1. For our CI setup, we will automatically set up a verification of our fifth requirement. This requirement states that the passwords must never be saved, they will always be computed in live time within a 15 second window any time a sign in is required. The algorithm decrypts the encrypted salts which are stored and uses those to compute the hash. To test this requirement using our workflow, we will develop an integrated test. Using Python's Selenium library, we can use a Chromium browser to load a test page for our extension to run on. Then we will simulate a query call to our AWS server to ensure there is no storage of our computed password. Finally, the integrated test will attempt a sign-in into the test website to ensure a proper calculation of the complex password.

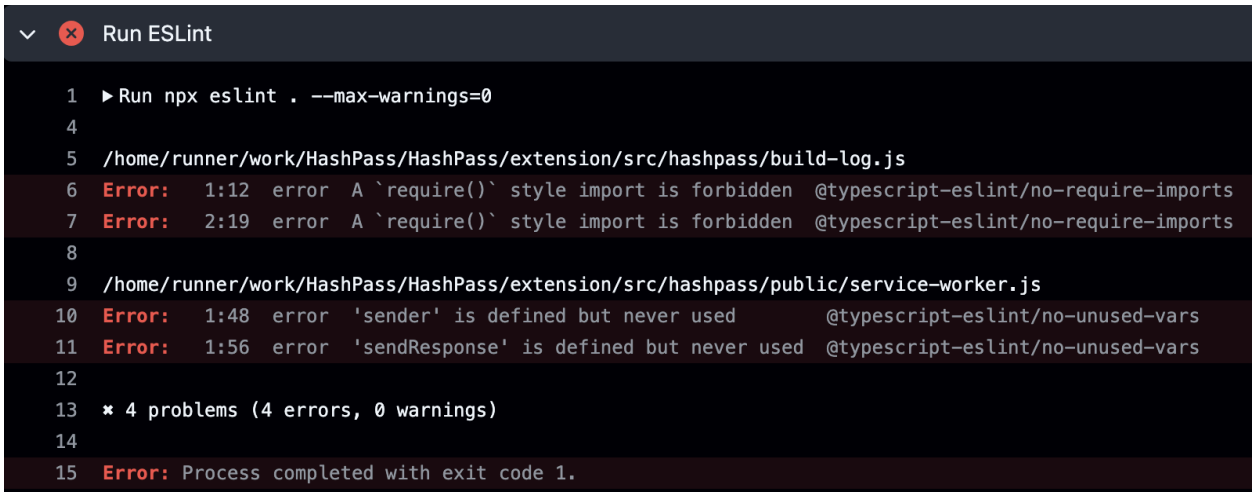
   The goal is to develop an end-to-end integrated test which performs an entire end-to-end test of our system and this specific requirement. This will allow us to identify failure points in our system and ensure that our requirement is always met. The identification of failure points will give us the ability to develop failure responses to ensure the system works even in unexpected scenarios. In addition, this end-to-end test comprises many different aspects of our system, which will help identify issues across the different systems of our product.

2. We added unit testing to our repository. In our build workflow file, we added a job that would run unit tests in our repository. Since we are doing a Typescript/React project, we decided to use the Jest testing framework. It is a framework that is easy to use and our team has some familiarity with. We configured Jest and wrote a test for our hashing API endpoint. The test ensures that the passed contents to the API call are hashed using the argon2 password hasher, which is what is specified in our plan. When we make pull requests, the workflow will run and the command that runs our unit tests will be called. This ensures that covered code will be tested upon each pull request. We plan on adding many more unit tests as our application logic grows. Below are links and screenshots showing the unit testing being called and running.

   Test suite being run by workflow:
   https://github.com/Ssharma1230/HashPass/actions/runs/13297939751/job/37133943711

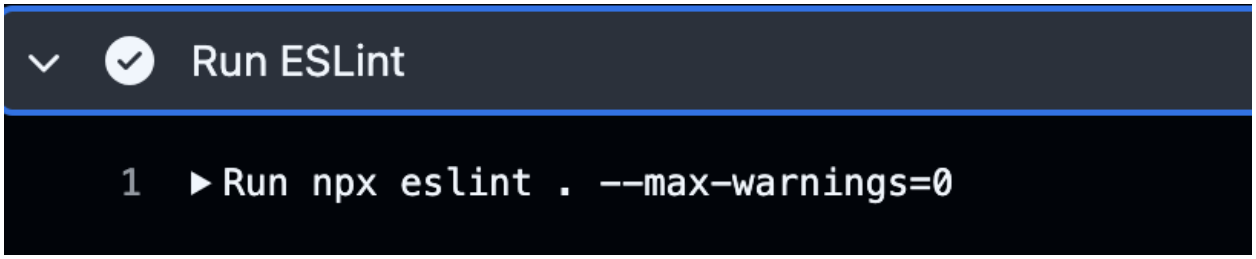
3. The automated SWQA tool that was added to enhance code quality and maintainability into our Continuous Integration pipeline was ESLint. This linter enforces coding standards, detects syntax errors, and ensures consistency across the project. GitHub Actions was configured to automatically run ESLint on every pull request, preventing code with unresolved issues from being merged. The linter executes with the command 'npx eslint . --max-warnings=0', ensuring strict quality control by failing the CI process if any errors or warnings are found. This automation streamlines development by acting as an initial code review, catching common mistakes early, and enforcing best practices. As a result, the project benefits from consistent coding style, early bug detection, and improved maintainability, reducing the need for manual intervention while keeping the codebase clean and efficient.

Failed Workflow caused by ESLint:



https://github.com/Ssharma1230/HashPass/actions/runs/13273692485/job/37058803666

Succeeded Workflow:



https://github.com/Ssharma1230/HashPass/actions/runs/13273881722/job/37059381128