2022

TED UNIVERSITY

CMPE_492 Final Report Secure Donation Chain

BURAK KATI
EGE TELLI
ÖZGÜR ÖZERTURAL
ÖMER MEKIN KARTAL

1. Introduction

With the introduction of “coins” to our lives, many people have wondered about the difference between the crypto currency and the real time stock market. After a certain time passed, people started realizing that the main difference is that stock markets have rules which control the companies. Today many people tend to keep their money in crypto currency for the sake of both investment and also, mainly in our country, to not be affected by inflation. Our project, Secure Donation Chain, mainly uses crypto currency and enables people to make transactions with it.

2. Main objective of SDC

The main objective of Secure Donation Chain is to enable people to both create and donate to campaigns with crypto currency to be able to avoid the “tax problem”. We aim to meet the needs of people in a secure and easy way. Since the current donations and campaigns created are subject to taxes, when an individual tries to donate to a certain campaign, a certain percentage of the money goes to the government. Another problem is that no individual is allowed to open a legal campaign just by themselves and collect donations for it.

To solve these two main problems we created the SDC website. SDC uses only crypto based transactions and allows its members to create any type of campaign, if necessary explanation is given. One can simply create a campaign for even buying a new mouse or just a shirt they liked, or they can create a campaign to help the children who have leukemia. The only necessity is for one to have a crypto currency wallet defined for them and since those wallets are uniquely defined for everyone, it will also be used as your ID in our system. After connection of the wallet there is no other condition that should be met.

3. External Resources

3.1. Libraries

3.1.1. Backend:

3.1.1.1. Cors: Stands for Cross Origin Resource Sharing. We are using this middleware to enable our client side to reach backend from multiple different ip numbers, broadening the domain for our resources.

3.1.1.2. BodyParser: Enables us to read the data that comes to our endpoints, enabling us to see more data that is carried in the request.

3.1.1.3. Jest: Jest is a javascript testing utility library that allows us to create test cases for both unit tests and integration tests. It provides useful information about the type safety of our system and its integrity.

3.1.2. Frontend:

3.1.2.1. **Vite with React:** Vite is a web app initializer for javascript , typescript and React or vue as different frameworks. We are using Vite with React to create components for our web-app in order to make it modular.

3.1.2.2. **Axios**: Axios is an http request manager that allows us to reach our API endpoints easily.

3.1.2.3. **Particles Bg:** This is a reactive particle background component for the background of your website. There are auto-generated backgrounds provided by the library, and it can be customized as you wish with custom codes. Therefore, it makes the background of your site look much more beautiful and different.

3.1.2.4. **React Swipeable:** This is a React component for swipeable views. For example, we used this for swiping trending campaigns as a carousel. Also it automatically swipes that carousel so it looks better than it is.

3.1.2.5. **Tailwind CSS:** The code of your application may be written and maintained more quickly with Tailwind CSS. You may style your application without writing unique CSS by utilizing this utility-first framework. Instead, you may utilize utility classes to regulate your application's padding, margin, color, font, shadow, and more.

3.1.2.6. **Slick Carousel:** A slideshow component for cycling through elements—images or slides of text—like a carousel. Front has taken the advantage of Slick carousel and extended it with dozens of new options. The carousel is a slideshow for cycling through a series of content, built with CSS 3D transforms and a bit of JavaScript. It works with a series of images, text, or custom markup. It also includes support for previous/next controls and indicators.

3.2. Components

3.2.1. Backend:

3.2.1.1. Services:

We went with services in terms of implementation of basic CRUD operations to advanced operations like filtering any data and the Base service does not differ for each model we have because the goal we had was to implement once, call multiple times. However, we have microservices that are childs of the Base service which implements the said functionality and these differ based on the model.

3.2.1.2. Controllers:

We have controllers for every endpoint and model we have and its purpose is to call the services' methods as needed when a request hits an endpoint and respond with appropriate response. Our login and error handling steps are handled here. Since we have access to both the requests and the models, it is the most logical way to handle those kinds of operations under the specific controller components. The error controller checks the request and tries to respond first, if an error occurs such as a database related error or a formatting error in the frontend data, we catch it and send appropriate response (e.g HTTP500 or HTTP404) to the frontend.

3.2.1.3. **Routes**

We specify the routes for specifying which functionality to be used at which endpoint exactly. For example, in the “api/users” endpoint, we call the userController’s appropriate methods for each type of request (e.g GET, POST, PUT, DELETE) .

3.2.1.4. **Models**

We map our entities’ data fields to mongoDB Schemas in a JSON-like format here and specify the type, validity conditions and limits.

3.2.1.5. **Configurator**

This is a middleware that runs before any request hits our endpoints, all it does is set the ports and log in to the cloud database system and inform us if there are any errors. Having a separate middleware for connection and setting up the server is especially important in terms of modularity.

3.2.1.6. **DotEnv**

We have a .env file containing our API keys and secrets. The configurator reads this .env file and makes necessary connections using this data. This enables us to keep the sensitive data from reaching all the internet as we can easily add this file to our .gitignore file.

3.2.2. Frontend:

3.2.2.1 **Metamask**: The Ethereum object will be injected into the window by the MetaMask app in the background, precisely like the Chrome extension does on a desktop. The remainder of the code just functions on mobile devices. MetaMask, many thanks! Every time a website loads, MetaMask, a browser plugin, will add an Ethereum object to the window object. The "Please Install Metamask!" message is displayed if the ethereum object is missing, which indicates that MetaMask is not installed. A list of all the accounts to whom the user granted access will be returned by the MetaMask API. We call the onConnected callback using the first available account, accounts[0].

3.2.2.2 **React-simply-carousel**: A straightforward, light-weight, fully controllable isomorphic (with support for SSR) carousel component for React.js. Both responsive and touch-enabled. autoplay and infinite options are supported. fully adaptable

3.2.2.3 **Ethereumjs**: The Blockchain package offers an Ethereum-compatible blockchain that stores a chain of blocks created using @ethereumjs/block and contains details about the chain's current canonical head block and the environment it is running in (e.g. the hard fork rules the current head block adheres to). The blockchain may be expanded with new blocks. With the help of the Blockchain.validateBlock() and Blockchain.consensus.validateConsensus() functions, validation makes sure that the block format complies with the established chain and consensus rules. The module also allows reorganization scenarios, for instance by enabling the addition of a new block using Blockchain.putBlock() that takes a different canonical path to the head than that provided by the head block that is now canonical.

3.3. Architecture

3.3.1. Server-Client Structure:

We are using RESTful APIs to enable communication between our client and server components. Since we are using REST, we have specific endpoints for every piece of data and data group. For example, there is “api/user” endpoint which sends all users or a specific user if an id is specified, as a response. This enables us to manage our data separately and maintain it easily. We could have used GraphQL, which is an API query language. It makes it easier to reach the data using queries so the system does relatively less HTTP requests. However, RESTful API’s are easier to implement and maintain. Also, the server is deployed on a service hosting platform called Railway. We make all our requests and get our responses from there.

3.3.2. Backend:

We are using an architecture that is similar to both MVC and Service Oriented Architecture for our backend. We model the data of each entity with necessary fields and for every model, and we are creating specific controllers for each model. We do our main computations in the controllers. We also have services that are parents to our controllers. We are using services and microservices as it makes our codebase more clean and we do not have to write anything more than once. Also it makes our system highly modular, reducing the risk of losing functionality when introducing a new update to the system.

Aside from those, we have routers, and middlewares which are essential to our server. All a router does is specify the endpoint for that route and use the corresponding controller for it. For example, we have a URL structure for our backend such as "http://url/api/{componentName}" where the url is where our backend server is published, the componentName is the specific data object we want to reach or modify. Middlewares are similar to microservices, when a request is made to the server, middlewares are executed first instead of the controllers in the router. This enables us to create an extra level of layer to our API endpoints making it highly modular.

4. Test Process

4.1. What to test

4.1.1. **Backend:** Different components with unique attribute fields and some edge unit testing cases. We test for crud operations and communication integrity here.

4.1.2. **Frontend:** Cross Browser Test, Jest Unit Test and Jest Component Test

4.2. Test cases

4.2.1. Backend:

4.2.1.1. Case 1

*Test the /users get endpoint,

*test the /users post endpoint,

* test the /users post endpoint with the same input,

* test the /users update endpoint. (Figure 1)

We validate these methods are working with verifying that the response statusCode comes back as 200 as intended or 500(depending on the situation), and examining the body. For example, the server returns back the updated item as a response if a put request is made to it.

4.2.1.2. Case 2

Test the api/users delete endpoint via checking the message body we send to the client as the deleted object's id and an informative message. (Figure 2)

4.2.1.3. Case 3

Test the api/wallets get an endpoint via checking the message body and the status code that comes back from the server. (figure 3)

4.2.1.4. Case 4 Test the api/wallets POST endpoint via checking the message body, if it contains the data we just sent to the server, and the status code. (Figure 4)

4.2.1.5. Case 5

Test the api/wallets PUT endpoint via checking the response status code. (Figure 8)

4.2.2. Frontend:

4.2.2.1 Case 1

Cross Browser Test:

Cross-browser testing, a kind of non-functional testing, enables you to determine whether your website functions as intended when accessed from:

Various combinations of browser and operating system, i.e., on widely used browsers like Firefox, Chrome, Edge, and Safari—on any of the widely used operating systems including Windows, macOS, iOS, and Android.

Users can read and interact with your website on a variety of common devices, such as smartphones, tablets, desktops, and laptops, among others.

Supportive Tools In other words, people with disabilities can use the website using assistive devices like screen readers.(Figure 5, Figure 6)

4.2.2.2 Case 2

Jest Component Test:

Component Testing is a testing method that tests an individual unit of software in isolation. Component testing for React Apps means testing an individual React Component.(Figure 7)

4.3. Bugs & Challenges

Our backend unit tests successfully worked by jest testing. It checked the status code of users create delete and update post results and they all passed. However when we were testing frontend, cross browser test passed but, jest component tests didn’t passed. We couldn’t understand why but there is a problem with jest library that it can not test our components because of our css and our javascript syntax -Jest failed to parse a file. This happens e.g. when your code or its dependencies use non-standard JavaScript syntax, or when Jest is not configured to support such syntax.-.

5. Potential Enhancements & Conclusion

Since this version is the launch version, there will be upgrades and enhancements to the project. Sample enhancements which are being planned right now are:

- Current Transactions;

  which will be a widget showing the current largest transaction being made to the campaign, if the person who donates the money wishes to be anonymous, name can be blurred.

- Following a campaign and notification;

  This enhancement is a quality of life change for the user. Users will be able to turn on notifications about the campaign and follow another campaign's process.

- Dependency on tester feedback;

  Rest of the enhancements will be provided after the users test the website and our team will take action according to the feedback given from the users.

To sum up Secure Donation Chain it can be said that since a lot of people lost trust in current associations and many others who wish to start a campaign are faced with large paperwork and obstacles, this website will be the solution. The loss of money on government's tax policies are also being bypassed thanks to this website. Any individual can create or donate to any campaign they wish. In our vision this website will also be an opportunity for those who wish to create a startup project and could not find any funding. Any project, any campaign, any idea can be supported thanks to SDC.

6. Glossary

Jest: Jest is a delightful JavaScript Testing Framework with a focus on simplicity. It works with projects using: Babel, TypeScript, Node, React, Angular, Vue and more.

SDC: Secure Donation Chain

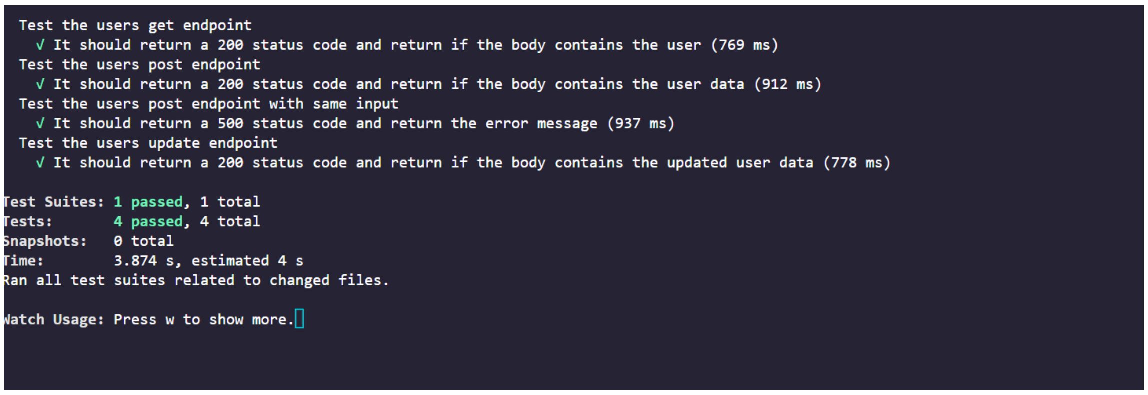
REST: RESTful API is an interface that two computer systems use to exchange information securely over the internet. Most business applications have to communicate with other internal and third-party applications to perform various tasks.

CRUD: Create, Read, Update, and Delete (CRUD) are the four basic functions that models should be able to do, at most.

GraphQL: GraphQL is a query language for APIs and a runtime for fulfilling those queries with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.

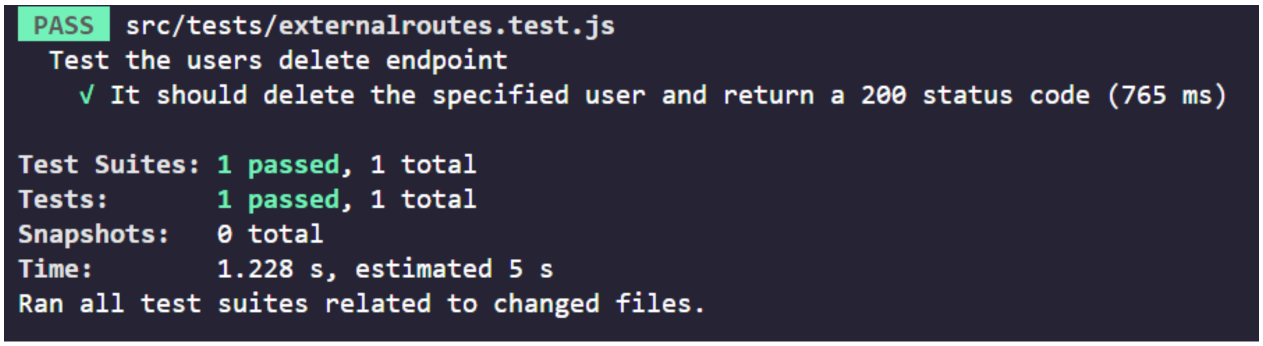
MVC: Model–view–controller (MVC) is a software architectural pattern commonly used for developing user interfaces that divide the related program logic into three interconnected elements.

7. APPENDIX



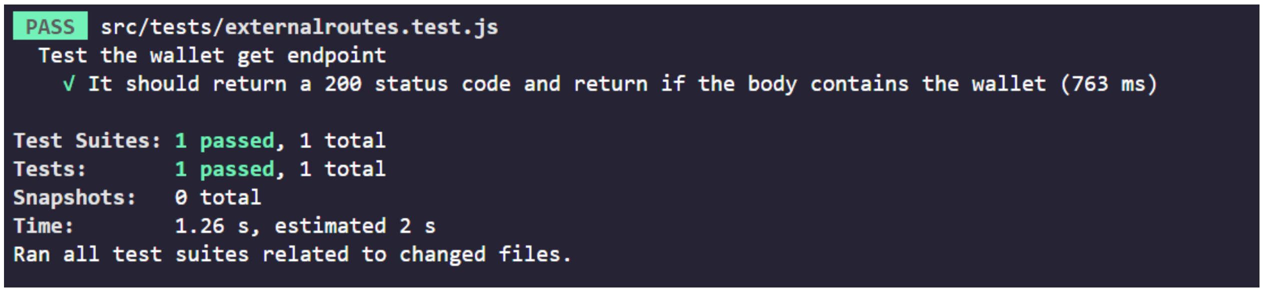
```
Test the users get endpoint
  √ It should return a 200 status code and return if the body contains the user (769 ms)
Test the users post endpoint
  √ It should return a 200 status code and return if the body contains the user data (912 ms)
Test the users post endpoint with same input
  √ It should return a 500 status code and return the error message (937 ms)
Test the users update endpoint
  √ It should return a 200 status code and return if the body contains the updated user data (778 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        3.874 s, estimated 4 s
Ran all test suites related to changed files.

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

Figure 1: Test results for users get, post, put endpoints.



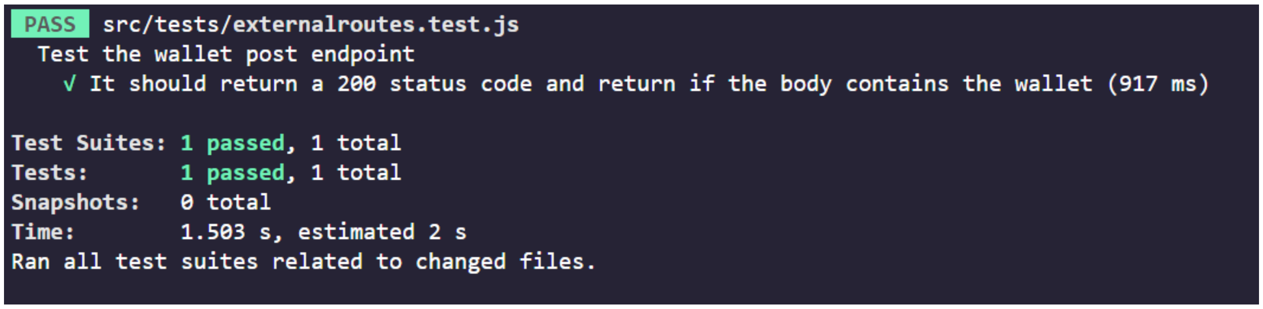
```
 PASS  src/tests/externalroutes.test.js
  Test the users delete endpoint
    √ It should delete the specified user and return a 200 status code (765 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.228 s, estimated 5 s
Ran all test suites related to changed files.
```

Figure 2: Test Results for Users delete endpoint



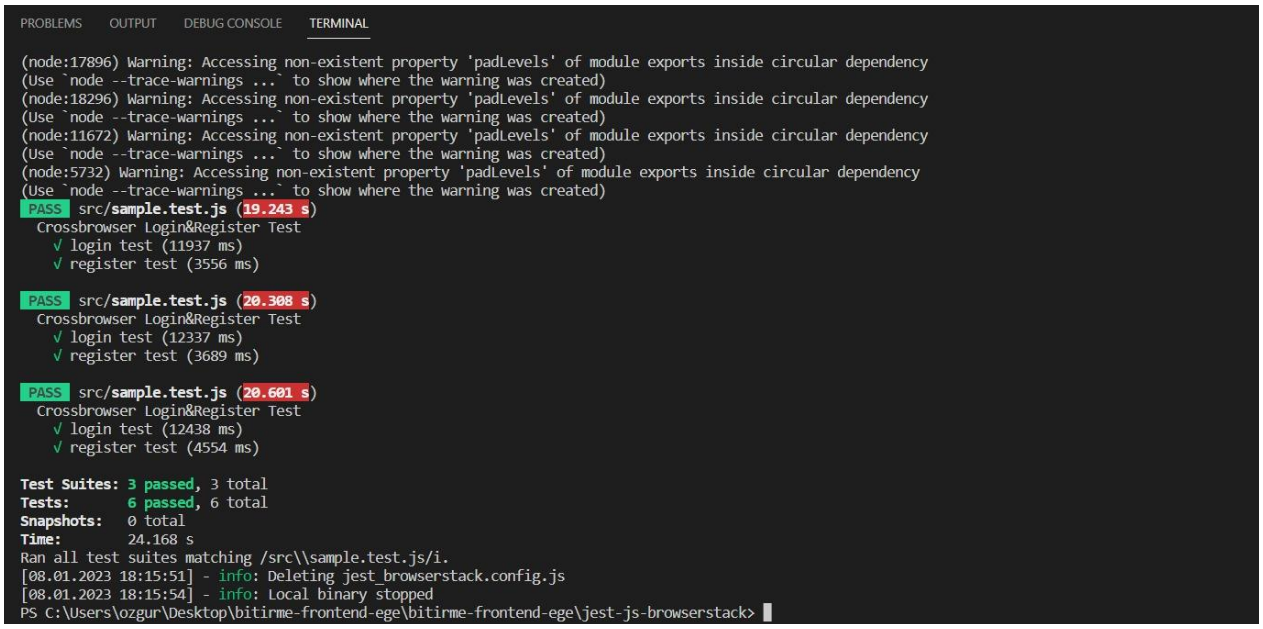
```
 PASS  src/tests/externalroutes.test.js
  Test the wallet get endpoint
    √ It should return a 200 status code and return if the body contains the wallet (763 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.26 s, estimated 2 s
Ran all test suites related to changed files.
```

Figure 3: Test Results for Wallet get endpoint



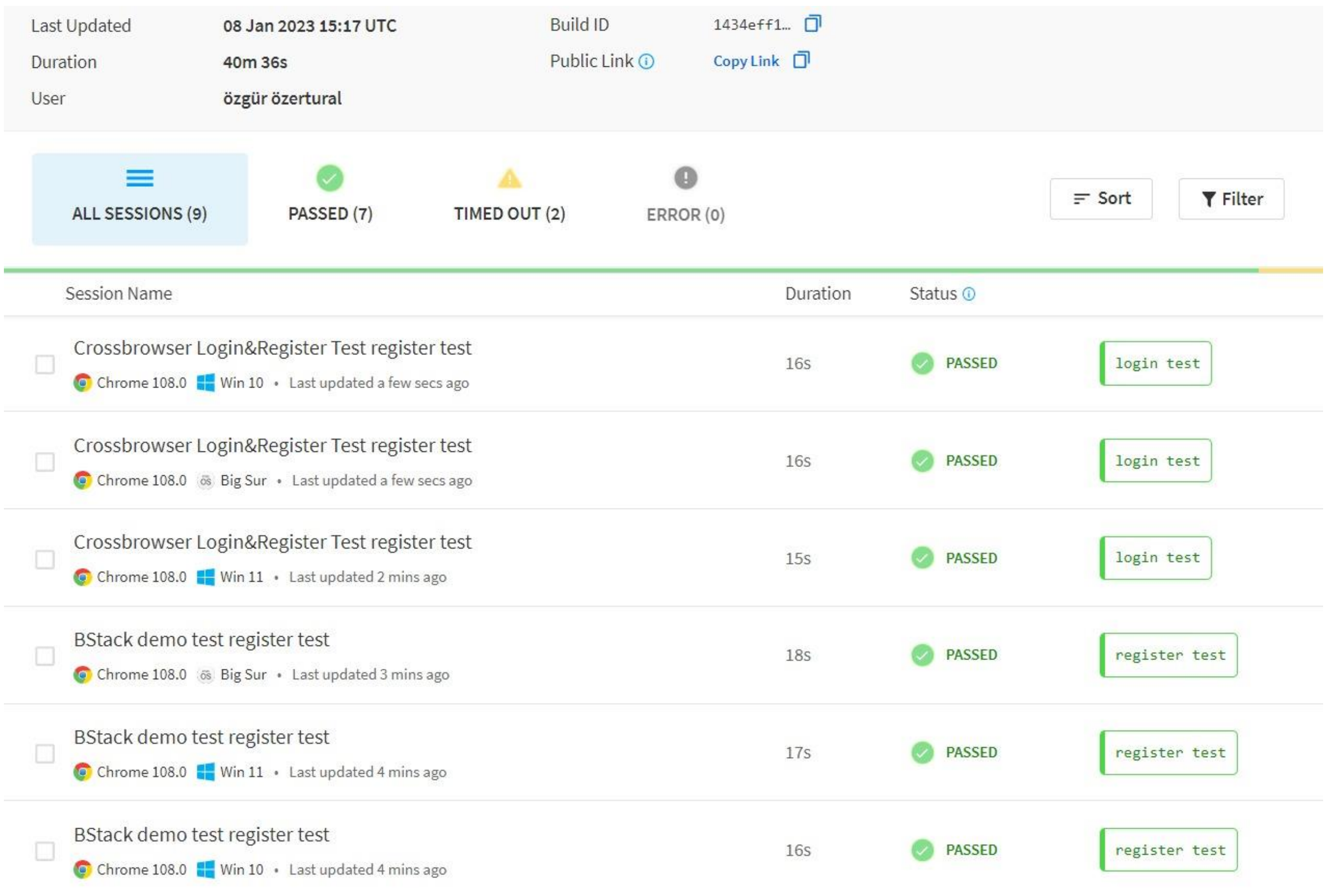
```
 PASS  src/tests/externalroutes.test.js
  Test the wallet post endpoint
    √ It should return a 200 status code and return if the body contains the wallet (917 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.503 s, estimated 2 s
Ran all test suites related to changed files.
```

Figure 4: Test Results for Wallet Post endpoint



```
(node:17896) Warning: Accessing non-existent property 'padLevels' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:18296) Warning: Accessing non-existent property 'padLevels' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:11672) Warning: Accessing non-existent property 'padLevels' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
(node:5732) Warning: Accessing non-existent property 'padLevels' of module exports inside circular dependency
(Use `node --trace-warnings ...` to show where the warning was created)
 PASS  src/sample.test.js (19.243 s)
  Crossbrowser Login&Register Test
    √ login test (11937 ms)
    √ register test (3556 ms)

 PASS  src/sample.test.js (20.308 s)
  Crossbrowser Login&Register Test
    √ login test (12337 ms)
    √ register test (3689 ms)

 PASS  src/sample.test.js (20.601 s)
  Crossbrowser Login&Register Test
    √ login test (12438 ms)
    √ register test (4554 ms)

Test Suites: 3 passed, 3 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        24.168 s
Ran all test suites matching /src\\sample.test.js/i.
[08.01.2023 18:15:51] - info: Deleting jest_browserstack.config.js
[08.01.2023 18:15:54] - info: Local binary stopped
PS C:\Users\ozgur\Desktop\bitirme-frontend-ege\bitirme-frontend-ege\jest-js-browserstack>
```

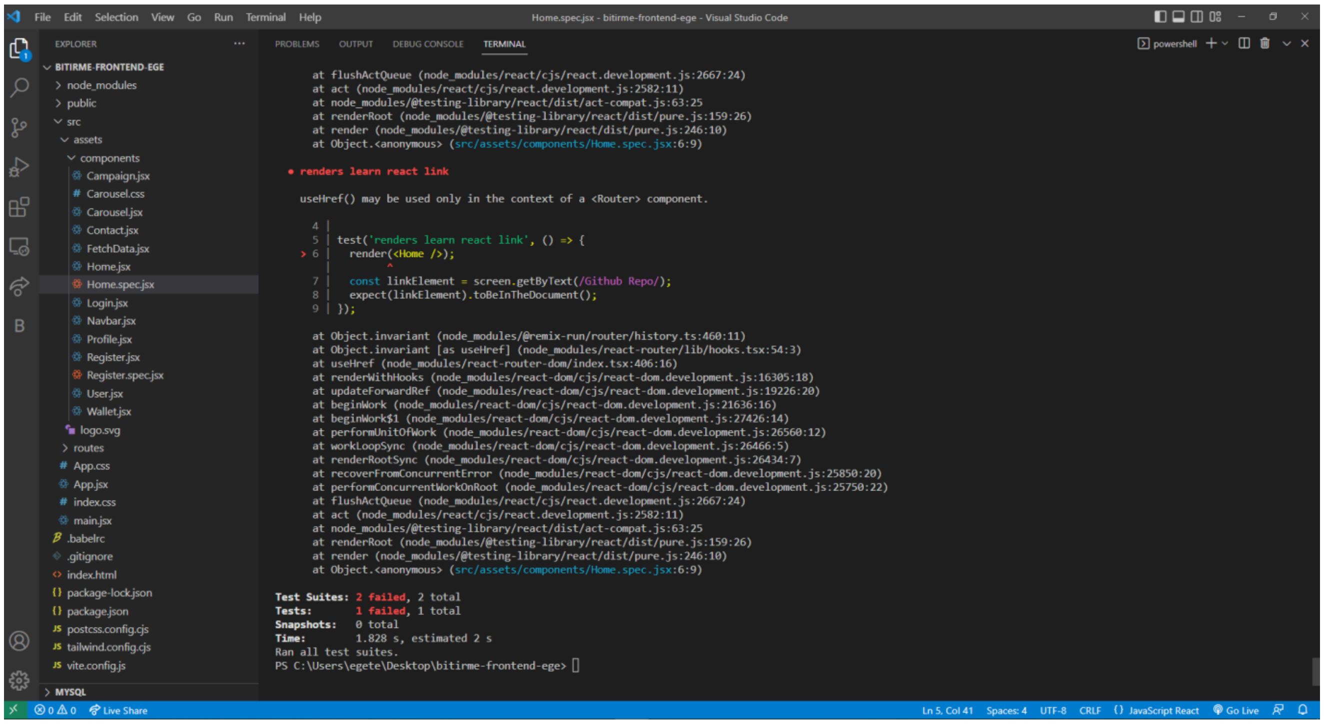
Figure 5: Login and Register test with Cross Browser Test



Last Updated 08 Jan 2023 15:17 UTC
Duration 40m 36s
User özgür özertural
Build ID 1434eff1...
Public Link Copy Link

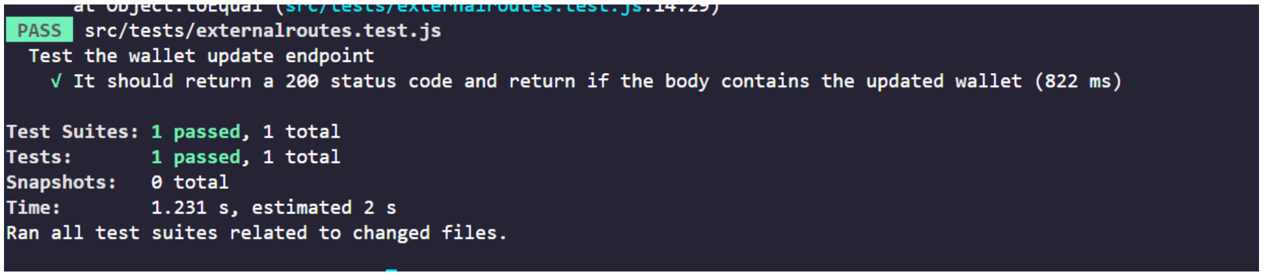
Session Name	Duration	Status	
Crossbrowser Login&Register Test register test Chrome 108.0 Win 10 • Last updated a few secs ago	16s	PASSED	login test
Crossbrowser Login&Register Test register test Chrome 108.0 Big Sur • Last updated a few secs ago	16s	PASSED	login test
Crossbrowser Login&Register Test register test Chrome 108.0 Win 11 • Last updated 2 mins ago	15s	PASSED	login test
BStack demo test register test Chrome 108.0 Big Sur • Last updated 3 mins ago	18s	PASSED	register test
BStack demo test register test Chrome 108.0 Win 11 • Last updated 4 mins ago	17s	PASSED	register test
BStack demo test register test Chrome 108.0 Win 10 • Last updated 4 mins ago	16s	PASSED	register test

Figure 6: Browserstack.com output about login and register tests



```
      at flushActQueue (node_modules/react/cjs/react.development.js:2667:24)
      at act (node_modules/react/cjs/react.development.js:2582:11)
      at node_modules/@testing-library/react/dist/act-compat.js:63:25
      at renderRoot (node_modules/@testing-library/react/dist/pure.js:159:26)
      at render (node_modules/@testing-library/react/dist/pure.js:246:10)
      at Object.<anonymous> (src/assets/components/Home.spec.jsx:6:9)

  ● renders learn react link

    useHref() may be used only in the context of a <Router> component.

      4 |
      5 | test('renders learn react link', () => {
    > 6 |   render(<Home />);
        |          ^
      7 |   const linkElement = screen.getByText(/Github Repo/);
      8 |   expect(linkElement).toBeInTheDocument();
      9 | });

      at Object.invariant (node_modules/@remix-run/router/history.ts:460:11)
      at Object.invariant [as useHref] (node_modules/react-router/lib/hooks.tsx:54:3)
      at useHref (node_modules/react-router-dom/index.tsx:406:16)
      at renderWithHooks (node_modules/react-dom/cjs/react-dom.development.js:16305:18)
      at updateForwardRef (node_modules/react-dom/cjs/react-dom.development.js:19226:20)
      at beginWork (node_modules/react-dom/cjs/react-dom.development.js:21636:16)
      at beginWork$1 (node_modules/react-dom/cjs/react-dom.development.js:27426:14)
      at performUnitOfWork (node_modules/react-dom/cjs/react-dom.development.js:26560:12)
      at workLoopSync (node_modules/react-dom/cjs/react-dom.development.js:26466:5)
      at renderRootSync (node_modules/react-dom/cjs/react-dom.development.js:26434:7)
      at recoverFromConcurrentError (node_modules/react-dom/cjs/react-dom.development.js:25850:20)
      at performConcurrentWorkOnRoot (node_modules/react-dom/cjs/react-dom.development.js:25750:22)
      at flushActQueue (node_modules/react/cjs/react.development.js:2667:24)
      at act (node_modules/react/cjs/react.development.js:2582:11)
      at node_modules/@testing-library/react/dist/act-compat.js:63:25
      at renderRoot (node_modules/@testing-library/react/dist/pure.js:159:26)
      at render (node_modules/@testing-library/react/dist/pure.js:246:10)
      at Object.<anonymous> (src/assets/components/Home.spec.jsx:6:9)

Test Suites: 2 failed, 2 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        1.828 s, estimated 2 s
Ran all test suites.
PS C:\Users\egete\Desktop\bitirme-frontend-ege>
```

Figure 7: Failed Jest Component Test



```
      at Object.toEqual (src/tests/externalroutes.test.js:14:29)
 PASS  src/tests/externalroutes.test.js
  Test the wallet update endpoint
    √ It should return a 200 status code and return if the body contains the updated wallet (822 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.231 s, estimated 2 s
Ran all test suites related to changed files.
```

Figure 8: Test Results for wallet put endpoint

8. References

https://www.digitalocean.com/community/tutorials/how-to-add-login-authentication-to-react-applications

https://morioh.com/p/afaa04e8cee8

https://docs.walletconnect.com/1.0/quick-start/dapps/react-native

https://tailwindui.com/?ref=top

https://docs.metamask.io/guide/getting-started.html#basic-considerations

https://medium.com/tinyso/how-to-create-the-responsive-and-swipeable-carousel-slider-component-in-react-99f433364aa0

https://railway.app/