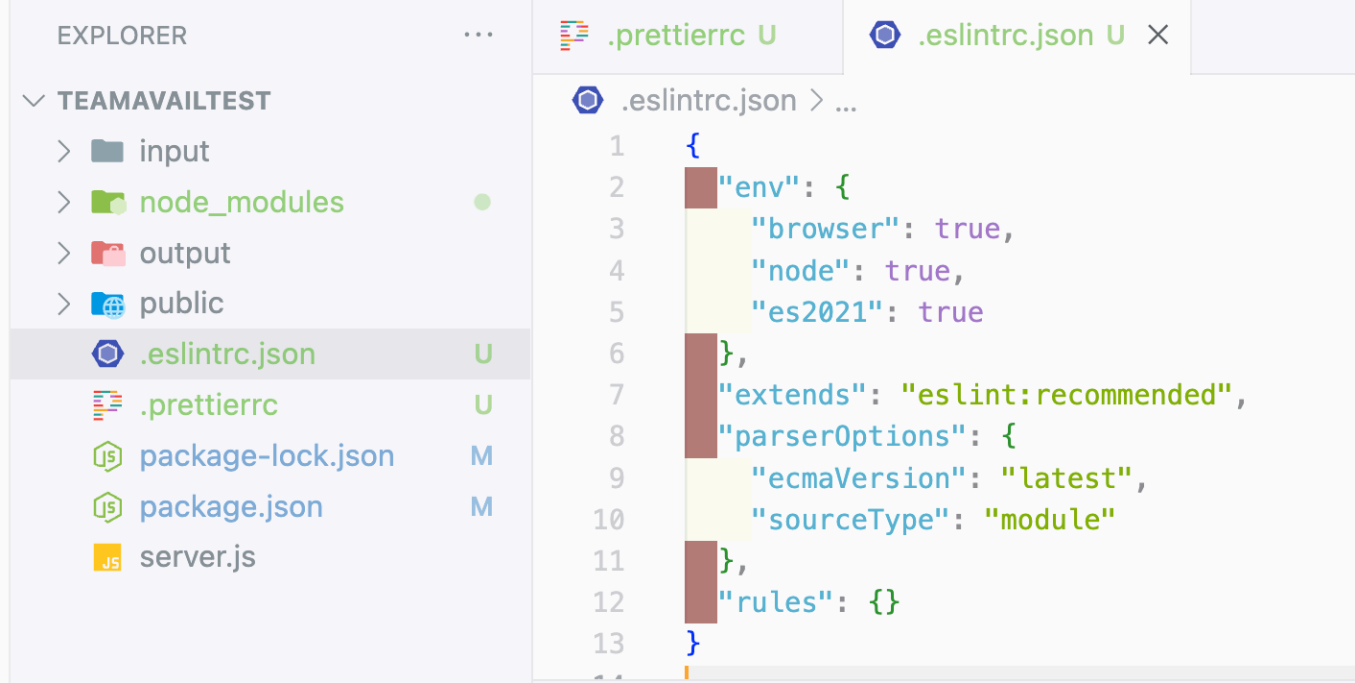
Task 1 Hands-on experience

- I started by **forking the project** to my own GitHub account.
- Then, I **cloned the repository** to my local machine, so I had the full project files available for development.
- Inside the project, I added the following configuration files to manage code quality:
  - `.eslintrc.json` – for JavaScript linting rules.
  - `.prettierrc` – to enforce consistent code formatting.

This setup ensures that both linting and formatting rules are applied consistently as I work on the project locally.



Prettier Test

After setting up the `.prettierrc` file, I ran a quick check to see if the existing code follows the formatting rules:

```
npx prettier --check .
```

The command scanned the project files and returned **errors**, indicating that some files did not follow the Prettier formatting rules.

> *This helped me identify areas in the code that needed formatting fixes before continuing with the pipeline setup.*

```
→  TeamavailTest git:(main) ✗ npx prettier --check .

Checking formatting...
[warn] input/selection.json
[warn] input/status.json
[warn] public/index.html
[warn] public/script.js
[warn] public/styles.css
[warn] Code style issues found in 5 files. Run Prettier with --write to fix.
```

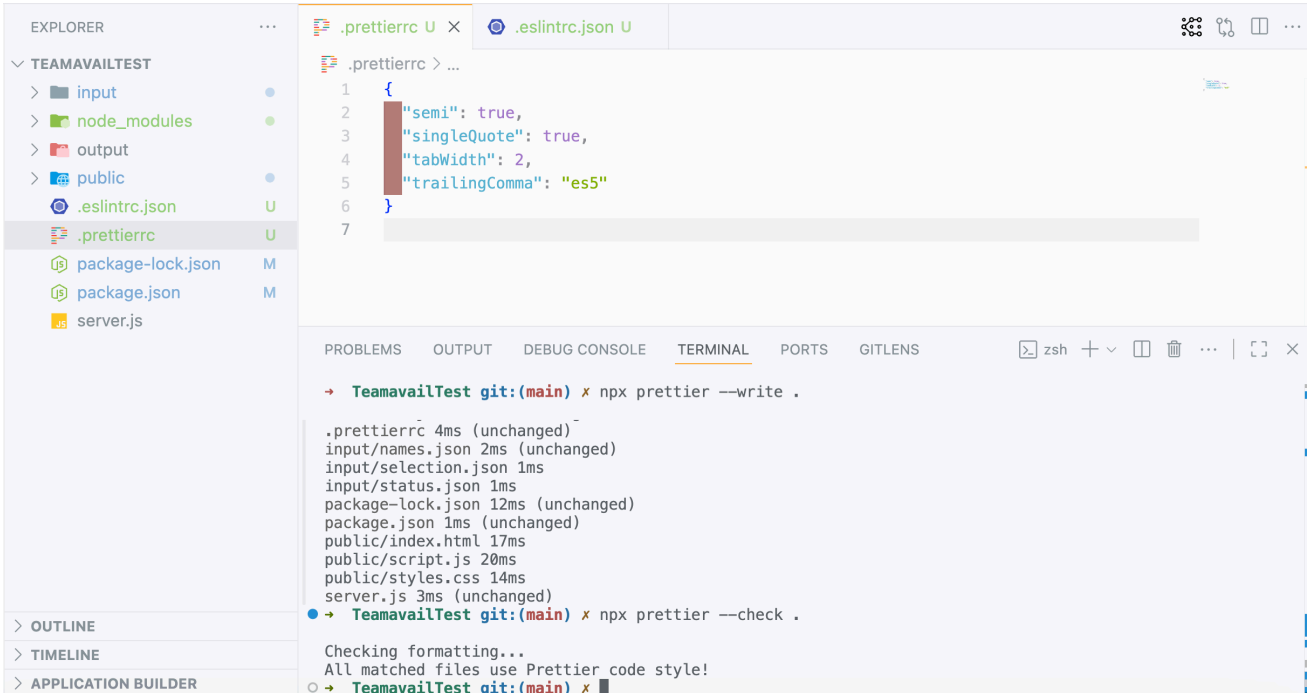
Fixing Prettier Errors

After seeing the formatting errors, I ran the following command to automatically fix them:

`npx prettier --write .`

This command **rewrites the code files** so that they comply with the rules defined in `.prettierrc`. Essentially, it saves time by automatically formatting everything instead of fixing each file manually.

The command successfully updated the files, and the terminal output confirmed that all formatting issues were resolved.

> *This ensured that the project had a consistent code style, which is crucial before moving on to linting, testing, and building the CI/CD pipeline.*
>
> 
>
>
> Next, I ran **ESLint** to check for potential code errors or issues:

`npx eslint .`

- I had to **install an older version** of ESLint because the latest version wasn't working properly on my machine.

- Using the older version worked perfectly, and **no errors were reported** in the project.

```
npm install --save-dev eslint@8

npm warn deprecated rimraf@3.0.2: Rimraf versions prior to v4 are no longer supported
npm warn deprecated @humanwhocodes/config-array@0.13.0: Use @eslint/config-array instead
npm warn deprecated eslint@8.57.1: This version is no longer supported. Please see https://eslint.org/vers
ion-support for other options.

added 94 packages, and audited 162 packages in 4s

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

found 0 vulnerabilities
➜  TeamavailTest git:(main) ✗ npx eslint .

➜  TeamavailTest git:(main) ✗
```

Creating the CI Script (`ci.sh`)

The next and most important step was to **automate everything locally** using a Bash script (`ci.sh`).

- Up to this point, I had been running **Prettier** and **ESLint** manually to check for formatting and code errors.
- The task required automating these checks, along with installing dependencies and starting the project, so I could run everything with a single command.

My Approach

Before writing the full CI/CD script with Docker and Docker Compose, I first created a **minimal version** to test the automation.

The script performs the following steps:

1. **Running Prettier** – automatically formats the code.
2. **Running ESLint** – checks the code for errors or issues.
3. **Installing dependencies** – ensures all required packages are available.
4. **Starting the project** – launches the app locally.

Making the Script Executable

To run the script, I had to make it executable using:

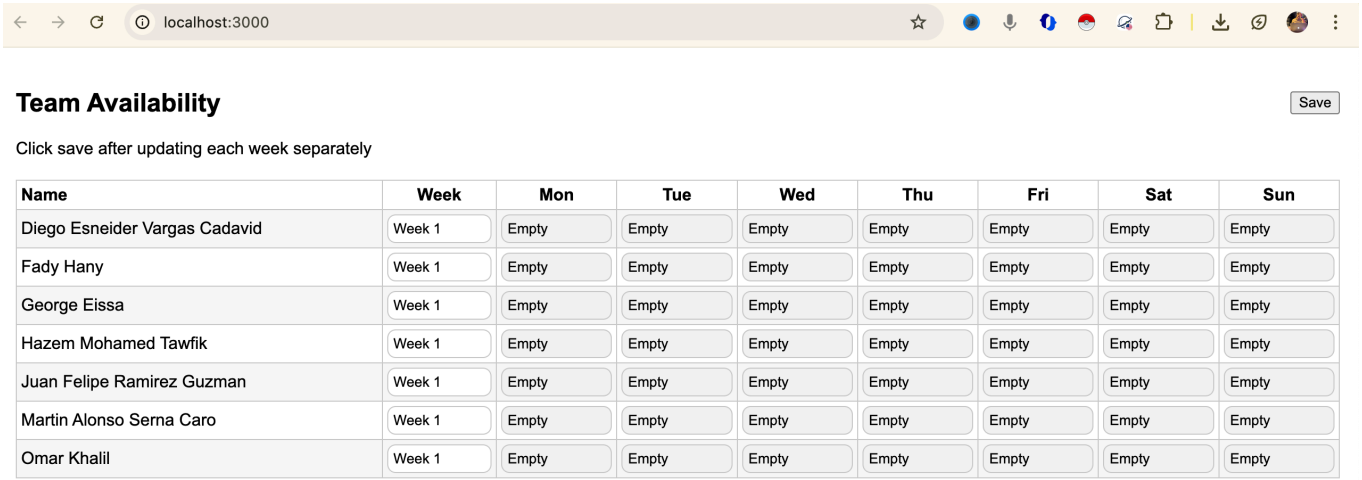
`chmod +x ci.sh`

Once the permissions were set, I ran the script:

`./ci.sh`

The command executed all the steps automatically, and the terminal output confirmed that everything worked as expected.

```
➜  TeamavailTest git:(main) ✗ chmod +x ci.sh
➜  TeamavailTest git:(main) ✗ ./ci.sh

=== Running Prettier (formatting) ===
.eslintrc.json 17ms (unchanged)
.prettierrc 4ms (unchanged)
input/names.json 2ms (unchanged)
input/selection.json 1ms (unchanged)
input/status.json 1ms (unchanged)
package-lock.json 12ms (unchanged)
package.json 1ms (unchanged)
public/index.html 15ms (unchanged)
public/script.js 21ms (unchanged)
public/styles.css 15ms (unchanged)
server.js 3ms (unchanged)
=== Running ESLint (code check) ===
=== Installing dependencies ===

up to date, audited 162 packages in 970ms

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

found 0 vulnerabilities
=== Starting the project ===

> version-1@1.0.0 start
> node server.js

Server running at http://localhost:3000
```



Dockerizing the Application

After confirming that all previous steps worked correctly (I like to **verify each step** to easily spot any errors), the next step was to **containerize the Node.js application**.

- I needed to create a **Dockerfile** that would build the image for my app.
- The Dockerfile, along with all the additional files I added (like `.eslintrc.json` and `.prettierrc`), was already pushed to my GitHub repository.

Building the Docker Image

I built the Docker image using the following command:

```
docker build -t teamavail-app .
```

- This created a Docker image named `teamavail-app`.
- Once the build was complete, I was ready to run the app in a container.
  -Screenshot while building the image :

```
➜  TeamavailTest git:(main) ✗ docker build -t teamavail-app .

[+] Building 119.4s (5/10)                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                        0.0s
 => => transferring dockerfile: 161B                                                        0.0s
 => [internal] load metadata for docker.io/library/node:20-alpine                           9.1s
 => [auth] library/node:pull token for registry-1.docker.io                                 0.0s
 => [internal] load .dockerignore                                                           0.0s
 => => transferring context: 2B                                                             0.0s
 => [1/5] FROM docker.io/library/node:20-alpine@sha256:eabac870db94f7342d6c33560d6613f188bbcf4  110.2s
 => => resolve docker.io/library/node:20-alpine@sha256:eabac870db94f7342d6c33560d6613f188bbcf4bb  0.0s
 => => sha256:127c05f5df6b075d5c314444a05d3de523268e2de5e9b235e7ba72aa60ed3c61 446B / 446B        0.7s
 => => sha256:c149c7c96aa9b49de8c5de3415d3692e81ced50773077ea0be1a0f3f36032234 1.26MB / 1.26MB   48.9s
 => => sha256:6e174226ea690ced550e5641249a412cdbefd2d09871f3e64ab52137a54ba606 0B / 4.13MB      110.1s
 => => sha256:5a8e8228254a218fbf68fbf8d7093ea99dfce63dd0a72ad0cc8be65e6da3f7c 1.05MB / 42.43MB  110.1s
 => [internal] load build context                                                           0.4s
 => => transferring context: 22.80MB                                                        0.3s
```

And it works !

```
➜  TeamavailTest git:(main) ✗ docker run -p 3000:3000 teamavail-app

Server running at http://localhost:3000
```

Setting Up Docker Compose

The next step was to create a `docker-compose.yml` file.

- This allows me to **run the project along with any additional services** easily (for example, Redis or PostgreSQL if we add them later).
- Docker Compose simplifies managing multiple containers and their dependencies, volumes, and ports in one place.

Running the Project with Docker Compose

Initially, I ran:

```
docker-compose up
```

However, I got the following error:

```
Error response from daemon: failed to set up container networking: driver failed programming external connectivity on endpoint teamavailtest-app-1 (c1ef5e851464165e7be32074c505ddc8a9331945f933905c1fd6da05b44746e0): Bind for 0.0.0.0:3000 failed: port is already allocated
```

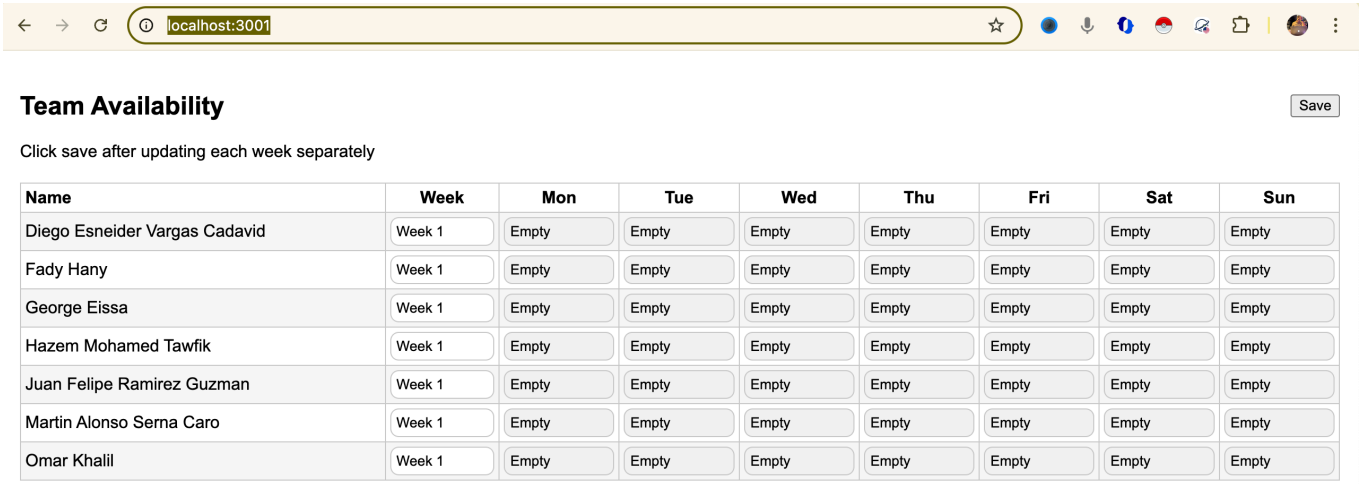
- This was expected because I already had the application running locally on **port 3000**.
- To solve this, I configured Docker to run the container on **port 3001**, while keeping the local app on port 3000.

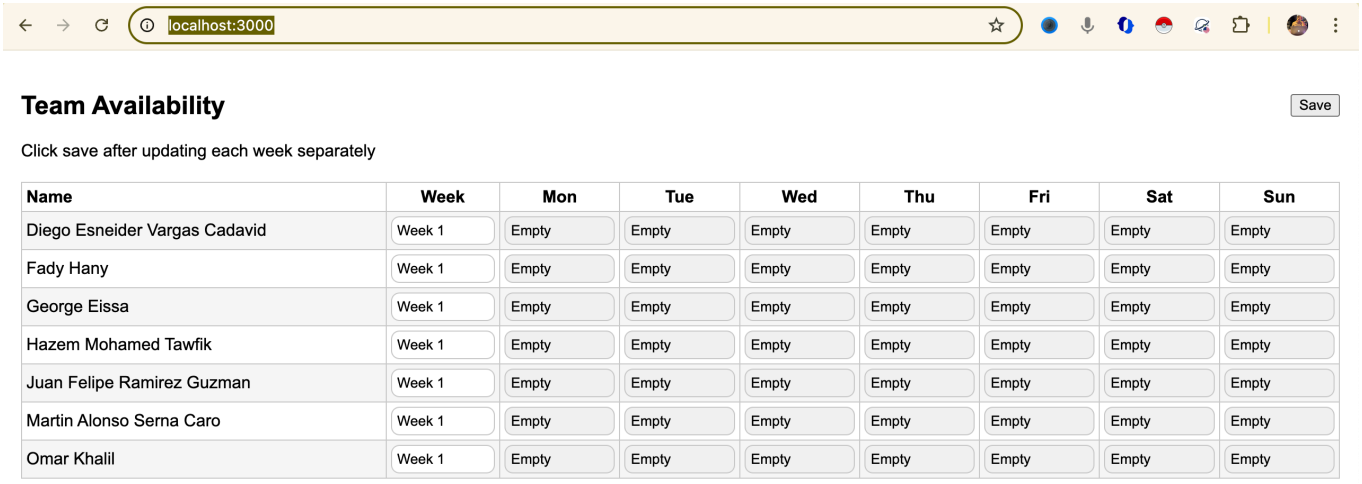
Final Setup

- Local app: `http://localhost:3000`
- Docker container: `http://localhost:3001`

Both instances ran **simultaneously without conflicts**, confirming that the Docker Compose setup works as intended.

_(Screenshots showing both the local app on port 3000 and the Docker container on port 3001 go here)





Adding Tests and Updating `ci.sh`

To make sure the project meets **100% of the task requirements**, I needed to **add automated testing**.

Adding Tests

1. I updated the `package.json` to include a test script:

```
"scripts": { "test": "jest", "start": "node server.js" }
```

2. Created a separate `tests` **folder** to hold test files.
   - For example, `server.test.js` contains tests for the application.

This ensures that the project can be tested automatically as part of the CI/CD pipeline.

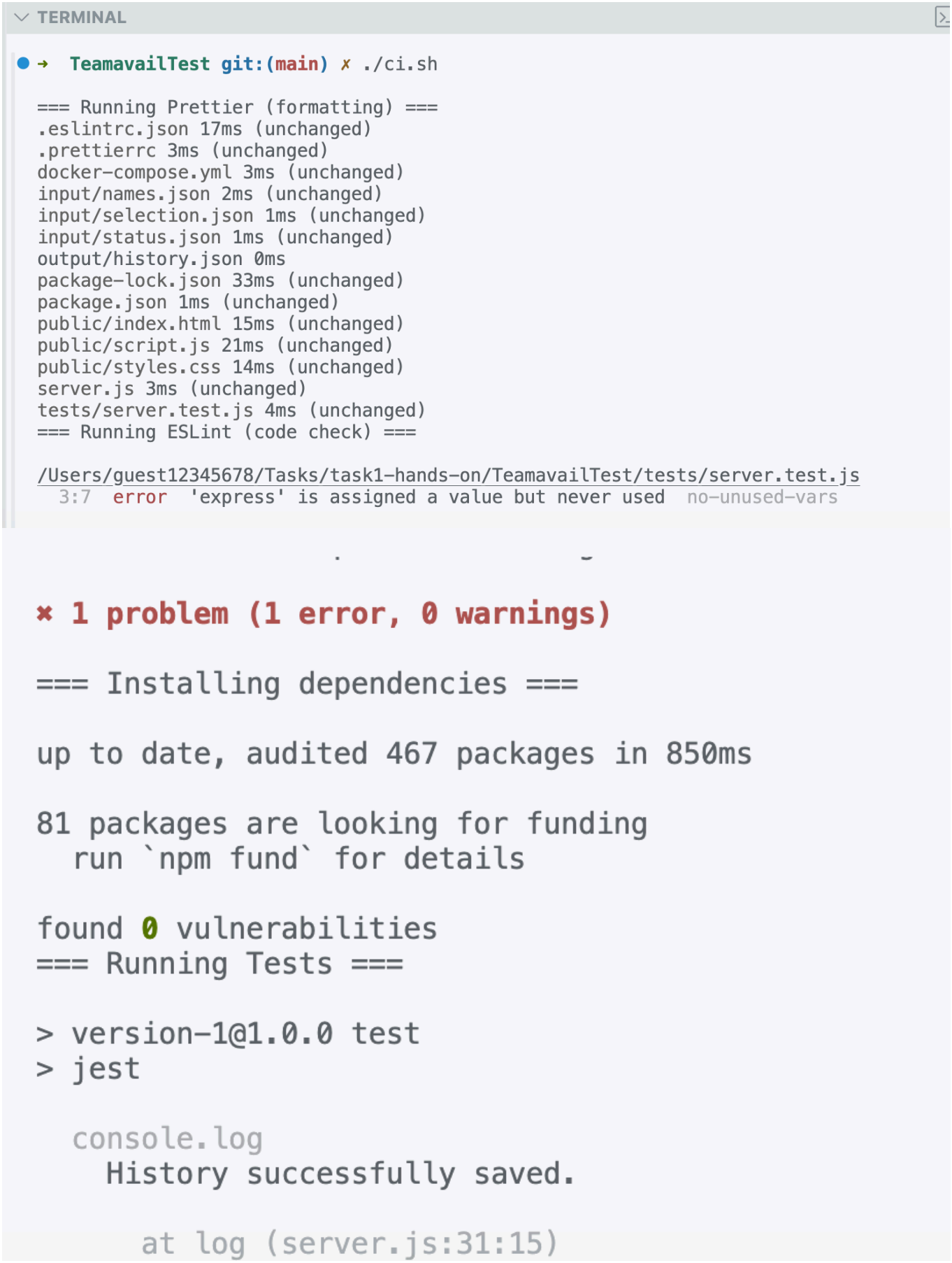
Updating `ci.sh`

After confirming that all previous steps worked correctly, I updated the `ci.sh` **script** to run the **full automated workflow**, including tests.

The updated script now runs:

1. **Prettier** – format the code automatically.
2. **ESLint** – check for linting issues.
3. **Tests** – run `jest` to verify the application works as expected.
4. **`npm install`** – install all dependencies.
5. **Docker build** – build the Docker image.
6. **Docker Compose up** – start the app along with any configured services.

> *Now, running `./ci.sh` executes the full workflow automatically, including formatting, linting, testing, dependency installation, and starting the app in Docker. This makes the project fully compliant with the task requirements.*


TERMINAL
→ TeamavailTest git:(main) ✗ ./ci.sh
=== Running Prettier (formatting) ===
.eslintrc.json 17ms (unchanged)
.prettierrc 3ms (unchanged)
docker-compose.yml 3ms (unchanged)
input/names.json 2ms (unchanged)
input/selection.json 1ms (unchanged)
input/status.json 1ms (unchanged)
output/history.json 0ms
package-lock.json 33ms (unchanged)
package.json 1ms (unchanged)
public/index.html 15ms (unchanged)
public/script.js 21ms (unchanged)
public/styles.css 14ms (unchanged)
server.js 3ms (unchanged)
tests/server.test.js 4ms (unchanged)
=== Running ESLint (code check) ===
/Users/guest12345678/Tasks/task1-hands-on/TeamavailTest/tests/server.test.js
3:7 error 'express' is assigned a value but never used no-unused-vars
✖ 1 problem (1 error, 0 warnings)
=== Installing dependencies ===
up to date, audited 467 packages in 850ms
81 packages are looking for funding
run `npm fund` for details
found 0 vulnerabilities
=== Running Tests ===
> version-1@1.0.0 test
> jest
console.log
History successfully saved.
at log (server.js:31:15)

```
➜  TeamavailTest git:(main) ✗ ./ci.sh

 PASS  tests/server.test.js
  API Tests
    ✓ GET / should return 200 (12 ms)
    ✓ POST /save-history should save JSON (26 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.274 s, estimated 1 s
Ran all test suites.
=== Building Docker image ===
[+] Building 4.6s (10/10) FINISHED                                  docker:desktop-li
 => [internal] load build definition from Dockerfile                                0
 => => transferring dockerfile: 161B                                                0
 => [internal] load metadata for docker.io/library/node:20-alpine                   0
 => [internal] load .dockerignore                                                   0
 => => transferring context: 2B                                                     0
 => [internal] load build context                                                   0
 => => transferring context: 1.09MB                                                 0
 => [1/5] FROM docker.io/library/node:20-alpine@sha256:eabac870db94f7342d6c33560d6  0
 => [5/5] COPY . .                                                              1.1s
 => exporting to image                                                          2.4s
 => => exporting layers                                                         1.8s
 => => exporting manifest sha256:05fafd390e86dc3dd9d006f0d8a9b31ab8634a5df802d4013bf80a00733c2a9  0.0s
 => => exporting config sha256:1e229f55209efec3c297dac4fc1f3b4c4cbb007583390a28132b00e9da3311ed   0.0s
 => => exporting attestation manifest sha256:d7512add570238d386b1da6610648212796d6b1539fd0703c70  0.0s
 => => exporting manifest list sha256:7a9e4f9210071e118f3c67ffb7923b14cfa2c25a6a89f3a4765aa30228  0.0s
 => => naming to docker.io/library/teamavail-app:latest                         0.0s
 => => unpacking to docker.io/library/teamavail-app:latest                      0.6s
=== Starting Docker Compose ===
WARN[0000] /Users/guest12345678/Tasks/task1-hands-on/TeamavailTest/docker-compose.yml: the attribute `v
ersion` is obsolete, it will be ignored, please remove it to avoid potential confusion
[+] Running 1/1
 ✔ Container teamavailtest-app-1  Running                                       0.0s
```

Test Warnings

While running the tests, everything worked fine except for **one warning** in `tests/server.test.js`:

```
3:7 error 'express' is assigned a value but never used no-unused-vars
```

- This is just a **linting warning** indicating that the `express` variable is imported but not used in the test file.
- It **does not affect the application** or any of the CI/CD workflow steps.

> *Warnings like this are common in test files, especially when placeholders or sample tests are added.*