Testing Documentation

7th May 2021

Students:
Janesh Sharma - 17473124
Paul McNally - 17318221

Supervisor:
Dr. Stephen Blott

Table Of Contents

Glossary

- **React** - Open-source, front end, JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies. [1]
- **CI/CD** - Refers to the combined practices of continuous integration and either continuous delivery or continuous deployment. CI/CD bridges the gaps between development and operation activities and teams by enforcing automation in building, testing and deployment of applications.
- **MongoDB** - Cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. [2]
- **Jest** - Jest is a JavaScript testing framework maintained by Facebook, Inc. designed and built by Christoph Nakazawa with a focus on simplicity and support for large web applications. [3]
- **Docker -** Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. Containers are

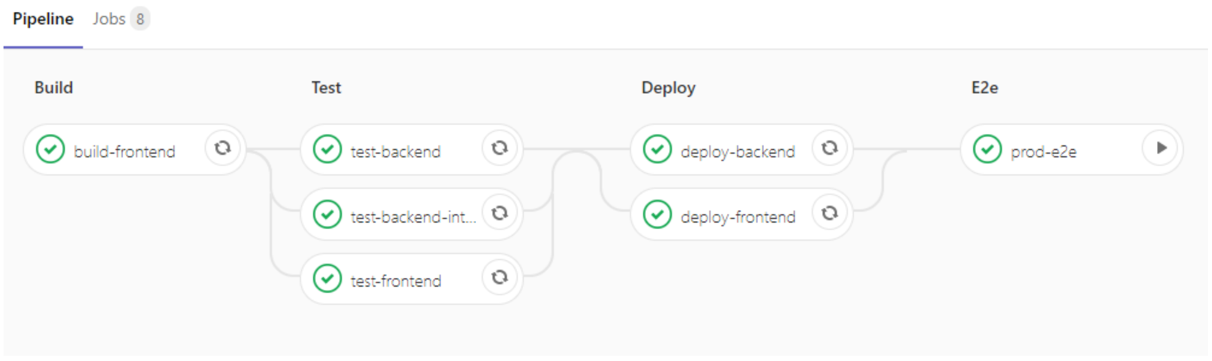
isolated from one another and bundle their own software, libraries and configuration files. [4]

- **Cypress -** Cypress is a next generation front end testing tool built for the modern web. Cypress enables developers to write end to end tests. [5]
- **React Testing Library** - The React Testing Library is a very light-weight solution for testing React components. It provides light utility functions on top of react-dom and react-dom/test-utils, in a way that encourages better testing practices. [6]
- **K6** - k6 is an open source load testing tool and SaaS for engineering teams. [7]
- **Lighthouse** - Lighthouse is an open-source, automated tool for improving the quality of web pages. [8]
- **Postman** - Postman is a collaboration platform for API development. [9]

Continuous Integration

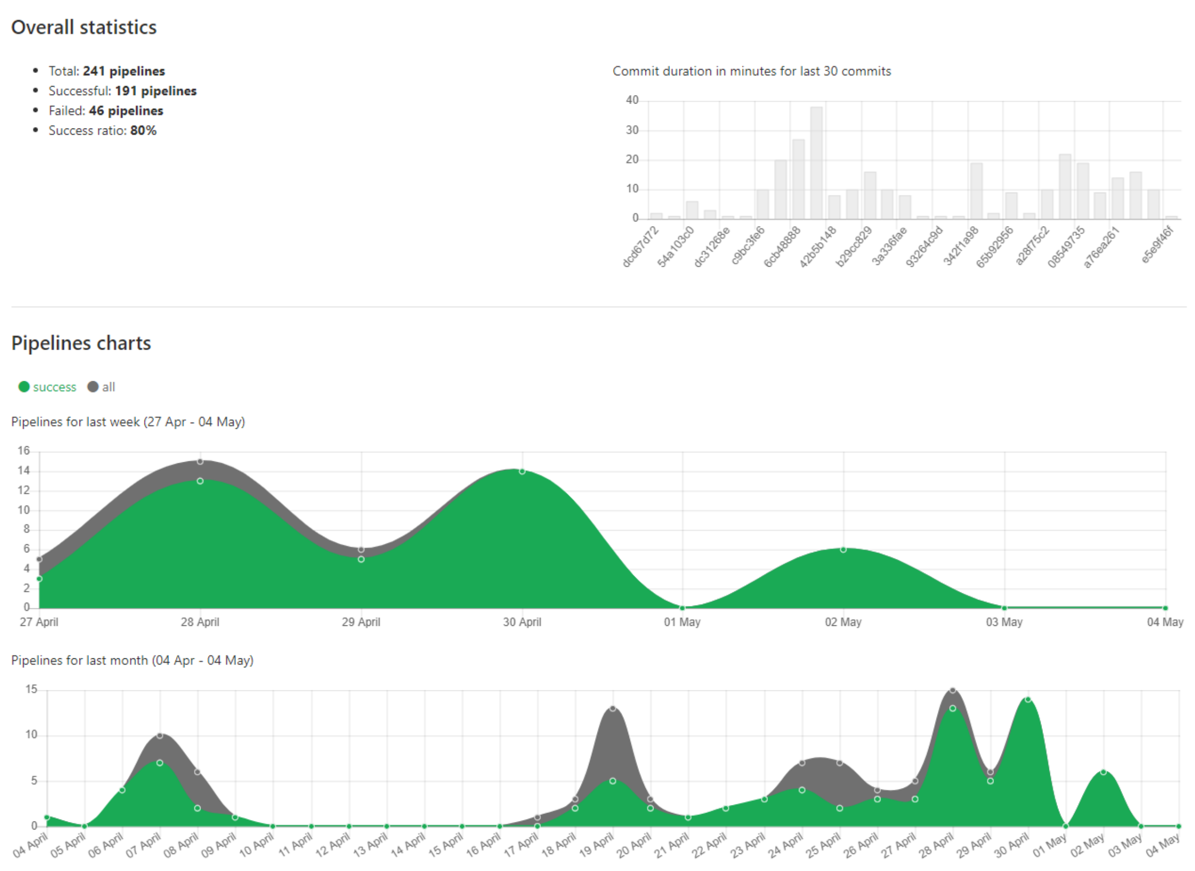
As you can see in the image below we have 7 jobs in our pipeline and 4 stages:

1. **Build**: Here the code is built to ensure it can be compiled without error.
2. **Test**: Here the unit and integration tests are executed.
3. **Deploy**: Here the code is deployed to a production environment.
4. **E2e**: Here the e2e tests are executed against the production environment.



The Test stage executes the unit and integration tests. The E2e stage executes end to end tests against the production deployed server. This stage is delayed by 10 mins as sometimes there are caching issues with firebase in which the new version is not available and can cause the stage to fail.

Below you can see the statistics for our pipelines over the course of the project. We have executed over 241 pipelines, 191 of which were successful and 46 of which failed. Overall we have a pipeline success ratio of 80%.



Overall statistics

- Total: **241 pipelines**
- Successful: **191 pipelines**
- Failed: **46 pipelines**
- Success ratio: **80%**

Pipelines charts

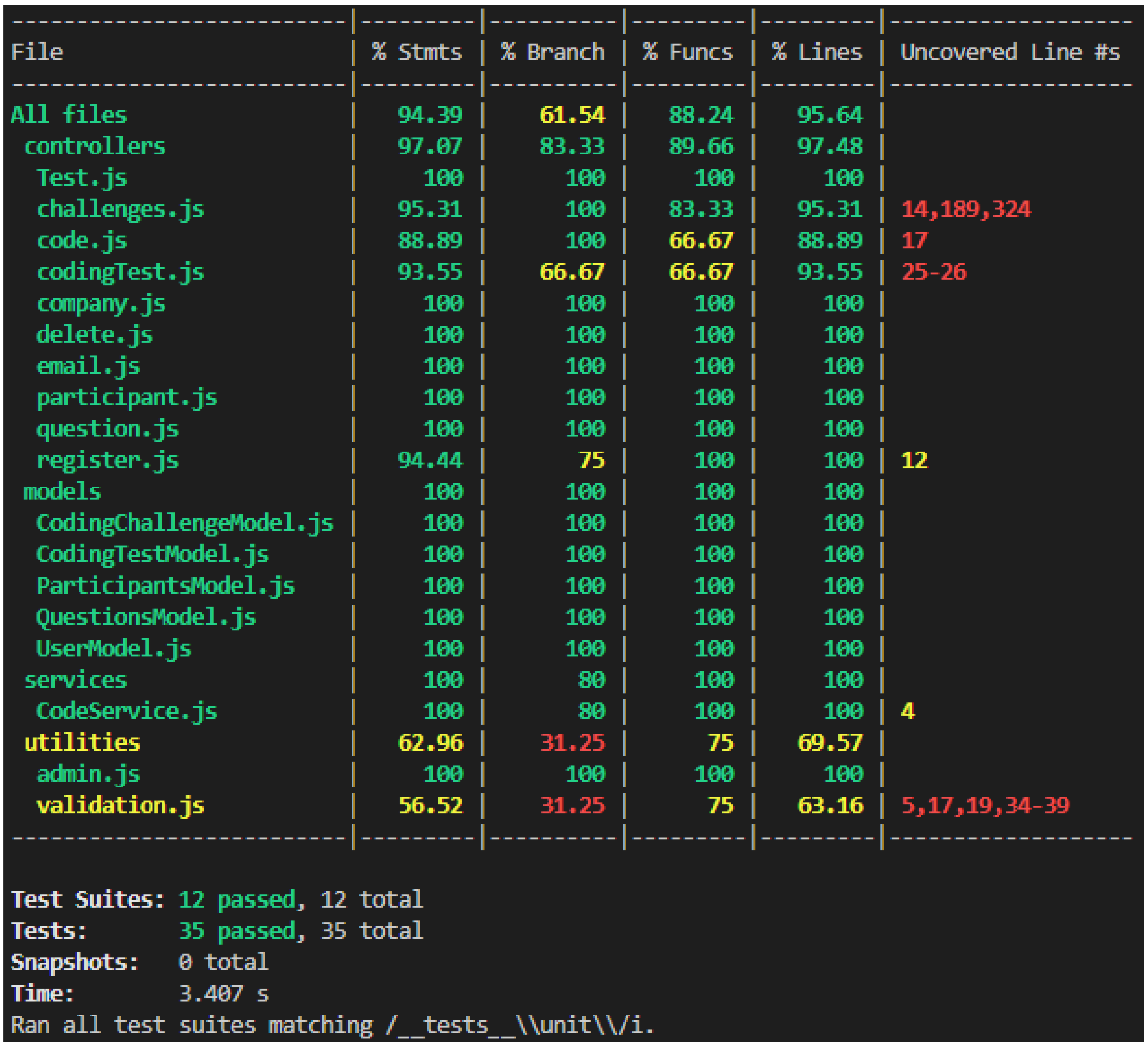
● success ● all

Unit Testing

All code in our project underwent heavy unit testing.

Backend

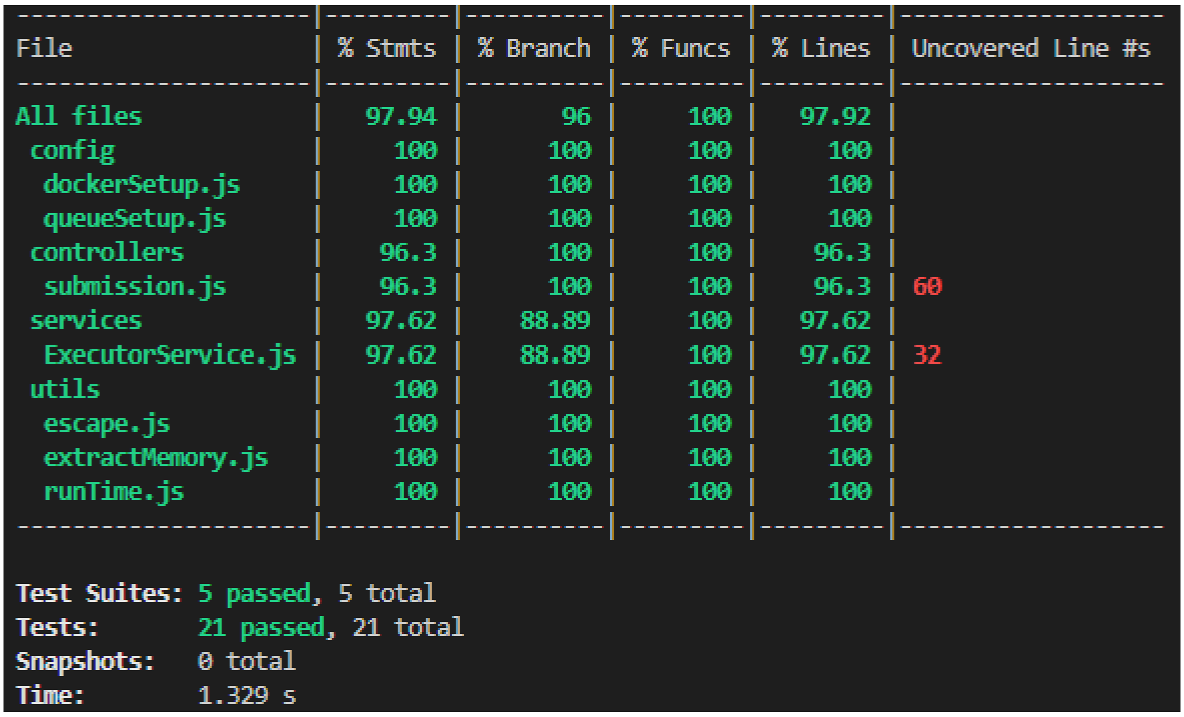
Below is a coverage report showing the coverage of our unit tests in the backend. As you can see from the report we have a total test coverage of 94.39% across all statements. We also have 12 test suites with 35 total tests, which execute using the jest library in ~3.4 seconds.



```
--------------------------|---------|----------|---------|---------|-------------------
File                      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------------------|---------|----------|---------|---------|-------------------
All files                 |   94.39 |    61.54 |   88.24 |   95.64 |
 controllers              |   97.07 |    83.33 |   89.66 |   97.48 |
  Test.js                 |     100 |      100 |     100 |     100 |
  challenges.js           |   95.31 |      100 |   83.33 |   95.31 | 14,189,324
  code.js                 |   88.89 |      100 |   66.67 |   88.89 | 17
  codingTest.js           |   93.55 |    66.67 |   66.67 |   93.55 | 25-26
  company.js              |     100 |      100 |     100 |     100 |
  delete.js               |     100 |      100 |     100 |     100 |
  email.js                |     100 |      100 |     100 |     100 |
  participant.js          |     100 |      100 |     100 |     100 |
  question.js             |     100 |      100 |     100 |     100 |
  register.js             |   94.44 |       75 |     100 |     100 | 12
 models                   |     100 |      100 |     100 |     100 |
  CodingChallengeModel.js |     100 |      100 |     100 |     100 |
  CodingTestModel.js      |     100 |      100 |     100 |     100 |
  ParticipantsModel.js    |     100 |      100 |     100 |     100 |
  QuestionsModel.js       |     100 |      100 |     100 |     100 |
  UserModel.js            |     100 |      100 |     100 |     100 |
 services                 |     100 |       80 |     100 |     100 |
  CodeService.js          |     100 |       80 |     100 |     100 | 4
 utilities                |   62.96 |    31.25 |      75 |   69.57 |
  admin.js                |     100 |      100 |     100 |     100 |
  validation.js           |   56.52 |    31.25 |      75 |   63.16 | 5,17,19,34-39
--------------------------|---------|----------|---------|---------|-------------------

Test Suites: 12 passed, 12 total
Tests:       35 passed, 35 total
Snapshots:   0 total
Time:        3.407 s
Ran all test suites matching /__tests__\\unit\\/i.
```

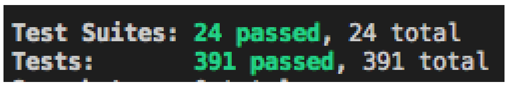
Remote Code Execution API

Below is a coverage report showing the coverage of our unit tests in the remote code execution API. As you can see from the report we have a total test coverage of 97.94% across all statements. We also have 5 test suites with 21 total tests, which execute using the jest library in ~1.3 seconds.


```
---------------------|---------|----------|---------|---------|-------------------
File                 | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
---------------------|---------|----------|---------|---------|-------------------
All files            |   97.94 |       96 |     100 |   97.92 |
 config              |     100 |      100 |     100 |     100 |
  dockerSetup.js     |     100 |      100 |     100 |     100 |
  queueSetup.js      |     100 |      100 |     100 |     100 |
 controllers         |    96.3 |      100 |     100 |    96.3 |
  submission.js      |    96.3 |      100 |     100 |    96.3 | 60
 services            |   97.62 |    88.89 |     100 |   97.62 |
  ExecutorService.js |   97.62 |    88.89 |     100 |   97.62 | 32
 utils               |     100 |      100 |     100 |     100 |
  escape.js          |     100 |      100 |     100 |     100 |
  extractMemory.js   |     100 |      100 |     100 |     100 |
  runTime.js         |     100 |      100 |     100 |     100 |
---------------------|---------|----------|---------|---------|-------------------

Test Suites: 5 passed, 5 total
Tests:       21 passed, 21 total
Snapshots:   0 total
Time:        1.329 s
```

Frontend

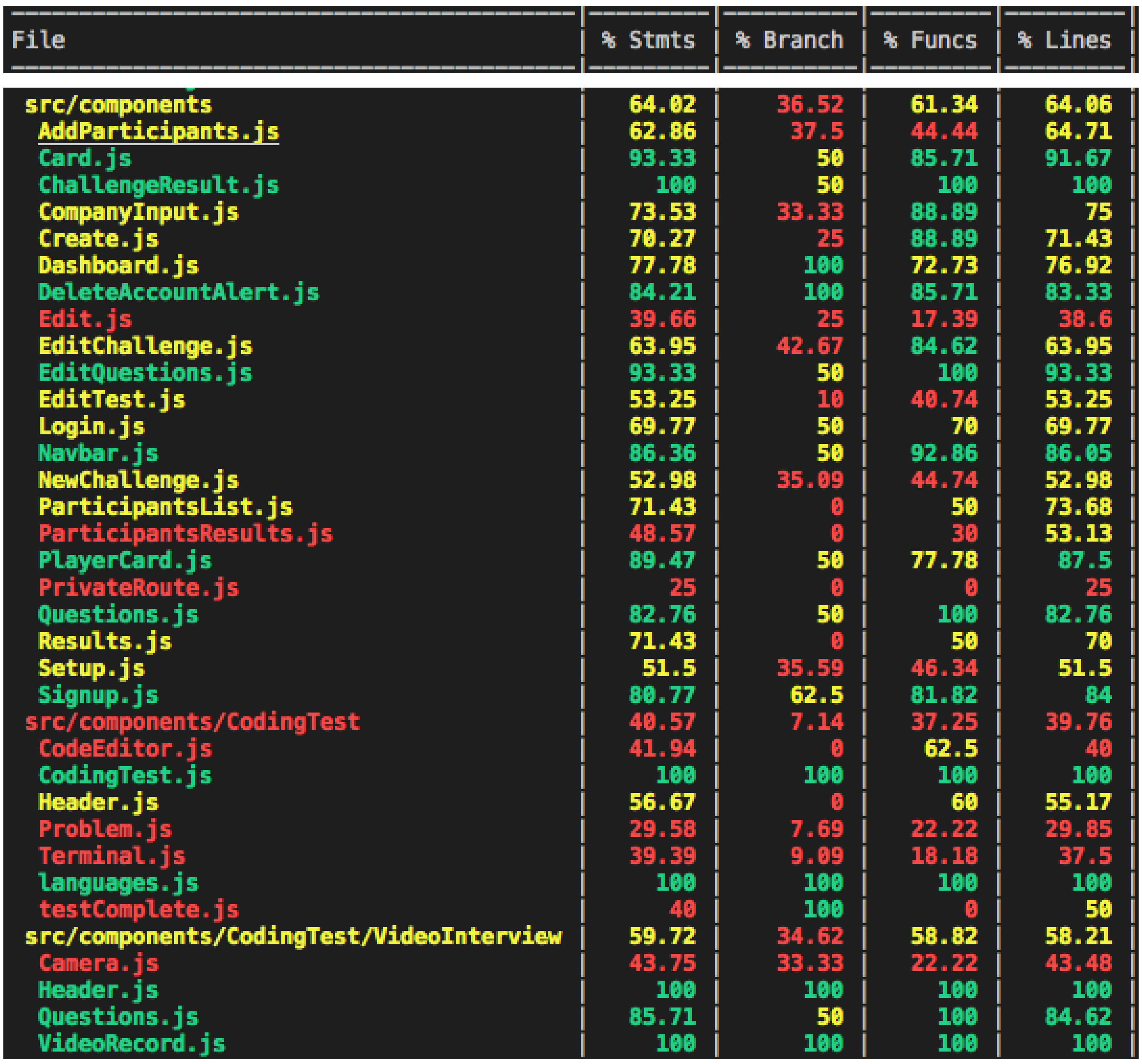
For the frontend we have 24 test suites consisting of ***391 unit tests.***


```
Test Suites: 24 passed, 24 total
Tests:       391 passed, 391 total
```

To conduct the testing for the frontend we used a combination of JEST and the react testing library. In order to test private routes that were wrapped in the firebase authentication context provider we created a utility file(test-utils.js) that re-exports everything from the react testing library. Instead of creating a new auth context this file had to still use the AuthContext from AuthContext.jst for <AuthContext.Provider>, as that's the context that the hook uses. We were then able to replace all our react testing library imports in our test files with imports from our test.utils.js file.

test-utils.js
```
import React from "react";
import { render } from "@testing-library/react";
import { BrowserRouter as Router } from "react-router-dom";
import { AuthContext } from "./contexts/AuthContext"

const AuthContex = AuthContext
const currentUser = {
 email: "abc@abc.com",
 uid: 1
};

const signup = jest.fn();
const login = jest.fn();
const logout = jest.fn();
const signInWithGoogle = jest.fn();
const AllTheProviders = ({ children }) => {
 return (
   <Router>
     <AuthContex.Provider value={{ currentUser, signup, login, logout,
signInWithGoogle }}>
       {children}
     </AuthContex.Provider>
   </Router>
 );
};
const customRender = (ui, options) => {
 return render(ui, { wrapper: AllTheProviders, ...options });
};

export * from "@testing-library/react";
export { customRender as render };
```

Below is a coverage report showing the coverage of our unit tests in the frontend.



```
----------------------------------------|---------|----------|---------|---------|
File                                    | % Stmts | % Branch | % Funcs | % Lines |
----------------------------------------|---------|----------|---------|---------|
 src/components                         |   64.02 |    36.52 |   61.34 |   64.06 |
  AddParticipants.js                    |   62.86 |     37.5 |   44.44 |   64.71 |
  Card.js                               |   93.33 |       50 |   85.71 |   91.67 |
  ChallengeResult.js                    |     100 |       50 |     100 |     100 |
  CompanyInput.js                       |   73.53 |    33.33 |   88.89 |      75 |
  Create.js                             |   70.27 |       25 |   88.89 |   71.43 |
  Dashboard.js                          |   77.78 |      100 |   72.73 |   76.92 |
  DeleteAccountAlert.js                 |   84.21 |      100 |   85.71 |   83.33 |
  Edit.js                               |   39.66 |       25 |   17.39 |    38.6 |
  EditChallenge.js                      |   63.95 |    42.67 |   84.62 |   63.95 |
  EditQuestions.js                      |   93.33 |       50 |     100 |   93.33 |
  EditTest.js                           |   53.25 |       10 |   40.74 |   53.25 |
  Login.js                              |   69.77 |       50 |      70 |   69.77 |
  Navbar.js                             |   86.36 |       50 |   92.86 |   86.05 |
  NewChallenge.js                       |   52.98 |    35.09 |   44.74 |   52.98 |
  ParticipantsList.js                   |   71.43 |        0 |      50 |   73.68 |
  ParticipantsResults.js                |   48.57 |        0 |      30 |   53.13 |
  PlayerCard.js                         |   89.47 |       50 |   77.78 |    87.5 |
  PrivateRoute.js                       |      25 |        0 |       0 |      25 |
  Questions.js                          |   82.76 |       50 |     100 |   82.76 |
  Results.js                            |   71.43 |        0 |      50 |      70 |
  Setup.js                              |    51.5 |    35.59 |   46.34 |    51.5 |
  Signup.js                             |   80.77 |     62.5 |   81.82 |      84 |
 src/components/CodingTest              |   40.57 |     7.14 |   37.25 |   39.76 |
  CodeEditor.js                         |   41.94 |        0 |    62.5 |      40 |
  CodingTest.js                         |     100 |      100 |     100 |     100 |
  Header.js                             |   56.67 |        0 |      60 |   55.17 |
  Problem.js                            |   29.58 |     7.69 |   22.22 |   29.85 |
  Terminal.js                           |   39.39 |     9.09 |   18.18 |    37.5 |
  languages.js                          |     100 |      100 |     100 |     100 |
  testComplete.js                       |      40 |      100 |       0 |      50 |
 src/components/CodingTest/VideoInterview |   59.72 |    34.62 |   58.82 |   58.21 |
  Camera.js                             |   43.75 |    33.33 |   22.22 |   43.48 |
  Header.js                             |     100 |      100 |     100 |     100 |
  Questions.js                          |   85.71 |       50 |     100 |   84.62 |
  VideoRecord.js                        |     100 |      100 |     100 |     100 |
```

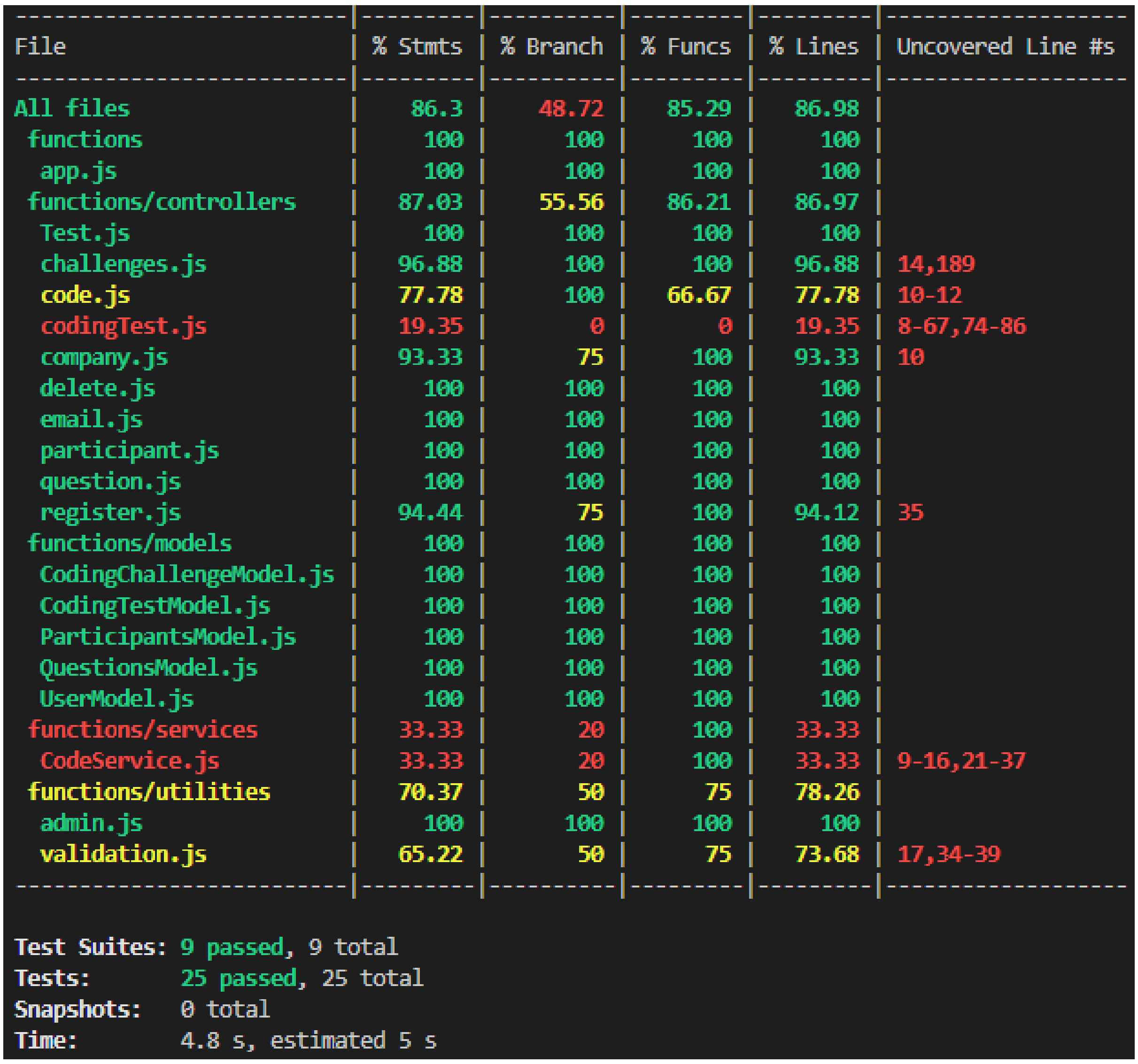
Although the coverage report shows we did not achieve an extremely high coverage percentage of the lines of code we did cover a considerable amount and it is very important to note that the coverage report refers to statements, hooks and functions and not the DOM elements in the frontend components. Testing an internal state variable very much goes against the philosophy behind react-testing-library. The react testing library is focused on the user, and what the user can see. The user has no concept of a state variable. Therefore when testing the frontend we considered how to test from the user's perspective. We thought of the changes the user would see, and test for that. How is the UI updated? What different markup or styling is displayed? All of this was extensively and thoroughly tested within our ***391 frontend tests.*** For example, for a textbox we first tested that it rendered correctly for the user and that when the value in the textbox changed, we tested that it updated correctly.

Integration Testing

The backend and remote code execution API underwent integration testing.

Backend

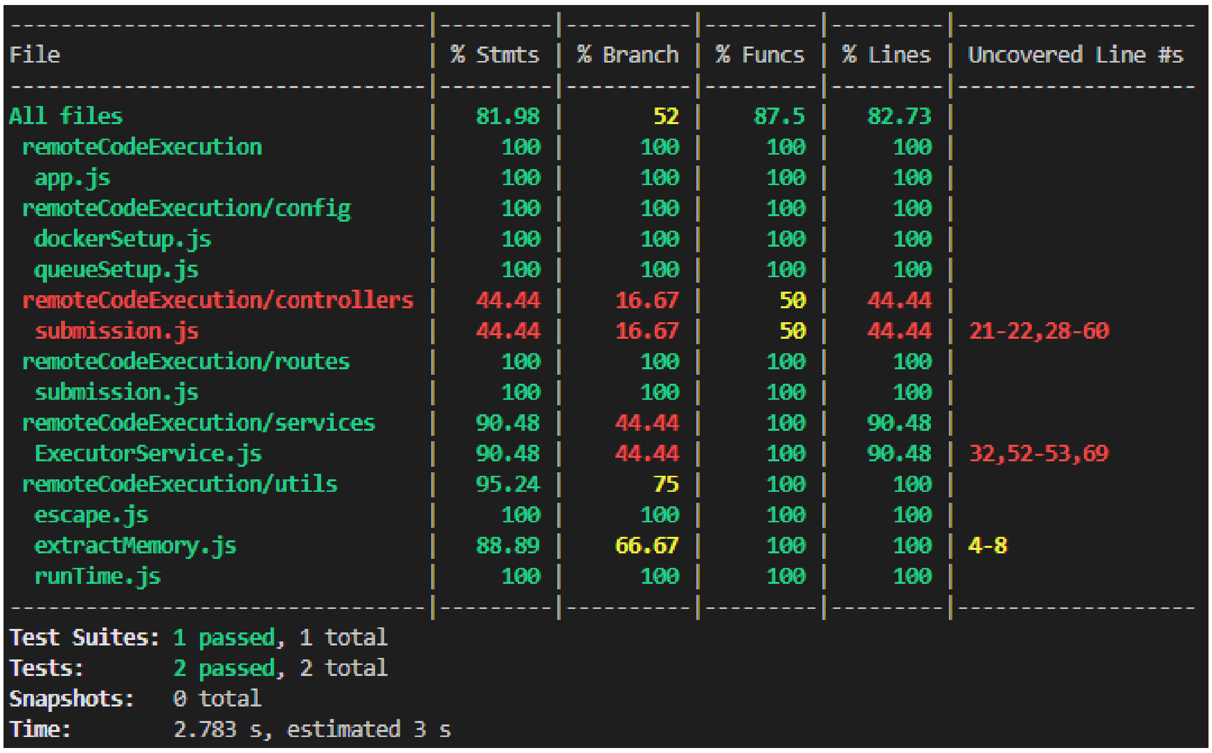
Below is a coverage report showing the coverage of our integration tests in the backend. As you can see from the report we have a total test coverage of 86.30% across all statements. We also have 9 test suites with 25 total tests, which execute in ~4.8 seconds, using the jest and supertest library with MongoDB container running to simulate writing to and from a real database.



```
--------------------------|---------|----------|---------|---------|-------------------
File                      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------------------|---------|----------|---------|---------|-------------------
All files                 |    86.3 |    48.72 |   85.29 |   86.98 |
 functions                |     100 |      100 |     100 |     100 |
  app.js                  |     100 |      100 |     100 |     100 |
 functions/controllers    |   87.03 |    55.56 |   86.21 |   86.97 |
  Test.js                 |     100 |      100 |     100 |     100 |
  challenges.js           |   96.88 |      100 |     100 |   96.88 | 14,189
  code.js                 |   77.78 |      100 |   66.67 |   77.78 | 10-12
  codingTest.js           |   19.35 |        0 |       0 |   19.35 | 8-67,74-86
  company.js              |   93.33 |       75 |     100 |   93.33 | 10
  delete.js               |     100 |      100 |     100 |     100 |
  email.js                |     100 |      100 |     100 |     100 |
  participant.js          |     100 |      100 |     100 |     100 |
  question.js             |     100 |      100 |     100 |     100 |
  register.js             |   94.44 |       75 |     100 |   94.12 | 35
 functions/models         |     100 |      100 |     100 |     100 |
  CodingChallengeModel.js |     100 |      100 |     100 |     100 |
  CodingTestModel.js      |     100 |      100 |     100 |     100 |
  ParticipantsModel.js    |     100 |      100 |     100 |     100 |
  QuestionsModel.js       |     100 |      100 |     100 |     100 |
  UserModel.js            |     100 |      100 |     100 |     100 |
 functions/services       |   33.33 |       20 |     100 |   33.33 |
  CodeService.js          |   33.33 |       20 |     100 |   33.33 | 9-16,21-37
 functions/utilities      |   70.37 |       50 |      75 |   78.26 |
  admin.js                |     100 |      100 |     100 |     100 |
  validation.js           |   65.22 |       50 |      75 |   73.68 | 17,34-39
--------------------------|---------|----------|---------|---------|-------------------

Test Suites: 9 passed, 9 total
Tests:       25 passed, 25 total
Snapshots:   0 total
Time:        4.8 s, estimated 5 s
```

Remote Code Execution API

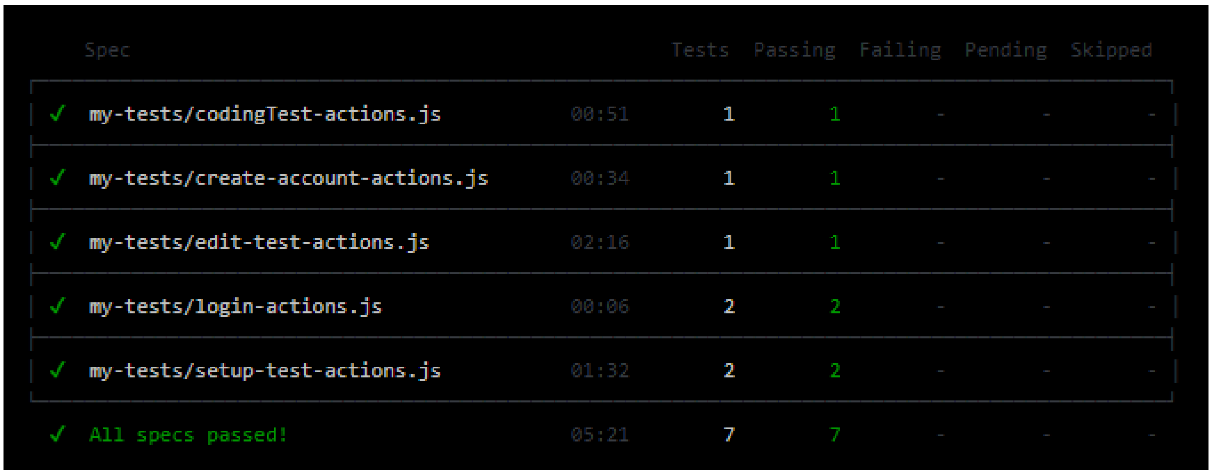
Below is a coverage report showing the coverage of our integration tests in the backend. As you can see from the report we have a total test coverage of 81.98% across all statements. We also have 1 test suite with 2 total tests, which execute in ~2.78 seconds, using the jest and supertest library.



```
----------------------------------|---------|----------|---------|---------|-------------------
File                              | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------------------|---------|----------|---------|---------|-------------------
All files                         |   81.98 |       52 |    87.5 |   82.73 |
 remoteCodeExecution              |     100 |      100 |     100 |     100 |
  app.js                          |     100 |      100 |     100 |     100 |
 remoteCodeExecution/config       |     100 |      100 |     100 |     100 |
  dockerSetup.js                  |     100 |      100 |     100 |     100 |
  queueSetup.js                   |     100 |      100 |     100 |     100 |
 remoteCodeExecution/controllers  |   44.44 |    16.67 |      50 |   44.44 |
  submission.js                   |   44.44 |    16.67 |      50 |   44.44 | 21-22,28-60
 remoteCodeExecution/routes       |     100 |      100 |     100 |     100 |
  submission.js                   |     100 |      100 |     100 |     100 |
 remoteCodeExecution/services     |   90.48 |    44.44 |     100 |   90.48 |
  ExecutorService.js              |   90.48 |    44.44 |     100 |   90.48 | 32,52-53,69
 remoteCodeExecution/utils        |   95.24 |       75 |     100 |     100 |
  escape.js                       |     100 |      100 |     100 |     100 |
  extractMemory.js                |   88.89 |    66.67 |     100 |     100 | 4-8
  runTime.js                      |     100 |      100 |     100 |     100 |
----------------------------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.783 s, estimated 3 s
```

End To End Testing

Below is a report showing the results of our end to end tests executing against the production server. As you can see from the report we have 5 test suites with 7 total tests which execute in ~5 minutes and 21 seconds. The e2e tests are run using the cypress library on the chrome browser to simulate many user flow scenarios throughout our application such as logging in, setting up a coding test, and attempting the coding test.



```
      Spec                                              Tests  Passing  Failing  Pending  Skipped
  ┌──────────────────────────────────────────────────────────────────────────────────────────────┐
  │ ✓  my-tests/codingTest-actions.js          00:51        1        1        -        -        - │
  ├──────────────────────────────────────────────────────────────────────────────────────────────┤
  │ ✓  my-tests/create-account-actions.js      00:34        1        1        -        -        - │
  ├──────────────────────────────────────────────────────────────────────────────────────────────┤
  │ ✓  my-tests/edit-test-actions.js           02:16        1        1        -        -        - │
  ├──────────────────────────────────────────────────────────────────────────────────────────────┤
  │ ✓  my-tests/login-actions.js               00:06        2        2        -        -        - │
  ├──────────────────────────────────────────────────────────────────────────────────────────────┤
  │ ✓  my-tests/setup-test-actions.js          01:32        2        2        -        -        - │
  └──────────────────────────────────────────────────────────────────────────────────────────────┘
    ✓  All specs passed!                       05:21        7        7        -        -        -
```

Performance Testing

Load Testing

Remote Code Execution API

In the below image perform a load test on /submission endpoint. We send up base64 encoded code that reads in input from command line arguments and sums the digits e.g an input of 123 would return 6.

We utilise k6 to perform the load test and run the following command:

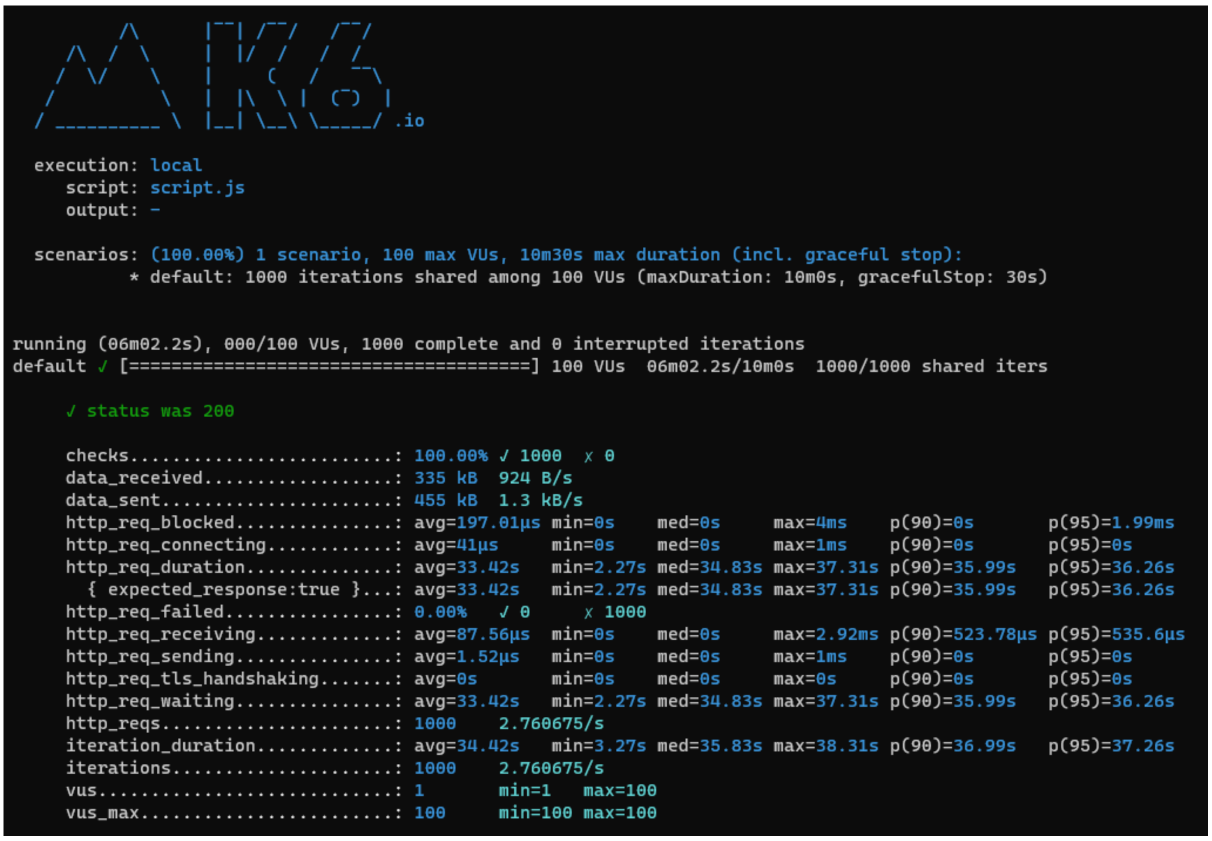
```
k6 run --vus 100 --iterations 1000 load.js
```

Here the --vus simulates a virtual user, thus '--vus 100' simulates 100 virtual users or 100 concurrent connections.

-- iterations determines the number of iterations that will be performed. Here the load.js makes 1 http post request, so the iterations are divided amongst the virtual users, meaning each user will make 10 requests, which in total is 1000.

Results

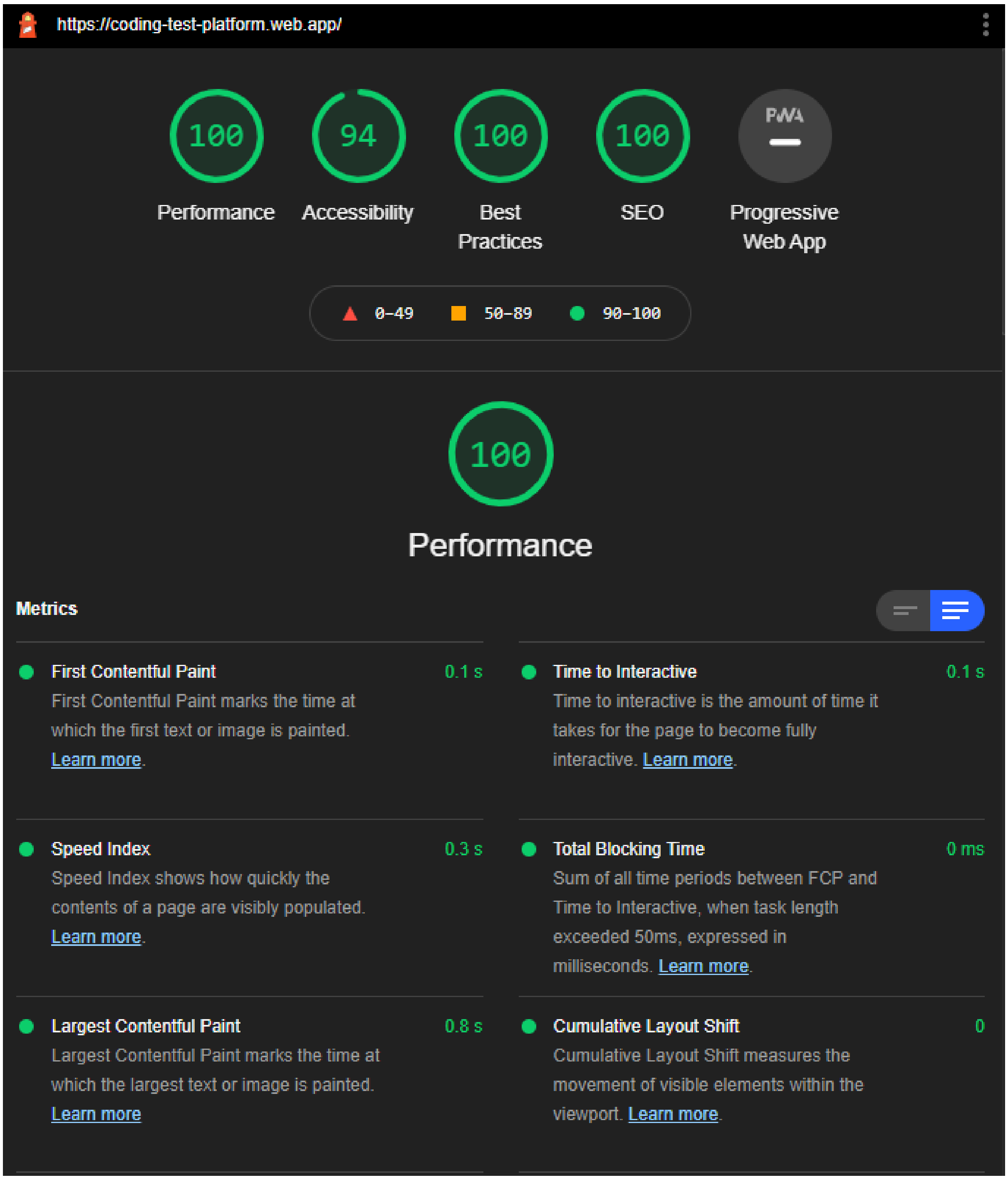
From the image below we can see that the minimum request completion time is 2.27s, with an average of 33.42s, median of 34.83s and maximum 37.31s



```
          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: script.js
     output: -

  scenarios: (100.00%) 1 scenario, 100 max VUs, 10m30s max duration (incl. graceful stop):
           * default: 1000 iterations shared among 100 VUs (maxDuration: 10m0s, gracefulStop: 30s)


running (06m02.2s), 000/100 VUs, 1000 complete and 0 interrupted iterations
default ✓ [======================================] 100 VUs  06m02.2s/10m0s  1000/1000 shared iters

     ✓ status was 200

     checks.........................: 100.00% ✓ 1000   ✗ 0
     data_received..................: 335 kB  924 B/s
     data_sent......................: 455 kB  1.3 kB/s
     http_req_blocked...............: avg=197.01µs min=0s     med=0s      max=4ms     p(90)=0s        p(95)=1.99ms
     http_req_connecting............: avg=41µs     min=0s     med=0s      max=1ms     p(90)=0s        p(95)=0s
     http_req_duration..............: avg=33.42s   min=2.27s med=34.83s max=37.31s p(90)=35.99s    p(95)=36.26s
       { expected_response:true }...: avg=33.42s   min=2.27s med=34.83s max=37.31s p(90)=35.99s    p(95)=36.26s
     http_req_failed................: 0.00%   ✓ 0      ✗ 1000
     http_req_receiving.............: avg=87.56µs  min=0s     med=0s      max=2.92ms p(90)=523.78µs p(95)=535.6µs
     http_req_sending...............: avg=1.52µs   min=0s     med=0s      max=1ms     p(90)=0s        p(95)=0s
     http_req_tls_handshaking.......: avg=0s       min=0s     med=0s      max=0s      p(90)=0s        p(95)=0s
     http_req_waiting...............: avg=33.42s   min=2.27s med=34.83s max=37.31s p(90)=35.99s    p(95)=36.26s
     http_reqs......................: 1000    2.760675/s
     iteration_duration.............: avg=34.42s   min=3.27s med=35.83s max=38.31s p(90)=36.99s    p(95)=37.26s
     iterations.....................: 1000    2.760675/s
     vus............................: 1       min=1   max=100
     vus_max........................: 100     min=100 max=100
```

Lighthouse Test

We ran a lighthouse audit for the main landing page of our web application against the production server and as you can see in the image below the audit passed with almost a perfect score in terms of performance, accessibility, best practices and SEO.



https://coding-test-platform.web.app/

Performance	Accessibility	Best Practices	SEO	Progressive Web App
100	94	100	100	PWA –

▲ 0-49 ■ 50-89 ● 90-100

100

Performance

Metrics

● First Contentful Paint	0.1 s
First Contentful Paint marks the time at which the first text or image is painted. Learn more.	
● Time to Interactive	0.1 s
Time to interactive is the amount of time it takes for the page to become fully interactive. Learn more.	
● Speed Index	0.3 s
Speed Index shows how quickly the contents of a page are visibly populated. Learn more.	
● Total Blocking Time	0 ms
Sum of all time periods between FCP and Time to Interactive, when task length exceeded 50ms, expressed in milliseconds. Learn more.	
● Largest Contentful Paint	0.8 s
Largest Contentful Paint marks the time at which the largest text or image is painted. Learn more	
● Cumulative Layout Shift	0
Cumulative Layout Shift measures the movement of visible elements within the viewport. Learn more.	

Manual Testing

API Documentation And Testing

We used Postman to document and test our API manually throughout the lifecycle of the project. API documentation can be found in the ‘docs’ folder of the repository or it is available online at https://documenter.getpostman.com/view/10756408/TzK14tzh.

Appendices

1. “React – A JavaScript Library for Building User Interfaces.” React, reactjs.org. Accessed 4 Dec. 2020.
2. MongoDB. “The Most Popular Database for Modern Apps.” MongoDB, www.mongodb.com. Accessed 4 Dec. 2020.
3. “Jest · 🃏 Delightful JavaScript Testing.” Jest, jestjs.io. Accessed 4 Dec. 2020.
4. Docker - Docker makes development efficient and predictable, https://www.docker.com. Accessed May 1 2021.
5. Cypress, https://docs.cypress.io/guides/overview/why-cypress#In-a-nutshell. Accessed May 1 2020.
6. React Testing Library, https://testing-library.com/, Accessed May 5th 2021.
7. K6, https://k6.io/, Accessed May 5th 2021.
8. Lighthouse, https://developers.google.com/web/tools/lighthouse, Accessed May 5th 2021.
9. Postman, https://www.postman.com/, Accessed May 6th 2021.