: Checkout repository
19         uses: actions/checkout@v2
20
21       - name: Set up Node.js ${{ matrix.node-version }}
22         uses: actions/setup-node@v1
23         with:
24           node-version: ${{ matrix.node-version }}
25
26       - name: Install dependencies
27         run: npm install
28
29       - name: Run the tests
30         run: npm test
31
32       - name: Build
33         run: npm run build
34
```

Then a new commit that adds this file into your repository will be created by GitHub. You can pull this change into your local repository.

NOTE: the spacing is very particular – get it wrong and things go wrong.

You should pull this to your local repo, and you should see the .github/workflows folder with ci.yml file. What will trigger this action? What actions will it take?

You can also start doing this from your local repository. Create a .github folder in your local repository's root folder (i.e. the folder that contains your .git folder), then create a "workflows" folder in it, create a ci.yml in the "workflows" folder, type the content from above. Then git add, commit and push it to the remote repo.

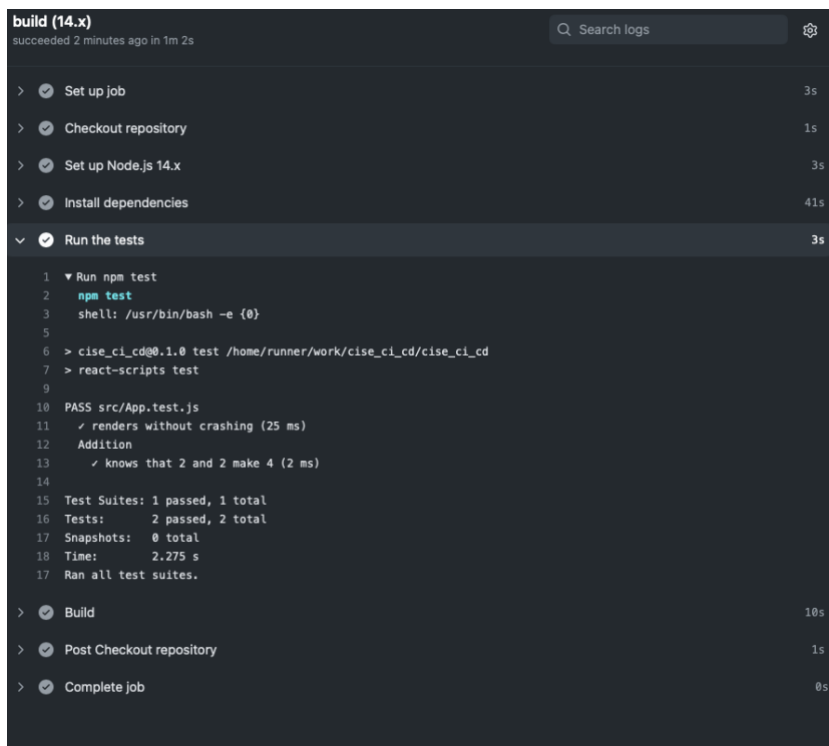


NOTE: The .github folder must be located in the repository's root folder to take effect, otherwise it won't be triggered. If your code and packages.json is not in the root folder (e.g. /root/second_layer/packages.json), then you can use "[working-directory](#)" to specify its path for each step or all steps in a job. The "working-directory" should be the relative path to your root folder (e.g. "./second_layer").

Trigger the action by changing this yaml file from node-version 12.x to 14.x –commit and push to GitHub. This will push to main and trigger the action.

Open the Actions tab in GitHub and click on the last commit (you should see your commit message). Keep clicking until you see the job (it will take around a minute).

You will see displayed each step in the ci.yml file as it runs, including the tests.



The screenshot shows a GitHub Actions workflow run for a job named 'build (14.x)'. The job is marked as 'succeeded' and took '2 minutes ago in 1m 2s'. The log displays the following steps and their durations:

- Set up job: 3s
- Checkout repository: 1s
- Set up Node.js 14.x: 3s
- Install dependencies: 41s
- Run the tests: 3s
- Build: 10s
- Post Checkout repository: 1s
- Complete job: 0s

The 'Run the tests' step is expanded, showing the following command and output:

```
1 Run npm test
2 npm test
3 shell: /usr/bin/bash -e {0}
5
6 > cise_ci_cd@0.1.0 test /home/runner/work/cise_ci_cd/cise_ci_cd
7 > react-scripts test
9
10 PASS src/App.test.js
11 ✓ renders without crashing (25 ms)
12 Addition
13 ✓ knows that 2 and 2 make 4 (2 ms)
14
15 Test Suites: 1 passed, 1 total
16 Tests: 2 passed, 2 total
17 Snapshots: 0 total
18 Time: 2.275 s
19 Ran all test suites.
```

This is the start of CI/CD!

Change this so the ci.yml action is triggered on a merge to main and test it out.

Name:**Date:****Worksheet Evidence:**

This worksheet requires some answers to questions and **at least three selectively captured screenshots with an in depth reflection** as evidence. The aim is to be able to learn from the exercise, and evidence that.

For each of your three selected screenshots (or sequence of shots) in a brief paragraph or two reflect on:

- Why you have selected it?
- What have you learnt in this part of the worksheet?
- What was new or surprising?
- What useful external resource(s) did you consult and why? Provide a link(s) to the resource.

Evidence	Check
<p>1. Discuss how you deployed the backend APIs at “localhost:5000/api/articles/3” on your local browser. A screenshot(s) to support your answer may be suitable.</p> <p>The backend API is created using a NestJS application that runs locally and services HTTP requests to port 5000.</p> <pre> async function bootstrap() { const app = await NestFactory.create(AppModule); dotenv.config(); const PORT = process.env.PORT 5000; await app.listen(PORT); } bootstrap(); </pre> <p>When an HTTP request is sent via a browser or another request sending application such as Postman to localhost:5000, the NestJS application responds by running the code located in the AppController class. Each request method is set by using @Method('endpoint') such as @Get('/api/articles'). Below this, a function can be declared that includes the code that should be run when this endpoint is hit.</p> <pre> @Get('/api/articles') getArticles(): any[] { return ARTICLES; } </pre> <p>The end result in the browser is the result of the function being displayed in the browser viewport.</p>	

	
<p>2. Show your knowledge of the jest testing framework, and how it supports testing, through discussing an example of one test passing and one test failing when you run the tests locally for the React frontend. A screenshot(s) to support your answer may be suitable.</p> <p>In the tests discussed in the worksheet, there is a test that checks that the React front end loads without crashing and another test that checks if $2 + 2 = 4$. If the expected result in the toBe() function is set to 5 instead of 4, a failed test is thrown, shown below.</p> 	
<p>3. What is an API endpoint?</p> <p>An API endpoint is a URI where HTTP requests can be sent. Usually, the server running the API has some logic to handle what happens when a request is sent, and what to do depending on what kind of HTTP request is sent. Common HTTP requests include GET, PUT, POST, DELETE, and are used for common CRUD operations.</p> <p>http://localhost:5000/api/articles/3</p>	
<p>4. What CLI command would install all the packages that are dependencies listed in the package.json file?</p> <p>The command <code>`npm install`</code> or <code>`npm i`</code> for short, is used to install all packages that are dependencies listed in package.json.</p>	

```

➔ cise_ass1a_worksheet2 git:(master) npm i
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.
npm warn deprecated @humanwhocodes/config-array@0.11.14: Use @eslint/config-array instead
npm warn deprecated rimraf@3.0.2: Rimraf versions prior to v4 are no longer supported
npm warn deprecated @humanwhocodes/object-schema@2.0.3: Use @eslint/object-schema instead
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported

added 707 packages, and audited 708 packages in 2s

110 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

```

5. What was the purpose of the “.env” file?

A .env file is a file where environment variables and secrets can be stored. It is a hidden file and is intended to be ignored by a gitignore file.

The reason .env files are so useful is production code does not have to have sensitive data such as database credentials, API keys/tokens, and other important data hard coded into the program.

It means if any of these credentials are updated or need to be changed, they can be easily accessed and edited in the .env file. It also means your code can be public or open source and other users can hook up their own databases in their own .env file without having access to the creators database credentials or secrets.

6. What is the purpose of the .gitignore file?

A .gitignore file is a file used for ignoring files from being tracked in the version control system Git. Git by default with no gitignore file tracks all files and changes to files in the parent directory of the .git directory.

Some files (such as .env files) include sensitive data, or are intended to be downloaded from package managers (such as npm), and are not meant to be packaged with the code in the repository.

This means that total repo sizes can be much smaller because repo dependencies can be downloaded from npm when a user clones the repository by running the npm i command.

When a file or directory is added to a gitignore file, it is untracked, meaning it will not be packaged with the repo when committing locally, and is not included when pushed or pulled from origin or another location.

Example gitignore from a NestJS application:

```

# compiled output
/dist
/node_modules
/build

# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
lerna-debug.log*

# OS
.DS_Store

# Tests
/coverage
/.nyc_output

# IDEs and editors
/.idea
.project
.classpath
/.c9/
*.launch
.settings/
*.sublime-workspace

# IDE - VSCode
.vscode/*
!.vscode/settings.json
!.vscode/tasks.json
!.vscode/launch.json
!.vscode/extensions.json

# dotenv environment variable files
.env
.env.development.local
.env.test.local
.env.production.local
.env.local

# temp directory
/tmp
*.tmp

# Runtime data
/pids
*.pid
*.sock
*.pid.lock

# Diagnostic reports (https://nodejs.org/api/report.html)
report.[0-9]*.[0-9]*.[0-9]*.json

```

7. What is a single-page application?

A single page application is an application like a React project. The idea behind single page applications is that they make use of a technology such as virtual DOM to update elements in the DOM without having to refresh the page.

Elements on the page are componentised and able to be refreshed independently of other elements. The term “single-page application” comes from the fact that the page appears to be non-changing from the routing, to lack of full-page refresh.

So, a full application can live on a single page, only changing what is required to display different content depending on user interaction in the virtual DOM.

8. Show an example of a CI/CD workflow with automated testing and a workflow enabled by GitHub actions. A screenshot(s) to support your answer may be suitable.

```

1  name: Worksheet2_CI_ACTION
2
3  on:
4    push:
5      branches:
6        - master
7      paths:
8        - "worksheet2/**"
9
10   pull_request:
11     branches:
12       - master
13     paths:
14       - "worksheet2/**"
15
16   jobs:
17     build:
18       runs-on: ubuntu-latest
19       defaults:
20         run:
21           working-directory: ./worksheet2/cise_essla_worksheet2-frontend
22
23       strategy:
24         matrix:
25           node-version: [16.x]
26
27       steps:
28         - name: Checkout repository
29           uses: actions/checkout@v2
30
31         - name: Set up Node.js ${{ matrix.node-version }}
32           uses: actions/setup-node@v1
33           with:
34             node-version: ${{ matrix.node-version }}
35
36         - name: Install dependencies
37           run: npm install
38
39         - name: Run the test
40           run: npm test
41
42         - name: Build
43           run: npm run build

```

build (16.x)
succeeded now in 34s

Search logs

✓ Install dependencies 16s

```

29 npm WARN deprecated @babel/plugin-proposal-numeric-separator@7.18.6: This proposal has
30 been merged to the ECMAScript standard and thus this plugin is no longer maintained.
31 Please use @babel/plugin-transform-numeric-separator instead.
32 npm WARN deprecated @babel/plugin-proposal-nullish-coalescing-operator@7.18.6: This
33 proposal has been merged to the ECMAScript standard and thus this plugin is no longer
34 maintained. Please use @babel/plugin-transform-nullish-coalescing-operator instead.
35 npm WARN deprecated @babel/plugin-proposal-class-properties@7.18.6: This proposal has
36 been merged to the ECMAScript standard and thus this plugin is no longer maintained.
37 Please use @babel/plugin-transform-class-properties instead.
38 added 1522 packages, and audited 1523 packages in 16s
39 261 packages are looking for funding
40 run 'npm fund' for details
41 8 vulnerabilities (2 moderate, 6 high)
42 To address all issues (including breaking changes), run:
43   npm audit fix --force
44 Run 'npm audit' for details.

```

✓ Run the test 3s

```

1  ▶ Run npm test
2  > cise_essla_worksheet2-frontend@0.1.0 test
3  > react-scripts test
4  PASS src/App.test.tsx
5  ✓ renders without crashing (9 ms)
6  Addition
7  ✓ knows that 2 and 2 make 4 (1 ms)
8  Test Suites: 1 passed, 1 total
9  Tests: 2 passed, 2 total
10 Snapshots: 0 total
11 Time: 1.147 s
12 Ran all test suites.

```

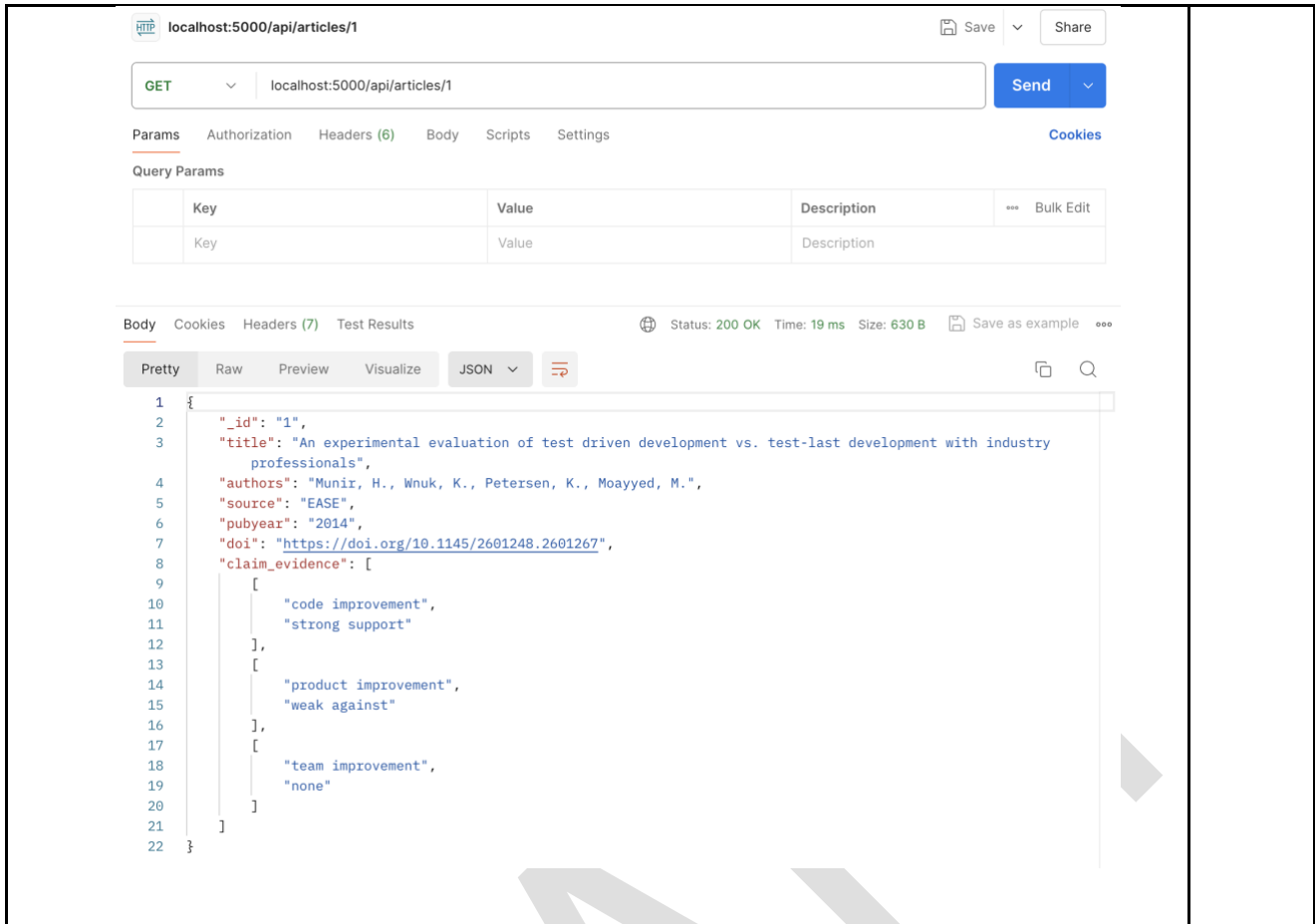
✓ Build 7s

```

1  ▶ Run npm run build
2
3  > cise_essla_worksheet2-frontend@0.1.0 build
4  > react-scripts build
5  Creating an optimized production build...
6  Compiled successfully.
7  File sizes after gzip:
8    44.52 kB build/static/js/main.79a15354.js
9    1.8 kB build/static/js/463.72f080b1.chunk.js
10   264 B build/static/css/main.e6c13ad2.css
11 The project was built assuming it is hosted at /.
12 You can control this with the homepage field in your package.json.
13 The build folder is ready to be deployed.
14 You may serve it with a static server:
15   npm install -g serve
16   serve -s build
17 Find out more about deployment here:
18   https://cra.link/deployment

```

9. You may choose to select another aspect of the worksheet to evidence (e.g. the use of Postman), with a screenshot(s) to support your answer.



The screenshot displays a REST client interface with the following components:

- URL Bar:** Shows the URL `localhost:5000/api/articles/1` with a `GET` method selected.
- Send Button:** A blue button labeled "Send" with a dropdown arrow.
- Params Tab:** A tab labeled "Params" is active, showing a table for query parameters.
- Query Params Table:**

Key	Value	Description
Key	Value	Description
- Body Tab:** A tab labeled "Body" is active, showing the response in JSON format.
- Status Bar:** Displays "Status: 200 OK", "Time: 19 ms", and "Size: 630 B".
- JSON View:** The response is shown in a "Pretty" JSON view, displaying the following structure:

```
{  "id": "1",  "title": "An experimental evaluation of test driven development vs. test-last development with industry professionals",  "authors": "Munir, H., Wnuk, K., Petersen, K., Moayyed, M.",  "source": "EASE",  "pubyear": "2014",  "doi": "https://doi.org/10.1145/2601248.2601267",  "claim_evidence": [    [      "code improvement",      "strong support"    ],    [      "product improvement",      "weak against"    ],    [      "team improvement",      "none"    ]  ]}
```