Testing Strategy Document

ID 1 and ID 2 Updates:
UI and End to End Testing

Our approach with UI testing is using a combination of Jest and the React testing library. We will particularly make sure that essential UI components (e.g. titles, descriptions, links, etc.) are in place, mainly using the *expect* API from Jest. On later deliverables where we have finished implementing all the features for our UI we would use snapshots to make sure that the visual aspect of the webpage remains as expected.

Regarding end to end testing we will use the Playwright Library. We will navigate through the pages testing the main function of them, for example being able to drop files, click buttons or links and ensuring that these work as expected. In this sense we will test the Beap Engine API, making sure that the data uploaded is transformed correctly into a .csv file and that the user can use the available ML models to get their predicted files.

Logging

This section briefly describes the motivation for logging in our project, our tools used, and recommends common best practices for developers and quality assurers.

Logging will be used in our project to address two primary concerns which are to (1) understand common (intended and unintended) use case scenarios from real users and (2) to record failures and help identify faults.

We plan to use Rollbar and Crashlytics. Rollbar is a multilevel logging framework that provides a familiar logging API for several languages and technologies including Java Spring, React, and Node.js. A different logging framework will be needed for R scripts.
Crashlytics will be used to record program crashes. This can be deployed on Digital Ocean. Crashlytic reports include logging that shows a user's path and stack trace leading up to the point of the crash.

Logging Severity Levels

Logging messages are expected to be written by both the dev team and the QA team.

Debug - These logging messages should be placed in most meaningful function and function calls. It is recommended to write debug messages with the same frequency and in similar places as assertions. The message types are usually only used in failure analysis.

Info - Record user events and API calls. For instance, “a controller of your authorization API may include an INFO log level with information on which user requested authorization if the authorization was successful or not” (Kuc, 2023).

Warning - This message type needs to be used anytime something *unexpected* occurs even if no functionality is impacted.

Error - This message type needs to be used when there is a failure that impedes one or more functionality but does not cause the program to crash.

Critical - This message type is used when there is a failure that causes the system to crash or prevents the system from running.

What work needs to be done:

Assertions and Debug messages need to be able to be written in only one line of code. This will promote logging and assertion use.

References for logging level section:

Isaiah, A. (2023, November 23). *Log levels explained and how to use them*. Better Stack Community. https://betterstack.com/community/guides/logging/log-levels-explained/

Kuc, R. (2023, November 24). *Logging levels: What they are & how to choose them*. Sematext. https://sematext.com/blog/logging-levels/

Mendes, E., & Petrillo, F. (2021, December 7). *Log severity levels matter: A multivocal mapping*. arXiv.org. https://arxiv.org/abs/2109.01192

Usage Example

source https://docs.rollbar.com/docs/javascript

```javascript
// Caught errors
try {
  doSomething();
} catch (e) {
  Rollbar.error("Something went wrong", e);
}

// Arbitrary log messages. 'critical' is most severe; 'debug' is least.
Rollbar.critical("Connection error from remote Payments API");
Rollbar.error("Some unexpected condition");
Rollbar.warning("Connection error from Twitter API");
Rollbar.info("User opened the purchase dialog");
Rollbar.debug("Purchase dialog finished rendering");

// Can include custom data with any of the above.
// It will appear as `message.extra.postId` in the Occurrences tab
Rollbar.info("Post published", {postId: 123});

// Callback functions
Rollbar.error(e, function(err, data) {
  if (err) {
    console.log("Error while reporting error to Rollbar: ", e);
  } else {
    console.log("Error successfully reported to Rollbar. UUID:", data.result.uuid);
  }
});
```

Notes on the use of Mockups:

We have created different Mockups for the pages that we are going to design, so we can be sure that their design is correct and we can ensure usability aspects. These Mockups are part of ID2.

Note on testing of R modules

Since one of the possible features asked to be implemented by the stakeholder was the adding support to upload data from Garmin watches, we may need to test the R module that would interact with this data. This would mainly consist of unit tests done manually. However, this feature would only be done if time allows to develop it.

Note on Spike Prototypes

We would invest in spike prototypes whenever we feel unsure of how the different technologies that we are using would interact between each other. Our first spike prototypes consist of a program that uses React as its frontend component and Java as its backend.

For ID2, we created spike prototypes on the use of playwright and rollbar. We integrated these spike prototypes into our main project.

Note on Smoke Test and Our different kind of testings:

Smoke Tests done for ID1 and ID 2 have basically been manual testing in which we have ensured that the pages that we created are rendered and that one can navigate through the different pages of the app. Additionally, for UI test smokes we rely on the automated tests that we have created using playwright and jest.

Regarding test smokes for functional features we will perform manual tests for example to make sure that we can upload files to the page and then download it as a .csv. In this case the first smoke test would be to ensure that the file that we uploaded gets into our database. In the same way we will test that the predicting ML algorithms actually produce different files of data.

As we have mentioned before, one of our main investments in testability is using a combination of manual and automated testing, which is currently done on the frontend(the main part where we have done changes). We will perform the same thing once we refactor the backend of our program with the new modules that we create. However, we have also invested in making our pipeline safe. Automated testing is part of the CI/CD pipeline because for every time we commit or create a pull request, all our unit and end to end tests will run. Finally, we have also invested in containerization, since we have moved our complete system to Docker.

ID3 Updates Notes:

Notes on Mocking

Based on our system, we could take advantage of mocking different scenarios. For example, we could mock calls to the database and the R repository of our system. Since calling the database takes a lot of time and we did not write the code that processes the uploaded file into a .csv file, we could mock these components and test that an already predetermined component (which already exists in the database) is correctly transformed into its csv representation and it is shown correctly to the user to be downloaded. In the same way we could mock the call to our R repository by already having the expected file that would be generated by this call.

It is worth mentioning that we are also mocking data and the logging function when we perform our unit tests for our hooks.

Smoke Test Plan for ID 4

For this deliverable our smoke test consisted on running the application, log into it, and make sure that you could navigate through the different pages.

For the next deliverable the smoke test that we will perform will consist of running the application, log into the page, uploading one file and making sure that it is in the database. Depending on development team advancement we may test the functionality of the predicting file page as well.

ID 4 Update Notes:

Testing plan for the Backend

As discussed with Dr. Osgood, we have done some testing on some relevant sections of the backend. For this deliverable we have tested the following files: Access Group Service.java , IncorrectLoginService.java, LoginService.java, MyUserDetails.java, and UserDetails.java. We have performed unit tests on these files.

We created automated unit test cases for all the methods in these files. Specifically we took advantage of mocking to fake the return calls of some functions that were indirectly called within the primarily tested functions. In the next ID we might perform integration testing with some of the backend files.

For ID 5, we will invest in research tools to perform stress and load testing into our system. We will use these tools to generate some test cases and check the performance of our project.

Additionally, we will use the built in profiler of Intellij to go through the code of our backend.

Notes on Code Coverage

Two different code coverage reports are generated for both frontend and backend. The frontend makes use of the jest testing library to generate a code coverage report for all overall files, and current files changed. The backend on the other hand makes use of jUnit and JaCoCo to generate code coverage reports for all overall files and current files changed.

The reports are automatically generated on all open pull requests where both frontend and backend coverage reports are commented. These reports are only generated once all test cases pass in the pipeline. Code coverage reports can also be manually run in the frontend by running **yarn test –coverage –watchAll**. While the backend coverage report can be run using **mvn clean test**.

Our code coverage goal is to reach a minimum of 70% overall coverage for frontend and backend code. This 70% is a metric recommended from CMPT 470 lectures

Note on test Hooks

We have a test hook that we are using in all our tests which allows us to first log in (make the user authenticated) so that the pages load correctly. For our backend code testing we created a Java class called MockFactory which contains several test hooks used to create Mock objects and Fake data. On all of the Jest Unit tests, we used a test hook called renderWithProvider to facilitate the use of Rollbar in our files. Additionally we had one member of the QA to create a playwright test that involves uploading a file and checking that it exists in the database.

Logs of tests:

Front end:

```
PASS  src/components/LoadingSpinner/LoadingSpinner.test.tsx
  LoadingSpinner
    √ T4.8 renders the HashLoader with correct props when loading is true (32 ms)
    √ T4.9 does not render the HashLoader when loading is false (19 ms)

 PASS  src/shared/hooks/useGetPredictedDataList.test.ts
  useGetPredictedDataList
    √ T4.25 should get predicted files successfully (40 ms)
    √ T4.26 should handle getPredictedFiles when it errors (8 ms)

 PASS  src/shared/hooks/useDownload.test.ts
  useDownload
    √ T4.3 should download a file from the database (69 ms)
    √ T4.4 should handle download when it errors (12 ms)

 PASS  src/App.test.tsx
  √ TID 1.4. renders all links (80 ms)

 PASS  src/pages/LogoutPage/Logout.test.tsx
  Logout
    √ T.14 should call handleLogout on render (26 ms)
    √ T4.15 should navigate to home page after logout (71 ms)
    √ T4.16 should display loading state (21 ms)
    √ T4.17 should display error state (4 ms)

 PASS  src/pages/LoginPage/LoginPage.test.tsx
  LoginPage component
 PASS  src/pages/LoginPage/LoginPage.test.tsx
  LoginPage component
    √ T4.11 renders login form with initial state (119 ms)
    √ T4.12 allows user to type in username and password fields (42 ms)
    √ T4.13 test that button submits form with username and password on click (38
ms)
    √ T4.13 test that the rotator buttons change the text on the screen (43 ms)

 PASS  src/pages/SignUpPage/SignUpPage.test.tsx
  SignUpPage component
    √ T4.18 renders sign up form with initial state (127 ms)
    √ T4.19 allows user to type in first name, last name, username and password
fields (65 ms)
    √ T4.20 test that button submits form with first name, last name, username and
password on click (63 ms)
    √ T4.21 test that form is not submitted if privacy policy is not agreed to (48 ms)
    √ T4.22 test that form info is not submitted if passwords don't match (48 ms)
    √ T4.23 test that form does not work if an element is missing (51 ms)
    √ T4.24 test that the rotator buttons change the text on the screen (47 ms)

 PASS  src/pages/LogoutPage/Logout.test.tsx
  Logout
    √ T.14 should call handleLogout on render (24 ms)
    √ T4.15 should navigate to home page after logout (66 ms)
    √ T4.16 should display loading state (22 ms)
    √ T4.17 should display error state (5 ms)

 PASS  src/App.test.tsx
  √ TID 1.4. renders all links (57 ms)

 PASS  src/pages/FileUploadPage/FileUploadPage.test.tsx
  √ TID 1.1. Renders FileUploadPage components (84 ms)

 PASS  src/shared/hooks/useDownload.test.ts
  useDownload
    √ T4.3 should download a file from the database (92 ms)
    √ T4.4 should handle download when it errors (10 ms)

 PASS  src/shared/hooks/useGetPredictedDataList.test.ts
  useGetPredictedDataList
    √ T4.25 should get predicted files successfully (34 ms)
    √ T4.26 should handle getPredictedFiles when it errors (5 ms)

 PASS  src/pages/HomePage/HomePage.test.tsx (5.058 s)
  √  TID 1.2. Render Home Page components (174 ms)

 PASS  src/pages/PredictedDataPage/PredictedDataPage.test.tsx (5.06 s)
  √ Renders PredictedDataPage components (205 ms)

 PASS  src/pages/FileUploadPage/components/FileDropzoneControls.test.tsx (5.152
s)
  √ Renders FileDropzoneControls components (160 ms)
  √  TID 1.7. Should be able to click Fitbit or Apple Watch radio (276 ms)

 PASS  src/pages/FileUploadPage/components/FileDropzone.test.tsx (5.57 s)
  √  TID 1.6. Renders FileDropzone components (77 ms)
  √ invoke onDragEnter when dragenter event occurs (356 ms)

 PASS  src/components/LoadingSpinner/LoadingSpinner.test.tsx
  LoadingSpinner
    √ T4.8 renders the HashLoader with correct props when loading is true (30 ms)
    √ T4.9 does not render the HashLoader when loading is false (5 ms)

 PASS  src/shared/hooks/useGetProcessedDataList.test.ts
  useListGetProcessedFiles
    √ should get processed files successfully (39 ms)
    √ should handle getProcessedFiles when it errors (6 ms)

 PASS  src/pages/ProcessDataPage/ProcessedDataPage.test.tsx (5.867 s)
  Processed Data Page
    √ T3.16 Should properly display the processed data page (854 ms)

 PASS  src/shared/hooks/useIsUserLoggedin.test.ts
  useIsUserLoggedIn
    √ T3.8 returns false and clears storage and cookie when expiresAt is in the past
(53 ms)
    √ T3.9 returns true when user is logged in and expiresAt is in the future (3 ms)

 PASS  src/components/HelpPopup/HelpPopup.test.tsx
  √ TID 3.1. Renders HelpPopup component (301 ms)

 PASS  src/shared/hooks/usePredictFile.test.ts
  usePredictFile
    √ should handle predict successfully (77 ms)
    √ should handle predict when it errors (9 ms)

 PASS  src/shared/hooks/useGetUploadedFiles.test.ts
  useGetUploadedFiles
    √ T3.19 should get uploaded files successfully (34 ms)
    √ T3.20 should handle getUploadedFiles when it errors (66 ms)

 PASS  src/shared/hooks/useLogin.test.ts
  useLogin
    √ T3.10 should handle login successfully (67 ms)
    √ T3.10 should handle login when it errors (5 ms)

 PASS  src/components/Navbar/Navbar.test.tsx
  √ renders all links when user is authenticated (44 ms)
  √ does not render when user is not authenticated (4 ms)

 PASS  src/shared/hooks/useDeleteFile.test.ts
  useDeleteFile
    √ T3.17 should handle delete successfully (52 ms)
    √ T3.18 should handle delete when it errors (6 ms)

 PASS  src/shared/hooks/useUpload.test.ts
  useUpload
    √ should handle uploads successfully (53 ms)
    √ should handle errors (5 ms)

 PASS  src/shared/api/Api.test.tsx
  API Tests
    √ T3.2 Login: should return mapped data upon successful login (7 ms)
```

```
  √ T3.3 Login: should throw an error upon failed login (12 ms)
  √ T3.4 Logout: should logout successfully (1 ms)
  √ T3.5 Logout: should throw an error if fails (1 ms)
  √ T3.6 should sign up successfully (1 ms)
  √ T3.7 should throw an error if sign up fails (1 ms)

PASS  src/shared/hooks/useLogout.test.ts
 useLogout
  √ T3.12 should handle logout successfully (30 ms)
  √ T3.13 should handle logout when it errors (4 ms)

PASS  src/shared/hooks/useSignup.test.ts
 useSignup
  √ T3.14 should handle signup successfully (42 ms)
  √ T3.15 should handle signup when it errors (4 ms)

Test Suites: 25 passed, 25 total
Tests:       57 passed, 57 total
Snapshots:   0 total
Time:        8.682 s
```

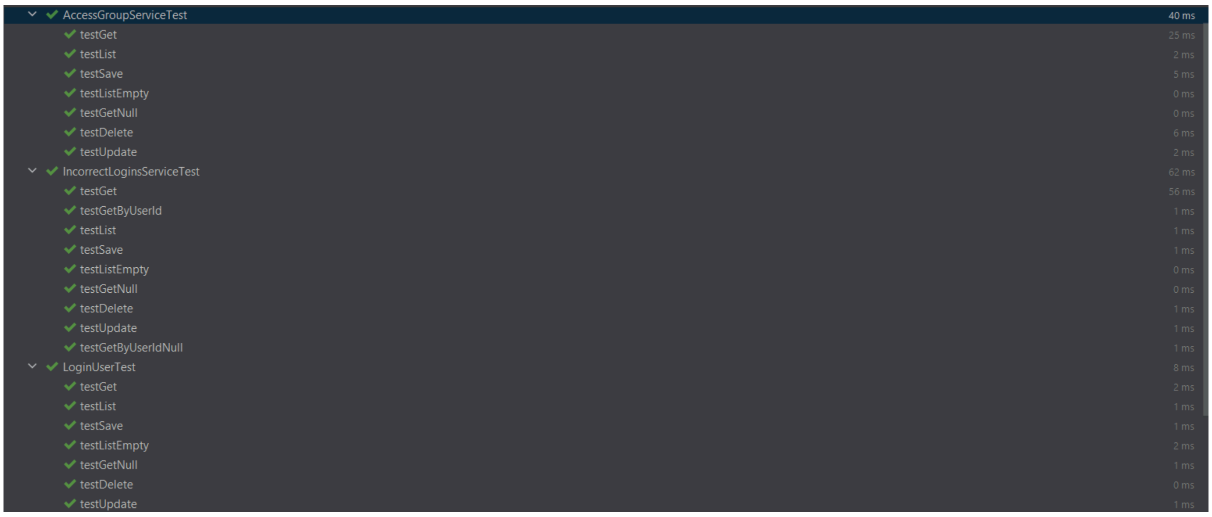
Backend:

Console output:

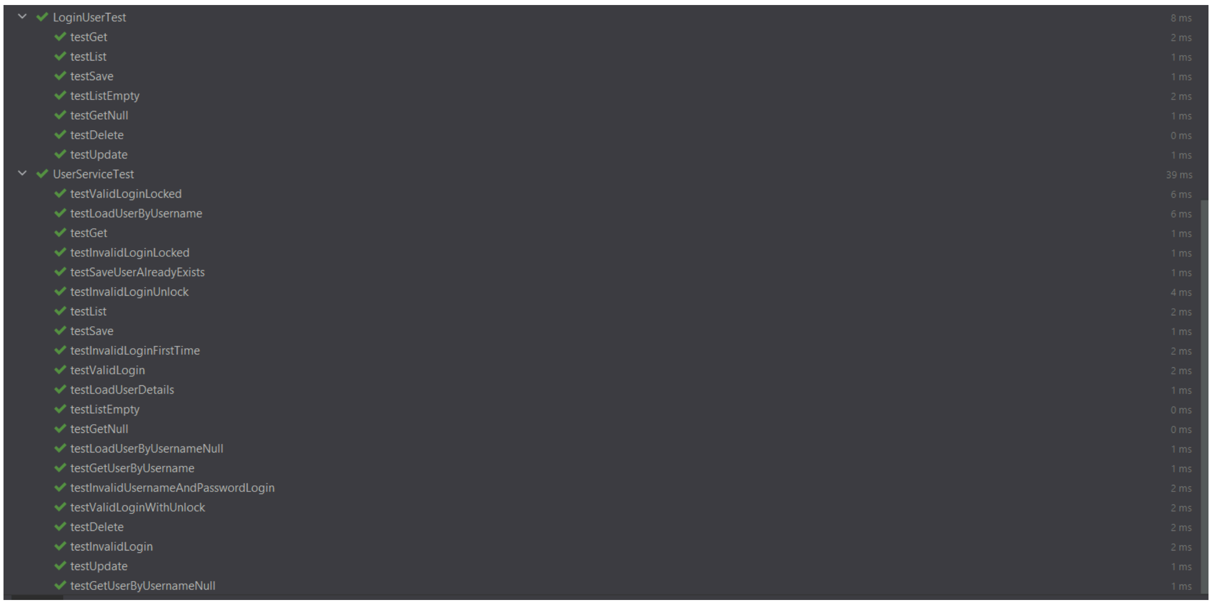
```
2024-03-16 15:58:41 INFO  AccessGroupService:90 - in AccessGroupService: get
2024-03-16 15:58:41 INFO  AccessGroupService:50 - in AccessGroupService: list
2024-03-16 15:58:41 INFO  AccessGroupService:63 - in AccessGroupService: save
2024-03-16 15:58:41 INFO  AccessGroupService:50 - in AccessGroupService: list
2024-03-16 15:58:41 INFO  AccessGroupService:90 - in AccessGroupService: get
2024-03-16 15:58:41 INFO  AccessGroupService:106 - in AccessGroupService: delete
2024-03-16 15:58:41 INFO  AccessGroupService:76 - in AccessGroupService: Update
2024-03-16 15:58:42 INFO  LoginUserService:81 - in IncorrectLoginsService: get
2024-03-16 15:58:42 INFO  LoginUserService:110 - in IncorrectLoginsService: getByUSerId
2024-03-16 15:58:42 INFO  LoginUserService:42 - in IncorrectLoginsService: list
2024-03-16 15:58:42 INFO  LoginUserService:54 - in IncorrectLoginsService: save
2024-03-16 15:58:42 INFO  LoginUserService:42 - in IncorrectLoginsService: list
2024-03-16 15:58:42 INFO  LoginUserService:81 - in IncorrectLoginsService: get
2024-03-16 15:58:42 INFO  LoginUserService:97 - in IncorrectLoginsService: delete
2024-03-16 15:58:42 INFO  LoginUserService:67 - in IncorrectLoginsService: Update
```

```
2024-03-16 15:58:42 INFO  LoginUserService:110 - in IncorrectLoginsService:
getByUSerId
2024-03-16 15:58:42 INFO  LoginUserService:92 - in LoginUserService: get
2024-03-16 15:58:42 INFO  LoginUserService:52 - in LoginUserService: list
2024-03-16 15:58:42 INFO  LoginUserService:65 - in LoginUserService: save
2024-03-16 15:58:42 INFO  LoginUserService:52 - in LoginUserService: list
2024-03-16 15:58:42 INFO  LoginUserService:92 - in LoginUserService: get
2024-03-16 15:58:42 INFO  LoginUserService:108 - in LoginUserService: delete
2024-03-16 15:58:42 INFO  LoginUserService:78 - in LoginUserService: Update
2024-03-16 15:58:42 INFO  UserService:148 - in UserService: login
2024-03-16 15:58:42 INFO  UserService:289 - in UserService:
loadUserByUsername
2024-03-16 15:58:42 INFO  UserService:117 - in UserService: get
2024-03-16 15:58:42 INFO  UserService:148 - in UserService: login
2024-03-16 15:58:42 INFO  UserService:269 - in UserService: getUserByUsername
2024-03-16 15:58:42 INFO  UserService:63 - in UserService: save
2024-03-16 15:58:42 INFO  UserService:148 - in UserService: login
2024-03-16 15:58:42 INFO  UserService:269 - in UserService: getUserByUsername
2024-03-16 15:58:42 INFO  UserService:315 - in UserService: loadUserDetails
2024-03-16 15:58:42 INFO  UserService:52 - in UserService: list
2024-03-16 15:58:42 INFO  UserService:63 - in UserService: save
2024-03-16 15:58:42 INFO  UserService:148 - in UserService: login
2024-03-16 15:58:42 INFO  UserService:269 - in UserService: getUserByUsername
2024-03-16 15:58:42 INFO  UserService:148 - in UserService: login
2024-03-16 15:58:42 INFO  UserService:315 - in UserService: loadUserDetails
2024-03-16 15:58:42 INFO  UserService:315 - in UserService: loadUserDetails
2024-03-16 15:58:42 INFO  UserService:52 - in UserService: list
2024-03-16 15:58:42 INFO  UserService:117 - in UserService: get
2024-03-16 15:58:42 INFO  UserService:289 - in UserService:
loadUserByUsername
2024-03-16 15:58:42 INFO  UserService:269 - in UserService: getUserByUsername
2024-03-16 15:58:42 INFO  UserService:148 - in UserService: login
2024-03-16 15:58:42 INFO  UserService:269 - in UserService: getUserByUsername
2024-03-16 15:58:42 INFO  UserService:148 - in UserService: login
2024-03-16 15:58:42 INFO  UserService:315 - in UserService: loadUserDetails
2024-03-16 15:58:42 INFO  UserService:132 - in UserService: delete
2024-03-16 15:58:42 INFO  UserService:148 - in UserService: login
2024-03-16 15:58:42 INFO  UserService:269 - in UserService: getUserByUsername
2024-03-16 15:58:42 INFO  UserService:104 - in UserService: Update
2024-03-16 15:58:42 INFO  UserService:269 - in UserService: getUserByUsername
```

ScreenShots with tests:

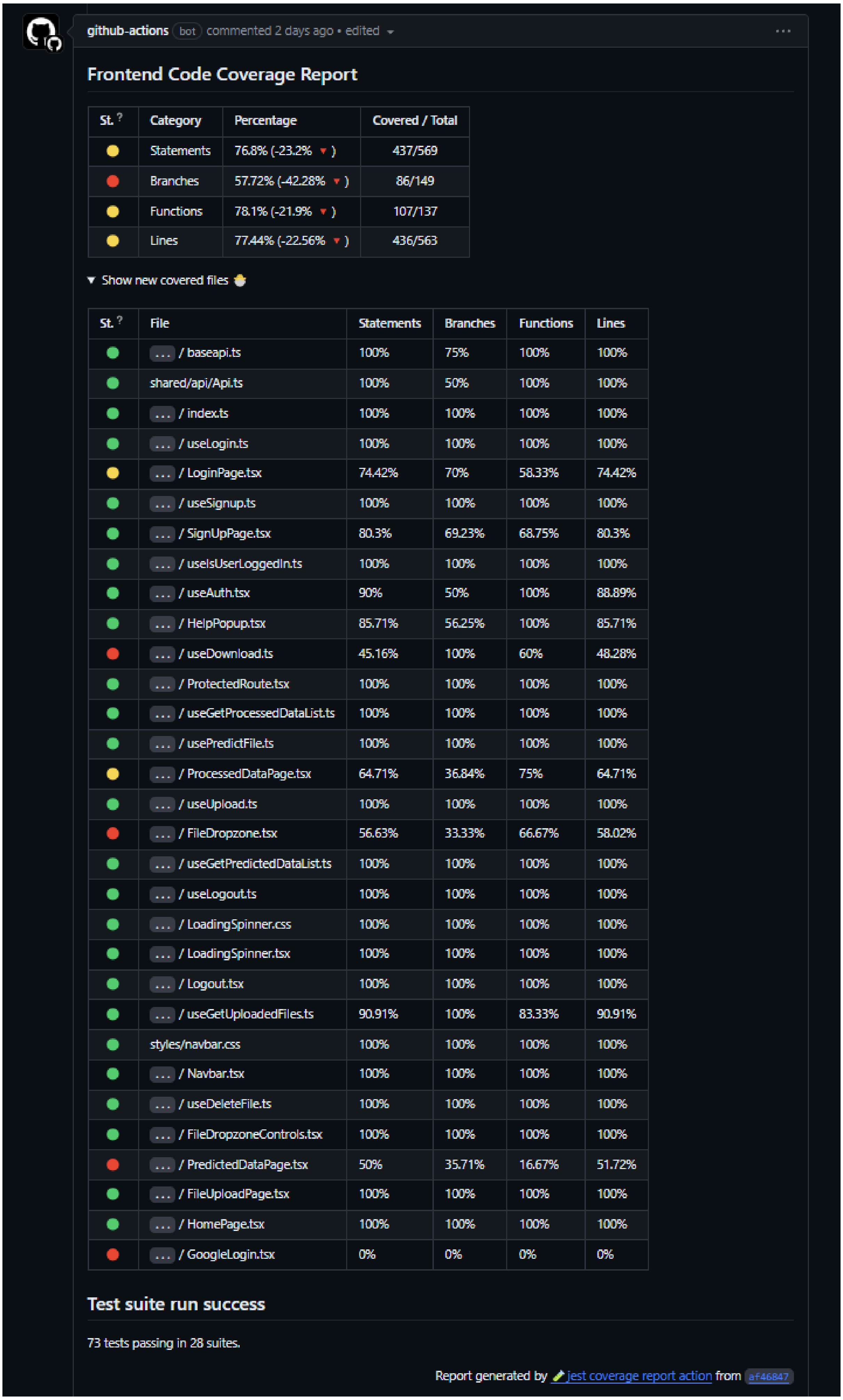


```
AccessGroupServiceTest            40 ms
    testGet                       25 ms
    testList                       2 ms
    testSave                       5 ms
    testListEmpty                  0 ms
    testGetNull                    0 ms
    testDelete                     6 ms
    testUpdate                     2 ms
IncorrectLoginsServiceTest        62 ms
    testGet                       56 ms
    testGetByUserId                1 ms
    testList                       1 ms
    testSave                       1 ms
    testListEmpty                  0 ms
    testGetNull                    0 ms
    testDelete                     1 ms
    testUpdate                     1 ms
    testGetByUserIdNull            1 ms
LoginUserTest                      8 ms
    testGet                        2 ms
    testList                       1 ms
    testSave                       1 ms
    testListEmpty                  2 ms
    testGetNull                    1 ms
    testDelete                     0 ms
    testUpdate                     1 ms
```



```
LoginUserTest                                  8 ms
    testGet                                    2 ms
    testList                                   1 ms
    testSave                                   1 ms
    testListEmpty                              2 ms
    testGetNull                                1 ms
    testDelete                                 0 ms
    testUpdate                                 1 ms
UserServiceTest                               39 ms
    testValidLoginLocked                       6 ms
    testLoadUserByUsername                     6 ms
    testGet                                    1 ms
    testInvalidLoginLocked                     1 ms
    testSaveUserAlreadyExists                  1 ms
    testInvalidLoginUnlock                     4 ms
    testList                                   2 ms
    testSave                                   1 ms
    testInvalidLoginFirstTime                  2 ms
    testValidLogin                             2 ms
    testLoadUserDetails                        1 ms
    testListEmpty                              0 ms
    testGetNull                                0 ms
    testLoadUserByUsernameNull                 1 ms
    testGetUserByUsername                      1 ms
    testInvalidUsernameAndPasswordLogin        2 ms
    testValidLoginWithUnlock                   2 ms
    testDelete                                 2 ms
    testInvalidLogin                           2 ms
    testUpdate                                 1 ms
    testGetUserByUsernameNull                  1 ms
```

Front end Code Coverage:



github-actions bot commented 2 days ago • edited

Frontend Code Coverage Report

St. ?	Category	Percentage	Covered / Total
🟡	Statements	76.8% (-23.2% 🔻)	437/569
🔴	Branches	57.72% (-42.28% 🔻)	86/149
🟡	Functions	78.1% (-21.9% 🔻)	107/137
🟡	Lines	77.44% (-22.56% 🔻)	436/563

▼ Show new covered files 🐣

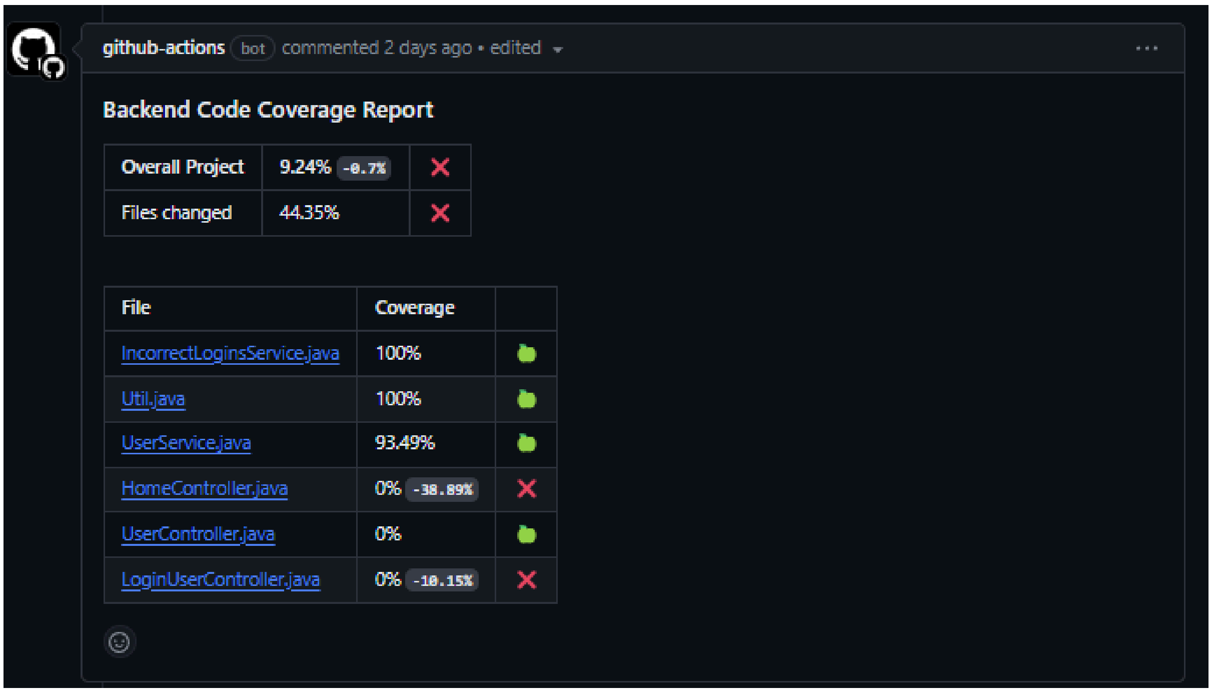
St. ?	File	Statements	Branches	Functions	Lines
🟢	... / baseapi.ts	100%	75%	100%	100%
🟢	shared/api/Api.ts	100%	50%	100%	100%
🟢	... / index.ts	100%	100%	100%	100%
🟢	... / useLogin.ts	100%	100%	100%	100%
🟡	... / LoginPage.tsx	74.42%	70%	58.33%	74.42%
🟢	... / useSignup.ts	100%	100%	100%	100%
🟢	... / SignUpPage.tsx	80.3%	69.23%	68.75%	80.3%
🟢	... / useIsUserLoggedIn.ts	100%	100%	100%	100%
🟢	... / useAuth.tsx	90%	50%	100%	88.89%
🟢	... / HelpPopup.tsx	85.71%	56.25%	100%	85.71%
🔴	... / useDownload.ts	45.16%	100%	60%	48.28%
🟢	... / ProtectedRoute.tsx	100%	100%	100%	100%
🟢	... / useGetProcessedDataList.ts	100%	100%	100%	100%
🟢	... / usePredictFile.ts	100%	100%	100%	100%
🟡	... / ProcessedDataPage.tsx	64.71%	36.84%	75%	64.71%
🟢	... / useUpload.ts	100%	100%	100%	100%
🔴	... / FileDropzone.tsx	56.63%	33.33%	66.67%	58.02%
🟢	... / useGetPredictedDataList.ts	100%	100%	100%	100%
🟢	... / useLogout.ts	100%	100%	100%	100%
🟢	... / LoadingSpinner.css	100%	100%	100%	100%
🟢	... / LoadingSpinner.tsx	100%	100%	100%	100%
🟢	... / Logout.tsx	100%	100%	100%	100%
🟢	... / useGetUploadedFiles.ts	90.91%	100%	83.33%	90.91%
🟢	styles/navbar.css	100%	100%	100%	100%
🟢	... / Navbar.tsx	100%	100%	100%	100%
🟢	... / useDeleteFile.ts	100%	100%	100%	100%
🟢	... / FileDropzoneControls.tsx	100%	100%	100%	100%
🔴	... / PredictedDataPage.tsx	50%	35.71%	16.67%	51.72%
🟢	... / FileUploadPage.tsx	100%	100%	100%	100%
🟢	... / HomePage.tsx	100%	100%	100%	100%
🔴	... / GoogleLogin.tsx	0%	0%	0%	0%

Test suite run success

73 tests passing in 28 suites.

Report generated by jest coverage report action from af46847

Backend Code coverage:



github-actions bot commented 2 days ago • edited

Backend Code Coverage Report

Overall Project	9.24% -0.7%	✗
Files changed	44.35%	✗

File	Coverage	
IncorrectLoginsService.java	100%	🍏
Util.java	100%	🍏
UserService.java	93.49%	🍏
HomeController.java	0% -38.89%	✗
UserController.java	0%	🍏
LoginUserController.java	0% -10.15%	✗