

Bachelor of Computer and Information Sciences**Contemporary Issues in Software Engineering
Semester 2, 2023****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 taking screenshots of your work, as well as explanations.

This worksheet should be Checked off and uploaded to Canvas by end of Tutorial Week 2.

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 "cise_ass1a_worksheet2". In terminal change to being in this new folder (use the >cd command).
2. Initialise this folder for use with node.js, use 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. See fig 1.
3. Initialise this folder ("cise_ass1a_worksheet2") as a local Git repo.

```

Jims-iMac-Pro:cise_ass1a_worksheet2 jbuchan$
Jims-iMac-Pro:cise_ass1a_worksheet2 jbuchan$ 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.

Jims-iMac-Pro:cise_ass1a_worksheet2 jbuchan$ 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: (cise_ass1a_worksheet2)
version: (1.0.0)
description: mern app
entry point: (index.js) server.js
test command:
git repository:
keywords:
author: Jim
license: (ISC)
About to write to /Users/jbuchan/Dropbox/00 AUT Current Year/00 CISE S2 2021/CISE_Repos/cise_ass1a_worksheet2/package.json
{
  "name": "cise_ass1a_worksheet2",
  "version": "1.0.0",
  "description": "mern app",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Jim",
  "license": "ISC"
}

Is this OK? (yes)
Jims-iMac-Pro:cise_ass1a_worksheet2 jbuchan$ git init
Initialized empty Git repository in /Users/jbuchan/Dropbox/00 AUT Current Year/00 CISE S2 2021/CISE_Repos/cise_ass1a_worksheet2/.git/
Jims-iMac-Pro:cise_ass1a_worksheet2 jbuchan$

```

Fig 1

You should have a new package.json file in the "cise_ass1a_worksheet2" folder, similar to Fig 2.

```

package.json — CISE_Repos

EXPLORER
CISE_REPOS
  cise_ass1a_worksheet2
    {} package.json
  CISE_React/cise-react-learn

cise_ass1a_worksheet2 > {} package.json > ...
1  {
2    "name": "cise_ass1a_worksheet2",
3    "version": "1.0.0",
4    "description": "mern app",
5    "main": "server.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "Jim",
10   "license": "ISC"
11  }

```

Fig 2

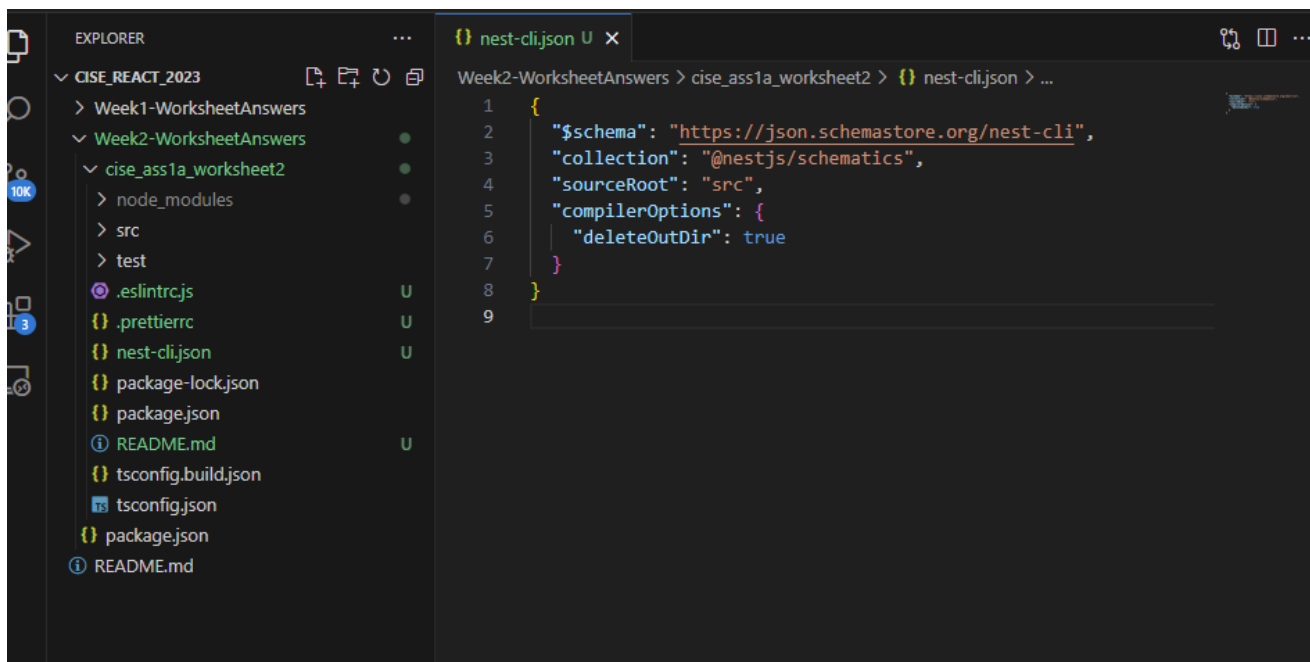
4. Ensure you have nest installed with: `>nest -version`

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

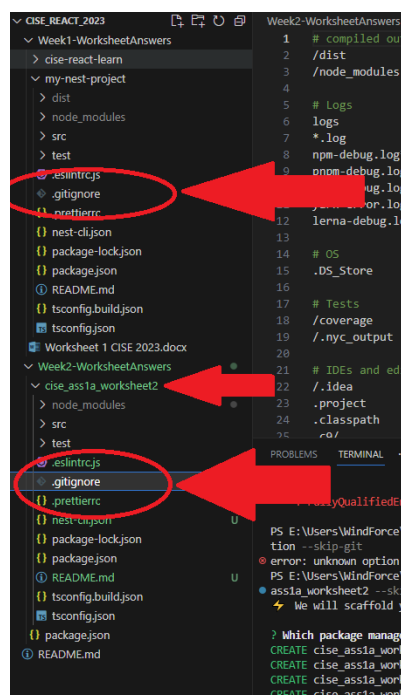
Now run the command `>nest new cise_ass1a_worksheet2 --skip-git`

(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 mainly add back in the .gitignore file. (read more about why we need [this file here](#)) Lets copy the one in week 1 “my-nest-project” and paste it into the cise_ass1a_worksheet2 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 .\cise_ass1a_worksheet2\
```

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 {NAME_OF_THE_SCRIPT_HERE}`

```

{
  "private": true,
  "license": "UNLICENSED",
  "scripts": {
    "build": "nest build",
    "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
    "start": "nest start",
    "start:dev": "nest start --watch",
    "start:debug": "nest start --debug --watch",
    "start:prod": "node dist/main",
    "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
    "test": "jest",
    "test:watch": "jest --watch",
    "test:cov": "jest --coverage",
    "test:debug": "node --inspect-brk -r tsconfig-paths/register",
    "test:e2e": "jest --config ./test/jest-e2e.json"
  },
  "dependencies": {
    "@nestjs/common": "^9.0.0",
    "@nestjs/core": "^9.0.0",
    "@nestjs/platform-express": "^9.0.0",
    "reflect-metadata": "^0.1.13",
    "rxjs": "^7.2.0"
  }
}

```

Now you can type `> npm start` to start the server. Pressing ctrl-c in terminal will stop the server.

```

sheet2> npm start

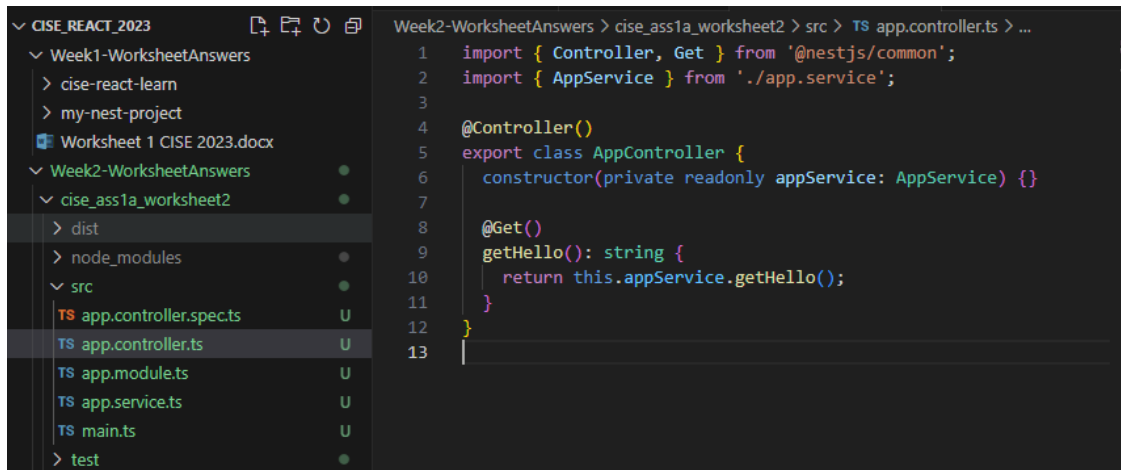
> cise_ass1a_worksheet2@0.0.1 start
> nest start

[Nest] 21016 - 05/06/2023, 6:07:46 pm    LOG [NestFactory] Starting Nest application
...
[Nest] 21016 - 05/06/2023, 6:07:46 pm    LOG [InstanceLoader] AppModule dependencies
initialized +24ms
[Nest] 21016 - 05/06/2023, 6:07:46 pm    LOG [RoutesResolver] AppController {/}: +25
ms
[Nest] 21016 - 05/06/2023, 6:07:46 pm    LOG [RouterExplorer] Mapped {/, GET} route
+5ms
[Nest] 21016 - 05/06/2023, 6:07:46 pm    LOG [NestApplication] Nest application succ
essfully started +3ms

```

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](#))



```

Week2-WorksheetAnswers > cise_ass1a_worksheet2 > src > 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).

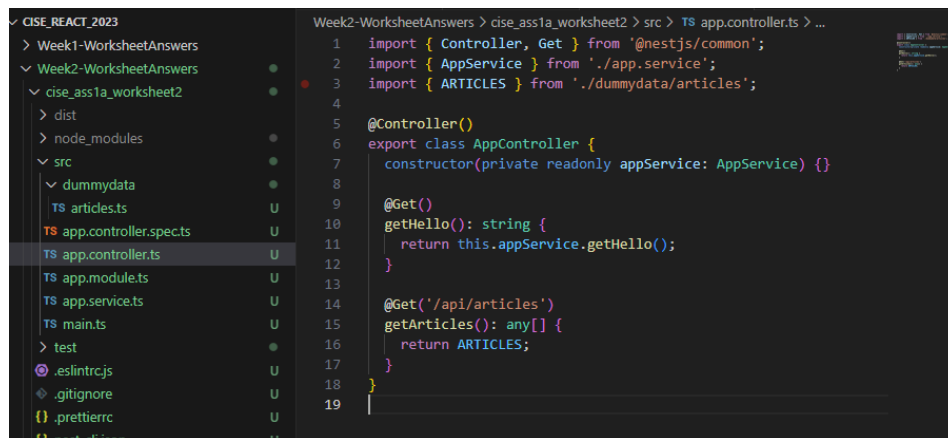


Fig 8

Now **stop and restart** (*control + c in the terminal*) your server and go to <https://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](#)

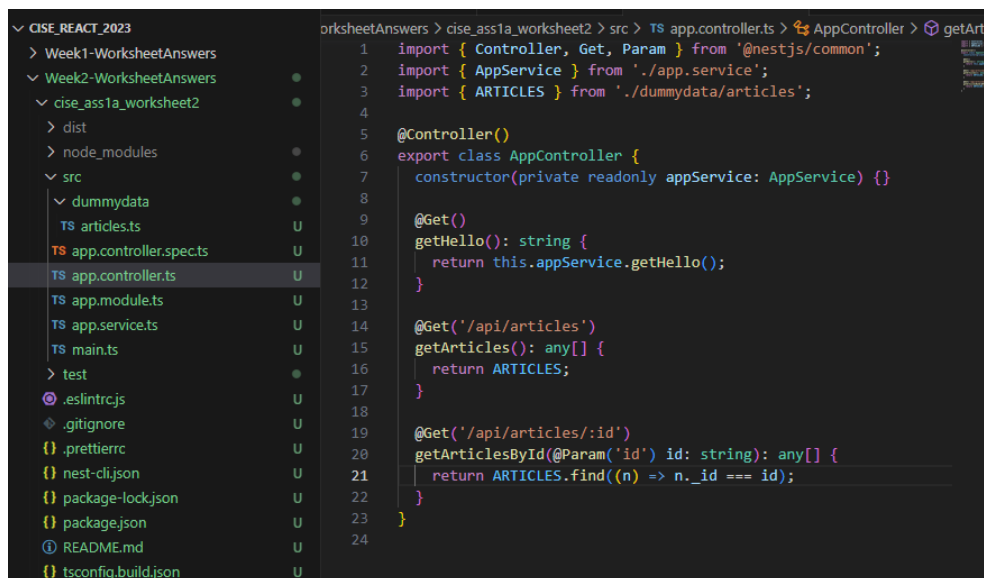


Fig 9

Now restart the server and type in the URL <https://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.

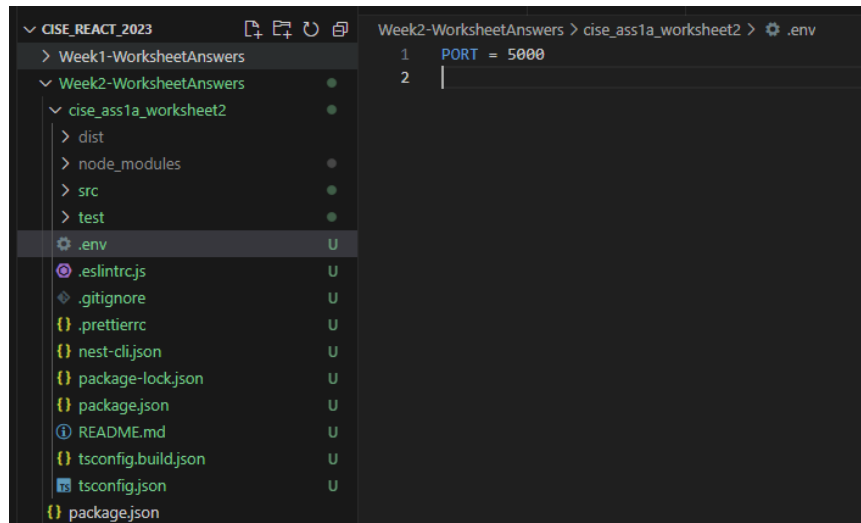
SIDE-NOTE: .env file

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 cise_ssla_worksheet2 folder called “.env”

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

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



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

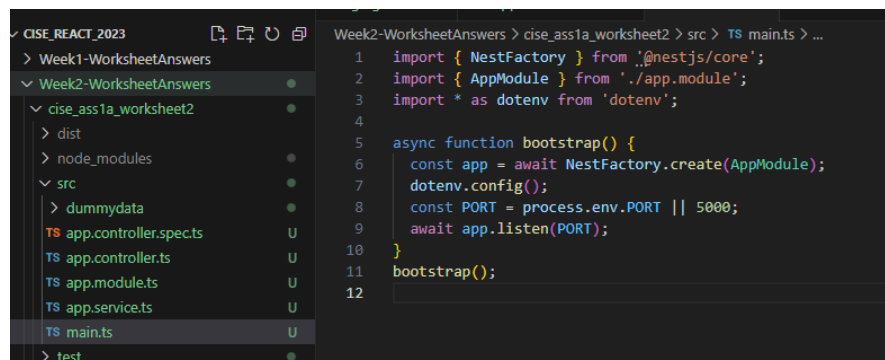
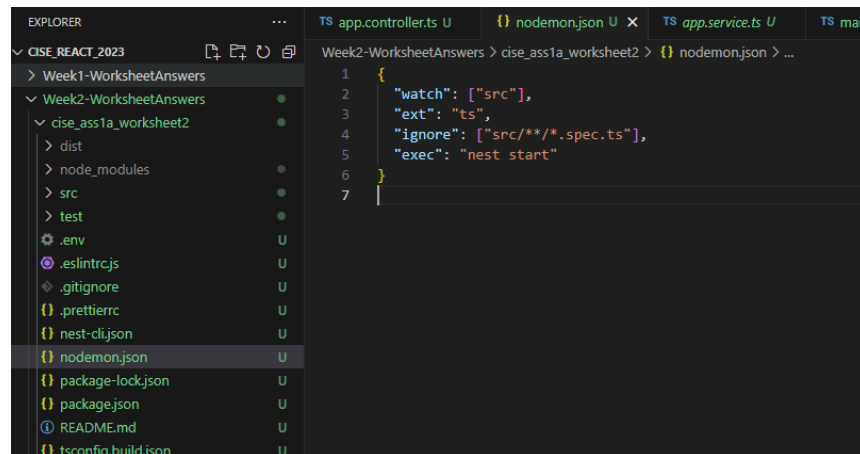


Fig 10

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:



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)

SIDE-NOTE: Postman (extension)

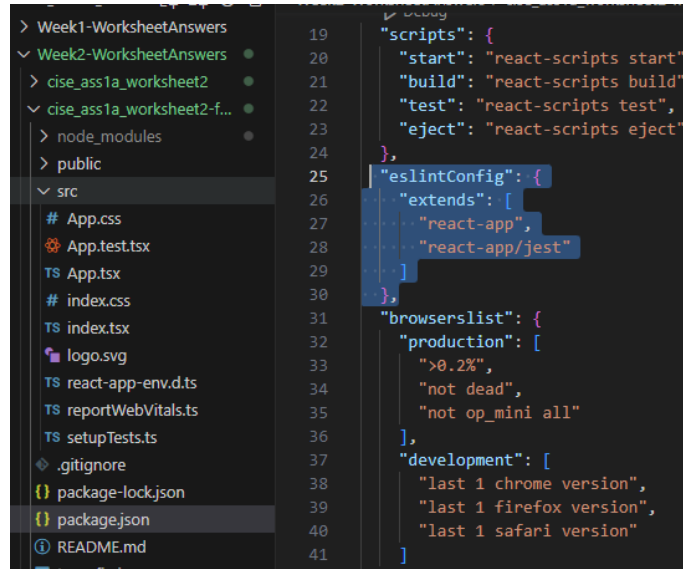
20. Postman should be installed on your machine also, for later use. It is software that allows you to test any API using the POST, GET, PUT, DELETE, http commands. Try installing it and use it to test your localhost:5000 API etc.

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 “cise_ass1a_worksheet2-frontend”
`>npx create-react-app cise_ass1a_worksheet2-frontend --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 **cise_ass1a_worksheet2-frontend** directory – we already have this configured we do not need it again:



```

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": [
38      "last 1 chrome version",
39      "last 1 firefox version",
40      "last 1 safari version"
41    ]
42  }

```

“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,
    };
  }

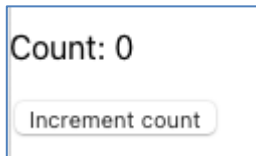
  makeIncrementer = (amount: number) => () =>
    this.setState((prevState: State) => ({
      count: prevState.count + amount,
    }));

  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.[]
```

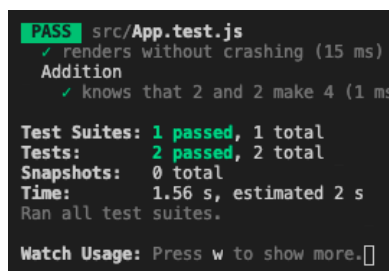
(you can press ctrl-c to exit this)

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.

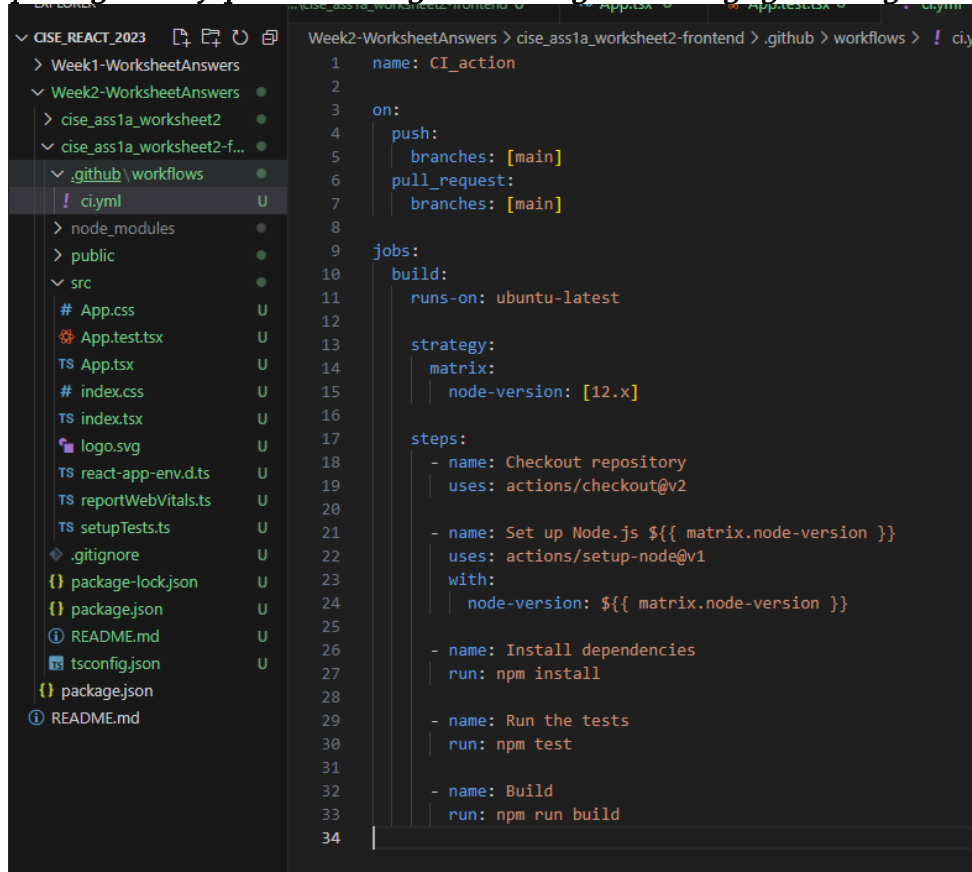
Automating the running tests using GitHub Actions

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” type the following into the ci.yml file:

Create a `.github` folder if one doesn’t exist already. Add another folder named `workflows` and add a file `ci.yml` with the following code:

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



```

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

```

The file explorer on the left shows the project structure:

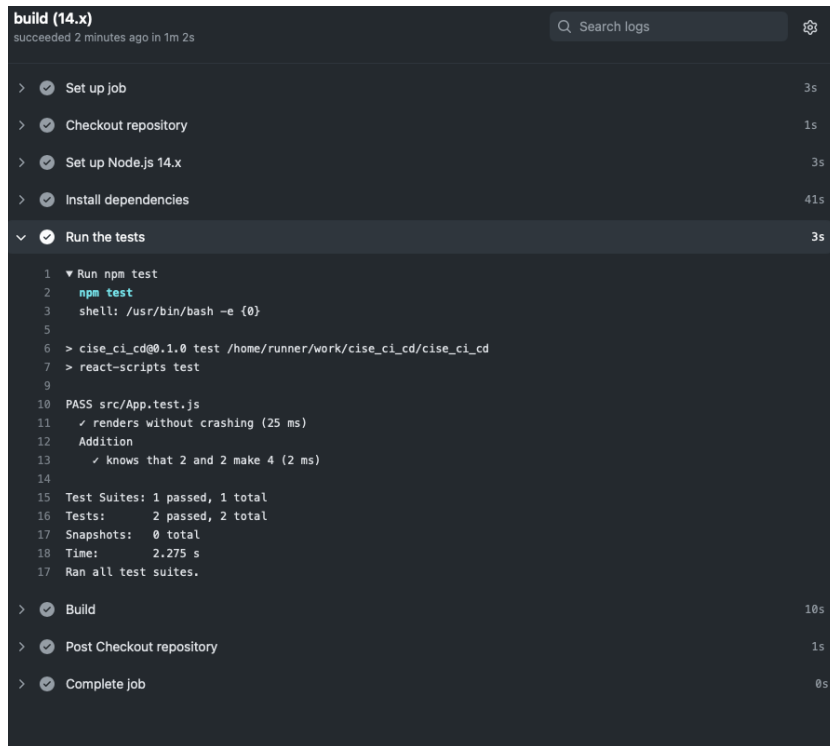
- CISE_REACT_2023
 - Week1-WorksheetAnswers
 - Week2-WorksheetAnswers
 - cise_ass1a_worksheet2
 - cise_ass1a_worksheet2-f...
 - .github**
 - workflows**
 - ci.yml**
 - node_modules
 - public
 - src
 - App.css
 - App.test.tsx
 - App.tsx
 - index.css
 - index.tsx
 - logo.svg
 - react-app-env.d.ts
 - reportWebVitals.ts
 - setupTests.ts
 - .gitignore
 - package-lock.json
 - package.json
 - README.md
 - tsconfig.json

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?

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 master 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 log for a job named 'build (14.x)'. The job is marked as 'succeeded 2 minutes ago in 1m 2s'. The log lists several steps: '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), and 'Complete job' (0s). The 'Run the tests' step is expanded, showing the execution of 'npm test'. The test output indicates that the tests passed, with a summary: 'Test Suites: 1 passed, 1 total', 'Tests: 2 passed, 2 total', 'Snapshots: 0 total', and 'Time: 2.275 s'.

```
build (14.x)
succeeded 2 minutes ago in 1m 2s

> Set up job 3s
> Checkout repository 1s
> Set up Node.js 14.x 3s
> Install dependencies 41s
v Run the tests 3s
  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
 17 Ran all test suites.

> Build 10s
> Post Checkout repository 1s
> Complete job 0s
```

This is the start of CI/CD!

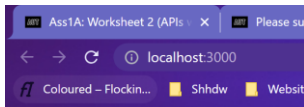
Extension

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

Worksheet Evidence Required to be Checked off and Uploaded to Canvas

Cut and paste screenshots of your web browser showing the evidence needed.

1. Paste a screenshot of the backend APIs at "localhost:5000/api/articles/3" on your local browser



Count: 7

Increment count

The hello world shows up on my browser as well as the incrementing count however the articles fail to load on the browser and this is an error I have not been able to fix.

2. Paste a screenshot of one test passing and one test failing when you run the tests locally for the React frontend.

```

App.test.tsx  App.tsx  package.json  nodemon.json
C:\Users\socce\Desktop\CISE_Repo\cise_ass1a_worksheet2\cise_ass1a_worksheet2
1  import React from 'react';
2  import ReactDOM from 'react-dom';
3  import App from './App';
4  it('renders without crashing', () => {
5    const div = document.createElement('div');
6    ReactDOM.render(<App />, div);
7    ReactDOM.unmountComponentAtNode(div);
8  })
9  describe('Addition', () => {
10   it('knows that 2 and 2 make 4', () => {
11     expect(2 + 2).toBe(5);
12   });
13 });

```

90 % No issues found

Developer PowerShell

+ Developer PowerShell

- Addition > knows that 2 and 2 make 4

```

expect(received).toBe(expected) // Object.is equality

Expected: 5
Received: 4

```

```

9 | describe('Addition', () => {
10 |   it('knows that 2 and 2 make 4', () => {
11 |     expect(2 + 2).toBe(5);
12 |   });
13 | });

```

at Object.<anonymous> (src/App.test.tsx:11:23)

A worker process has failed to exit gracefully and has been
rs can also cause this, ensure that .unref() was called on

Test Suites: 1 failed, 1 total
Tests: 1 failed, 1 passed, 2 total
Snapshots: 0 total
Time: 3.44 s
Ran all test suites.

Watch Usage: Press w to show more.

3. What CLI command would install all the packages that are dependencies listed in the package.json file?

npm install will install all the dependencies in the package.json file manually

4. What was the purpose of the ".env" file?

To store the port number/configuration settings in a file by hard coding the port number so it isn't in the source code

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

The .gitignore file is used to intentionally hide files and directories that shouldn't be included in the git repository.

6. What is a single-page application?

It's an application that operates on a single web page, where all the html, CSS and JavaScript are dynamically written on to the web page so it can rewrite the web page with new data from the web server instead of repeatedly loading new web pages.