

QUANTUM NEXUS

- **DEV Workflow :**

In the world of software development, a team of developers embarked on a journey to create a robust web application. We began our adventure by crafting a `super-linter.yml` file that would ensure the code's quality and consistency. This file was carefully shared among the team members.

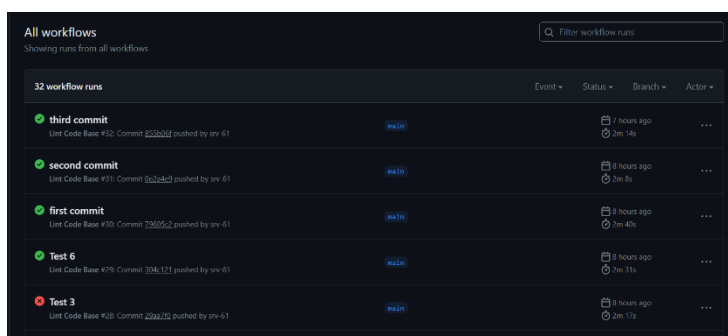
To set our project in motion, we developers created a dedicated GitHub repository within our newly formed organization. Within this repository, we diligently set up the initial structure. This involved creating a GitHub Actions workflow file named **super-linter.yml** inside the **.github/workflows** folder. This workflow file was designed to automatically scan our codebase for syntax errors and other issues.

With our development environment set up and the GitHub repository initialized, our team began the coding journey. We meticulously crafted the HTML code that would form the foundation of our web application. Each line of code was written with precision, ensuring that it met the highest standards of quality.

As the team's codebase evolved and improved, We took the next step in our journey. We created a GitHub organization to manage our development efforts more effectively. Inside this organization, we established our very first repository - the "Dev" repository.

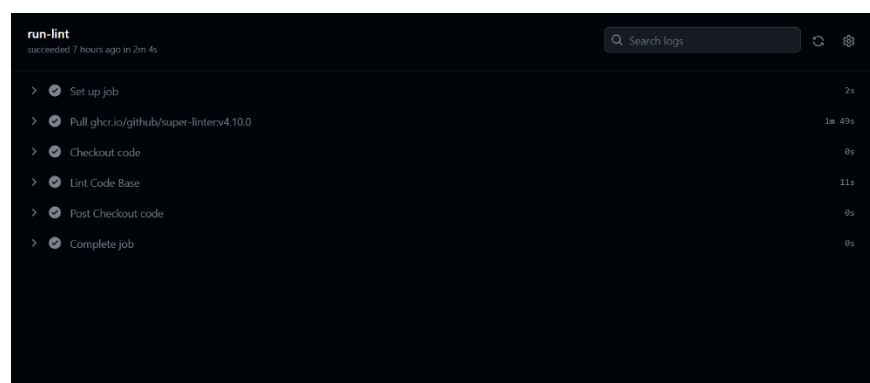
With our local codebase complete and ready for the world to see, we followed a set of commands to push our code to the "Dev" repository on GitHub. We initiated a Git repository using **git init** , added all our code with **git add .** , committed our work with a meaningful message using **git commit -m "message"** , set the default branch to "main" with **git branch -M main** , and linked our local repository to the "Dev" repository on GitHub with **git remote add origin repo-url.git** . To present our work to the world, we executed **git push -u origin main** .

Once our code is pushed, the magic of GitHub Actions kicked in. The **super-linter.yml** workflow automatically scanned our code, diligently checking for any issues. If the code passed the strict criteria for quality and consistency, a green tick mark would appear on the "Dev" repository, indicating that our code was not just functional but also of the highest quality.



The screenshot shows the 'All workflows' page in GitHub Actions. It displays a list of workflow runs for the 'super-linter' workflow. The table includes columns for 'Event', 'Status', 'Branch', and 'Actor'. The runs are listed in descending order of time, with the most recent run at the top. The status of the runs is indicated by a green checkmark for successful runs and a red cross for failed runs.

Event	Status	Branch	Actor
third commit Lint Code Base #12: Commit 25280d pushed by sv-61	Success	main	sv-61
second commit Lint Code Base #11: Commit 302d4d pushed by sv-61	Success	main	sv-61
first commit Lint Code Base #10: Commit 79605d pushed by sv-61	Success	main	sv-61
Test 6 Lint Code Base #78: Commit 308121 pushed by sv-61	Success	main	sv-61
Test 3 Lint Code Base #2: Commit 228a72 pushed by sv-61	Failure	main	sv-61



The screenshot shows the details of a specific workflow run named 'run-lint'. It indicates that the run succeeded 7 hours ago in 2m 4s. The table lists the steps of the workflow, including 'Set up job', 'Pull ghcr.io/github/super-linter:4.10.0', 'Checkout code', 'Lint Code Base', 'Post Checkout code', and 'Complete job'. The duration for each step is shown in the right column.

Step	Duration
Set up job	2s
Pull ghcr.io/github/super-linter:4.10.0	1m 43s
Checkout code	0s
Lint Code Base	11s
Post Checkout code	0s
Complete job	0s

- **TEST Workflow :**

With the "Dev" repository successfully launched, it was time to take the next step on our software development journey. The team decided to move forward with a comprehensive testing phase, and the "Test" workflow was about to unfold.

To start this leg of our journey, the team took the "Dev" repository and created a fork of it, naming it the "Test" repository. This allowed us to experiment and test changes in a controlled environment without affecting the original codebase.

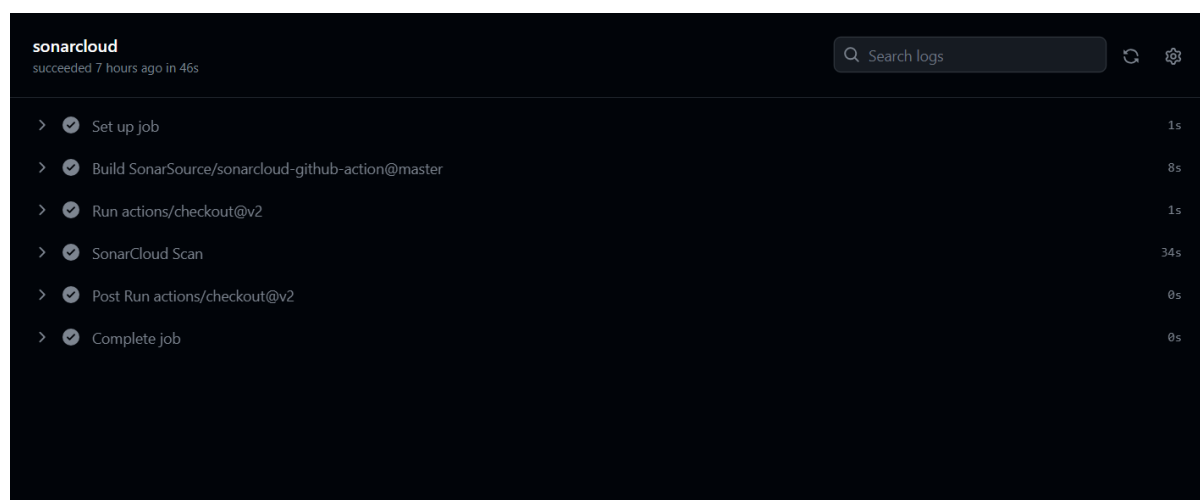
In preparation for advanced testing, the team created accounts on SonarCloud and linked them to our GitHub organization, where the "Test" repository resided. SonarCloud, a powerful code analysis tool, would play a crucial role in assessing our code quality.

Once the GitHub and SonarCloud accounts were connected, SonarCloud provided us with a special token known as the "SONAR_TOKEN." This token would grant the necessary access to SonarCloud's analysis capabilities.

To securely store the "SONAR_TOKEN," the team ventured into the "Test" repository's settings. There, we navigated to the "Secrets & Variables" section and, under "Actions," clicked the "New Repository Secret" button. We gave this secret the name "SONAR_TOKEN" and pasted the token we had received from SonarCloud, ensuring its safekeeping.

Now equipped with the power of SonarCloud, the team was ready to automate the testing process. We copied a GitHub Actions workflow file sent to the team, which was designed to integrate SonarCloud into their testing pipeline.

We moved swiftly to the "Test" repository's code section and navigated to the **.github/workflows** folder. There, we added the copied GitHub Actions code, committing the changes. With this new setup, every time they synchronized changes from the "Dev" repository to the "Test" repository, the SonarCloud action would spring to life. It would scan the code, analyzing it meticulously, and then display the results on the SonarCloud dashboard. If the tests passed and the code met the team's quality standards, the "Test" repository would proudly display a green tick mark, signaling success.



- **PROD Workflow :**

The team's journey in the realm of software development was reaching a critical point as they looked forward to deploying our work to a production environment. The "Prod" workflow was about to unfold.

To begin, the team needed a production-ready environment. We created a fork of the "Test" repository and aptly named it the "Prod" repository. This ensured that our production code would remain separate from our testing environment, reducing the risk of unexpected issues.

To deploy our code, the team turned to Netlify, a hosting and automation platform. We embarked on another adventure by logging into Netlify using our GitHub credentials and linking our GitHub account, which contained the "Prod" repository.

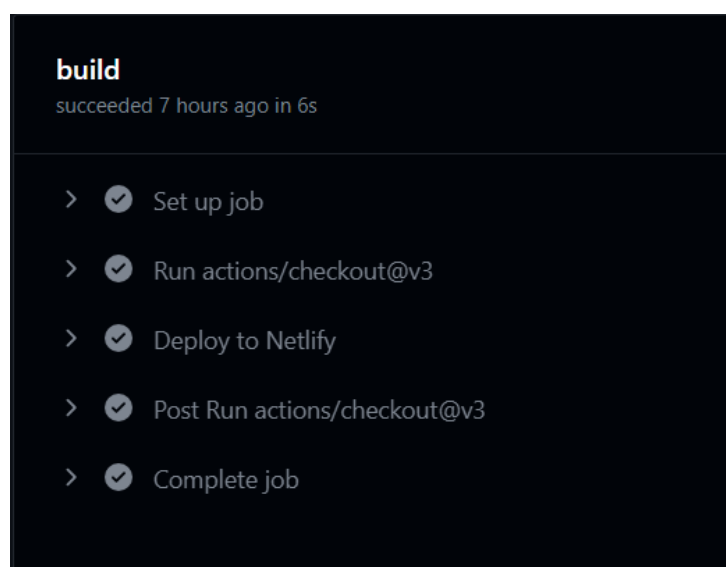
With the integration complete, the team went to Netlify's settings. There, under "Sites," we copied the unique "site id" and obtained a "personal access token" under "Access." These would be critical for securely interacting with Netlify.

To protect this sensitive information, We added these credentials as secrets in the "Prod" repository on GitHub. The secrets were named appropriately and stored securely to ensure safe and controlled access.

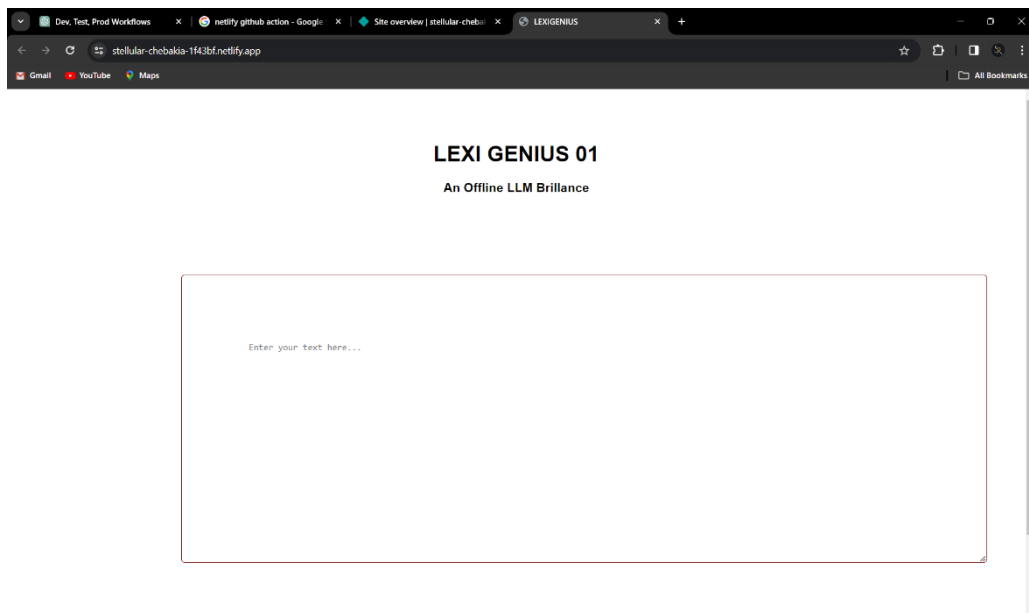
Now, the final piece of the puzzle awaited. The team received a GitHub Actions workflow file named "deploy.yml." This file held the instructions to orchestrate the deployment process on Netlify.

With determination, we headed to the "Prod" repository's code section, specifically the **.github/workflows** folder. Here, We added the copied GitHub Actions code and committed the changes.

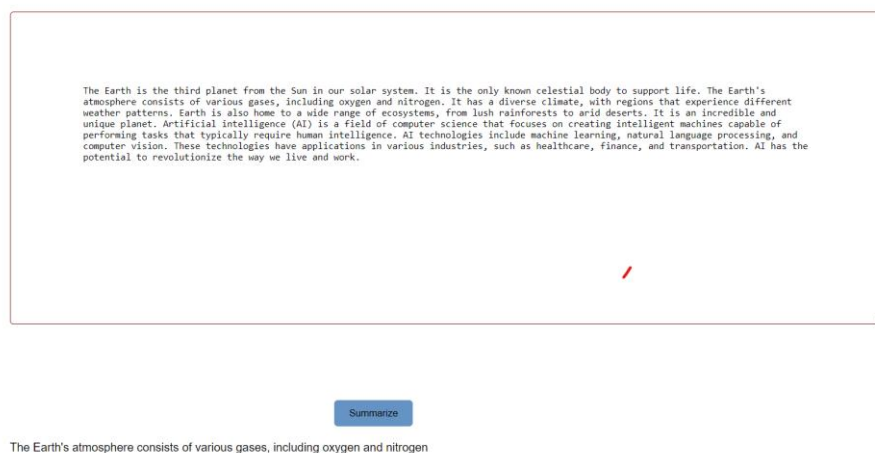
From this moment onward, every time we made updates to the "Prod" repository, the GitHub Actions workflow sprang into action. It would carefully deploy the changes, following the instructions in "deploy.yml," ensuring a smooth and automated deployment process to Netlify. This marked the beginning of our production-ready web application, ready to be accessed by users from all around the world.



OUTPUT



User Interface Done as a frontend , with summarization of the text



Story Created By :

Sohan R V (ENG21CT0037) - Product Owner
Lavanya (ENG21CT0018) - Developer
Harshitha (ENG21CT0008) - Tester
Soumya (ENG21CT0017) - Security
Kaviyarasu (ENG22CT1001) - Operations
Manikanta (ENG21CT0022) - Deployment