AUTOMATED PULL REQUEST REVIEWER SYSTEM

DevOps Internship Assessment - Technical Report

Submitted To: Metana Hirun Weerasuriya / Thinal Pradeep

Submitted By: Karindra Gimhan,

mailto:Karindragimhan49@gmail.com

Date: January 2, 2026

Project Repository: metana-pr-reviewer

Table of Contents

PART 1: Infrastructure & Connection Setup

1. Establishing the Secure Tunnel

To begin the development, I needed a way to expose my local environment to GitHub's cloud events. I used **Ngrok** to create a secure tunnel forwarding traffic to my local port 3000.

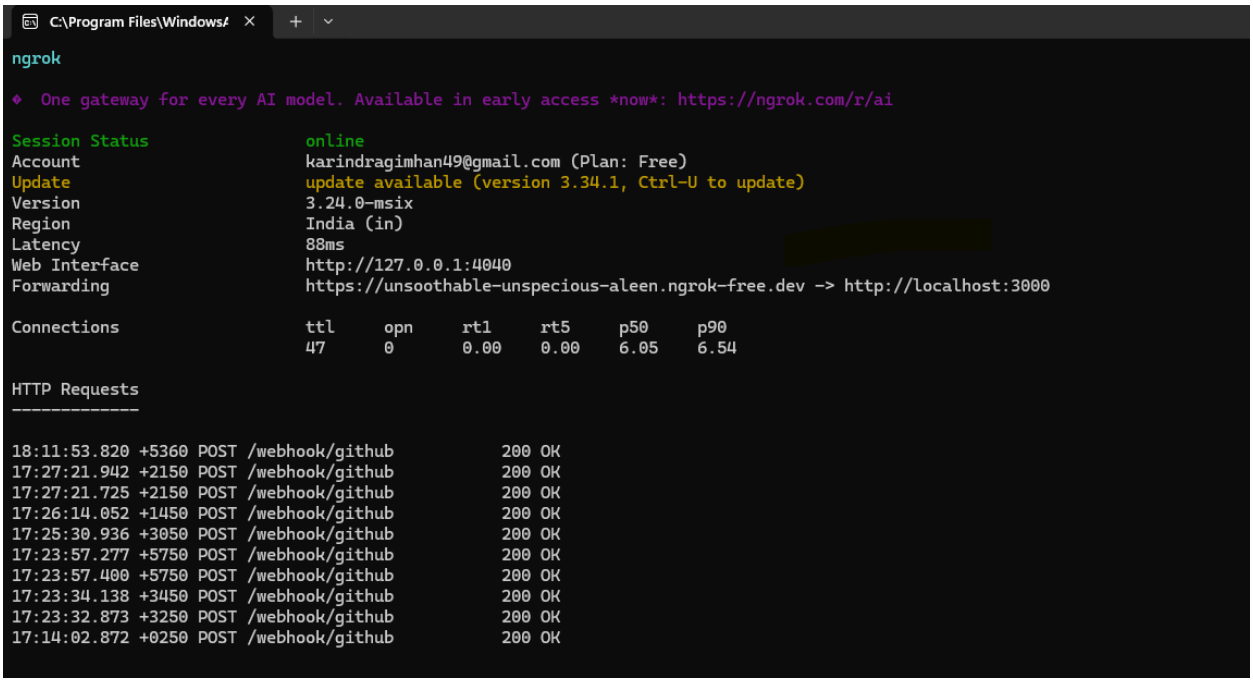

Figure 1 Ngrok terminal running successfully, forwarding requests to localhost.

2. Configuring GitHub Webhooks

Next, I configured the GitHub repository to communicate with my backend. I set up a Webhook to listen specifically for Pull Request events. I ensured security by implementing a **Webhook Secret** to verify the payload signature.

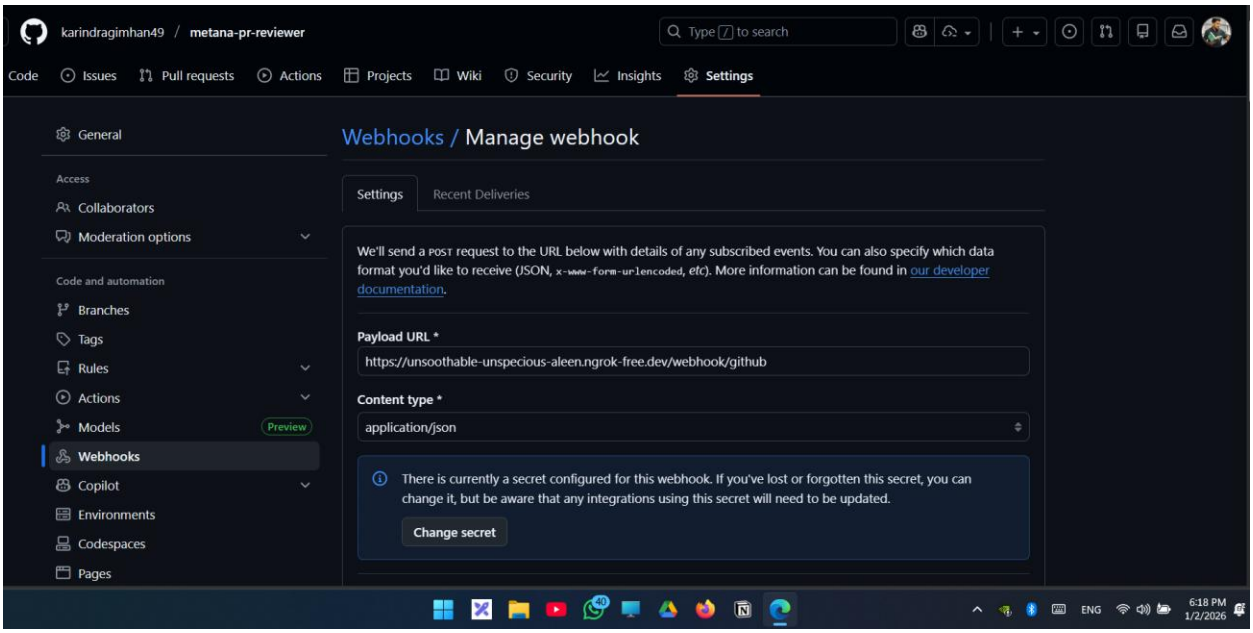

Figure 2 Setting up the Payload URL in GitHub settings.

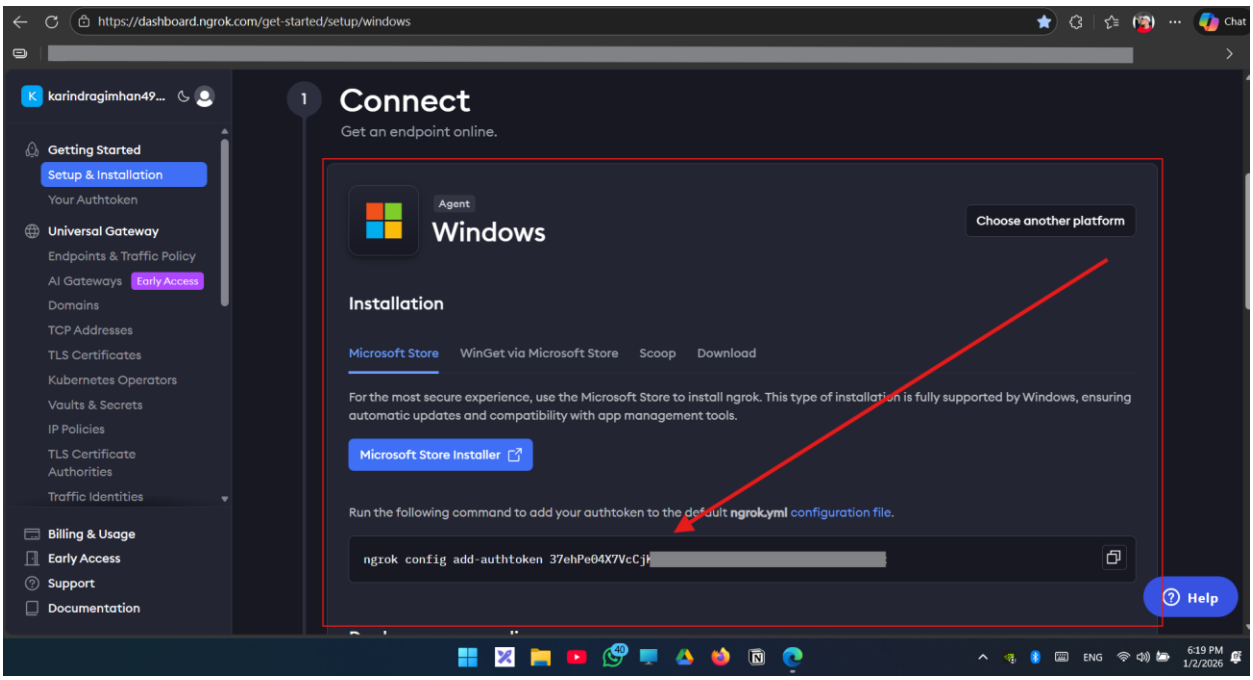

Figure 3 Configuring the content type and secret token for security.

3. Verifying Connectivity

After configuration, I verified that GitHub was successfully sending events to my backend. The green checkmarks indicate that the handshake between GitHub and my Ngrok tunnel was successful.

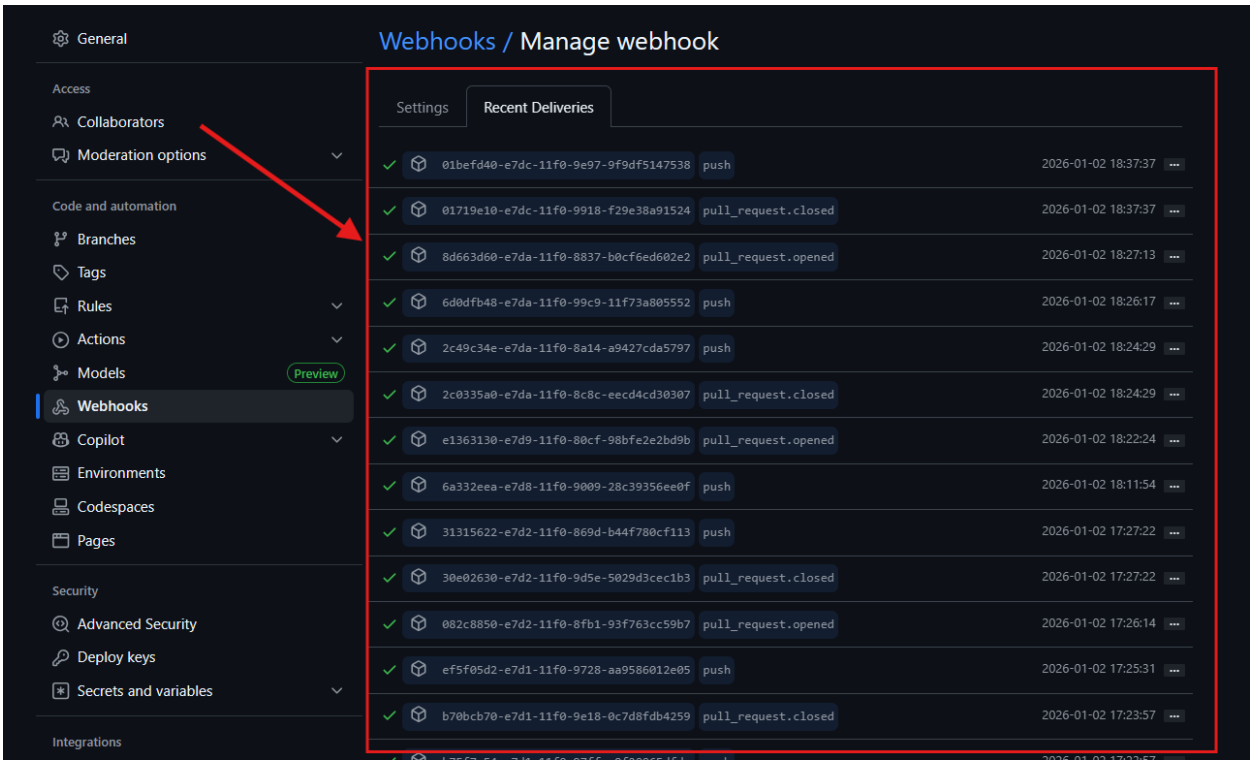

Figure 4 GitHub Webhook deliveries log showing successful (200 OK) status.

PART 2: The Review Workflow (The Core Logic)

4. Initiating a Pull Request

To test the system, I created a new branch named feature/final-test (following the naming convention logic I implemented) and pushed some code changes.

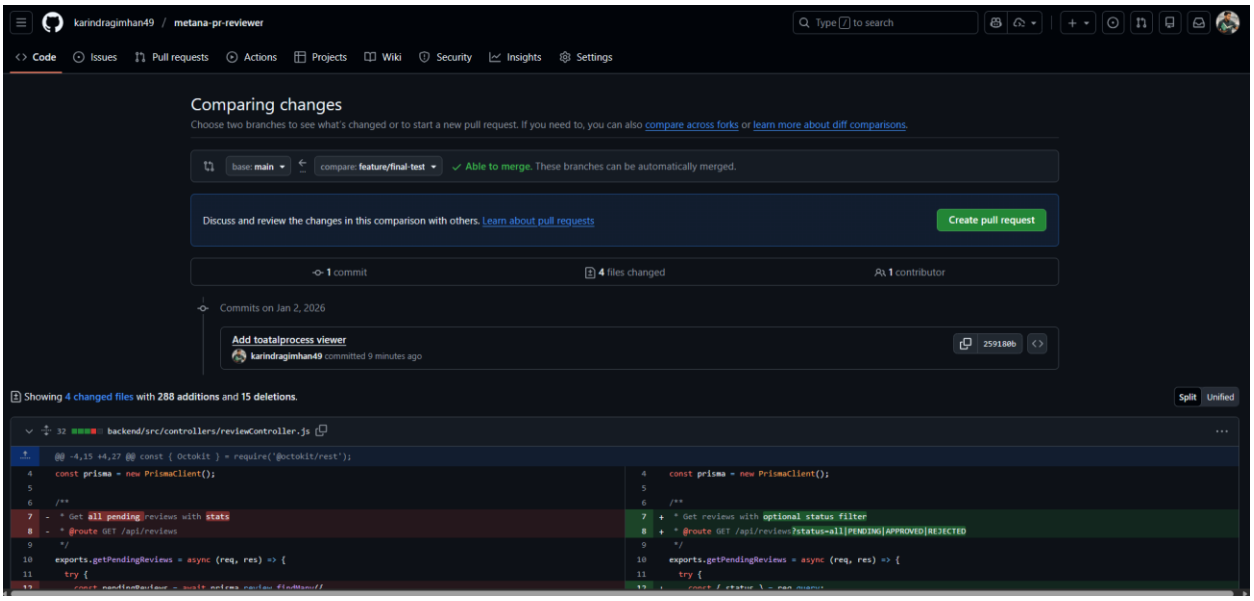

Figure 5 Checking the file differences on GitHub before creating the PR.

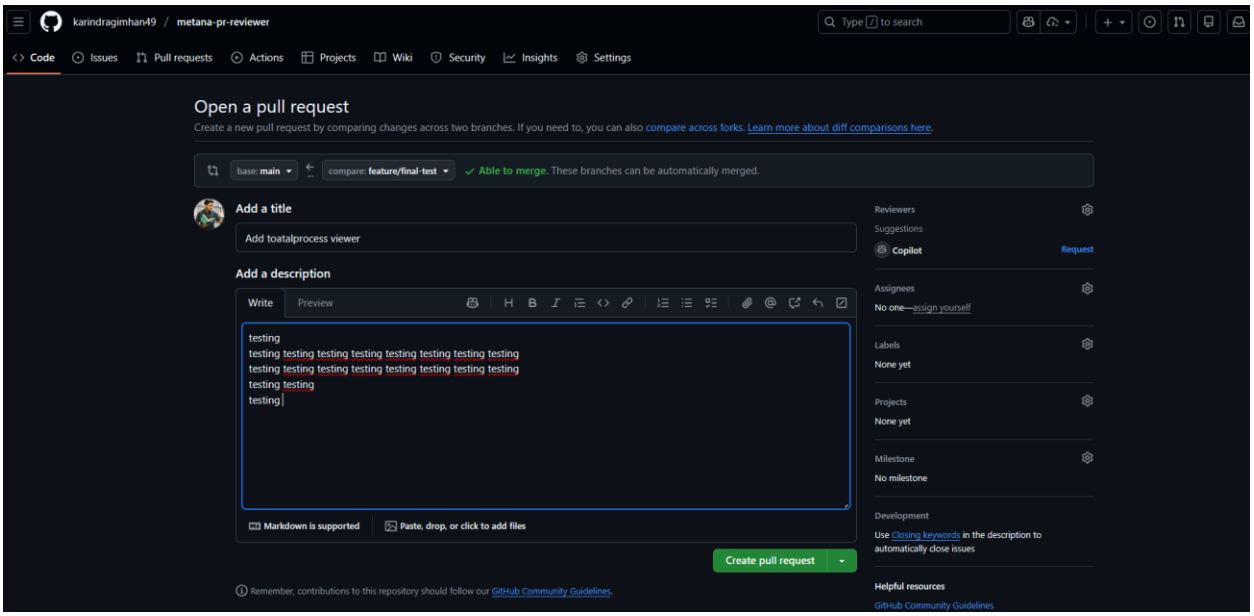

Figure 6 Opening the Pull Request with a description.

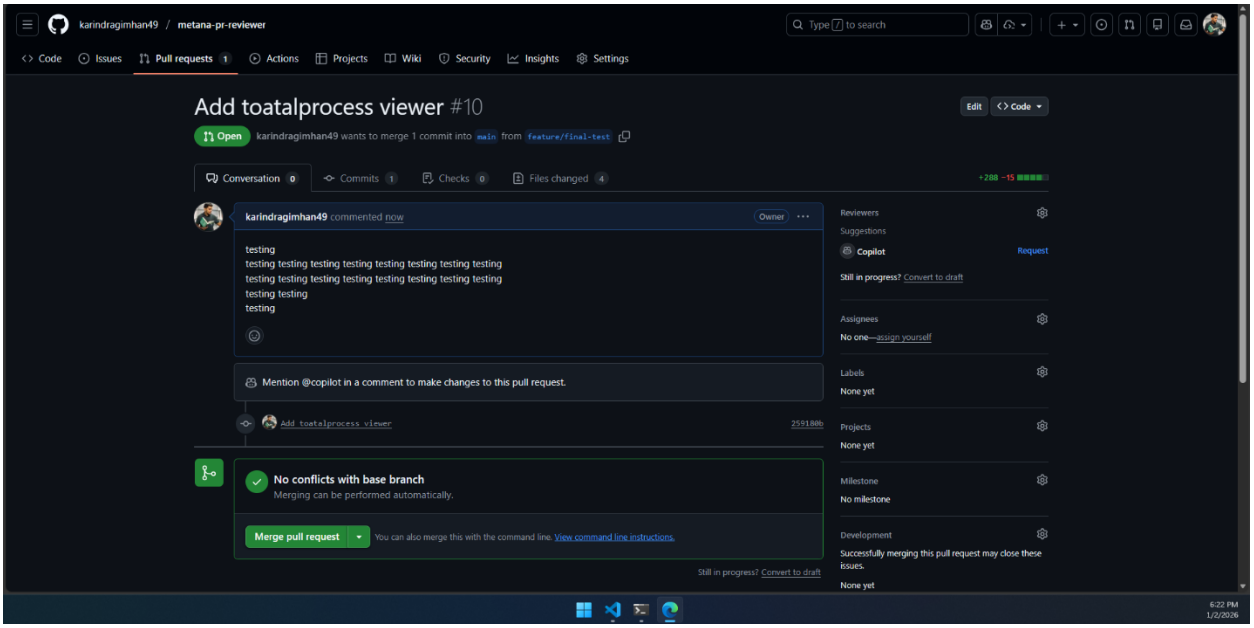

Figure 7 Pull Request successfully created on GitHub.

5. Automated Detection & AI Analysis

Immediately after the PR was created, the Webhook triggered my backend. The Instructor Dashboard instantly fetched the data. The system analyzed the branch name (feature/*) and generated specific feedback using the AI logic.

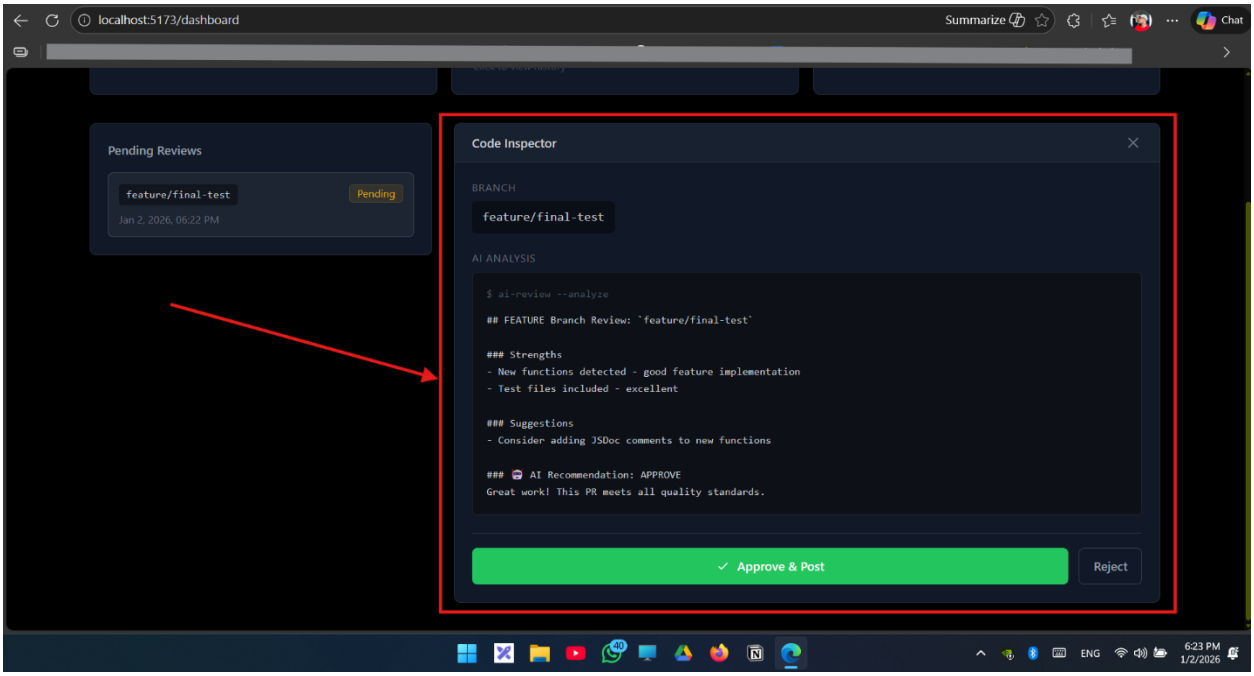

Figure 8 The Dashboard displaying the new PR with AI-generated feedback and "Recommendation: APPROVE".

6. Instructor Approval

After reviewing the AI's analysis, I proceeded to approve the PR via the dashboard. I clicked the "Approve & Post" button.

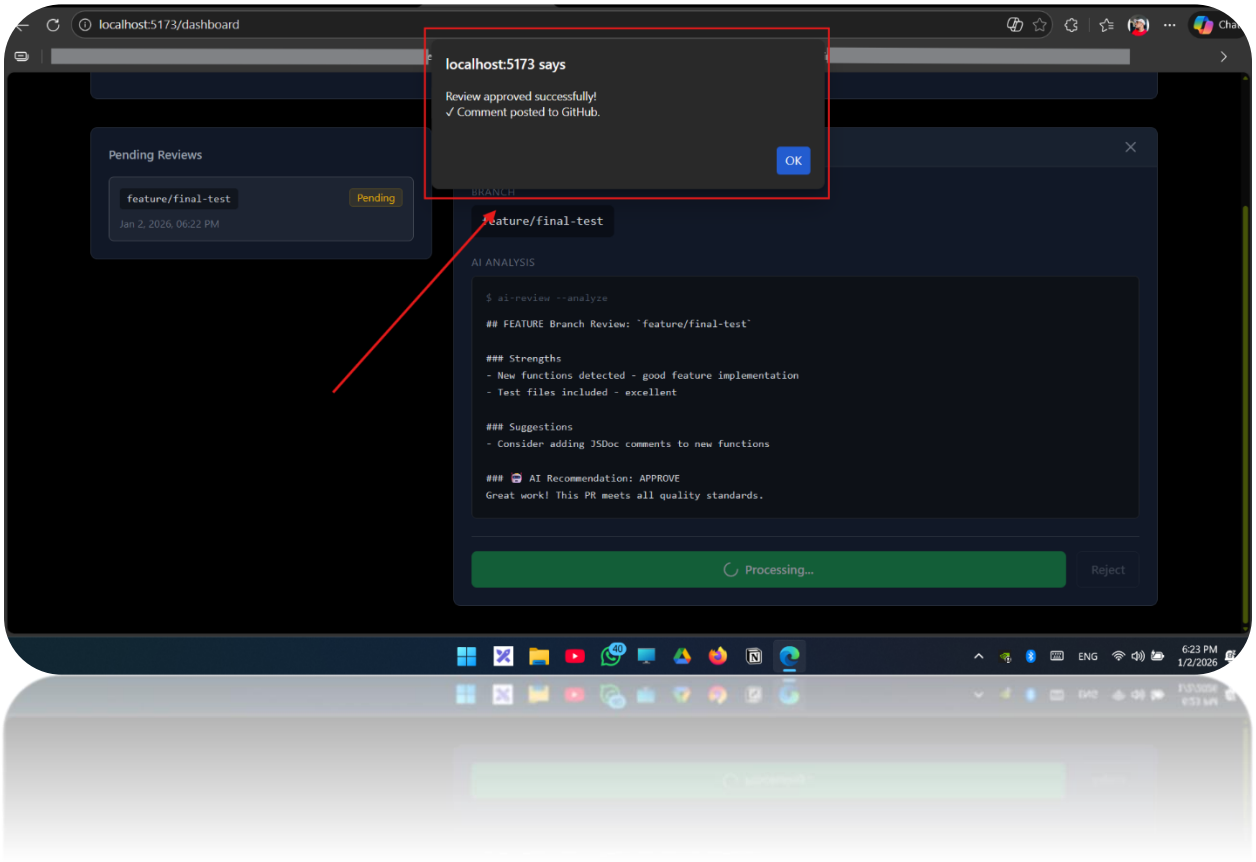

Figure 9 System processing the approval action.

7. Automated GitHub Feedback

Once approved on the dashboard, the system automatically used the GitHub API to post the formatted feedback directly onto the PR timeline.

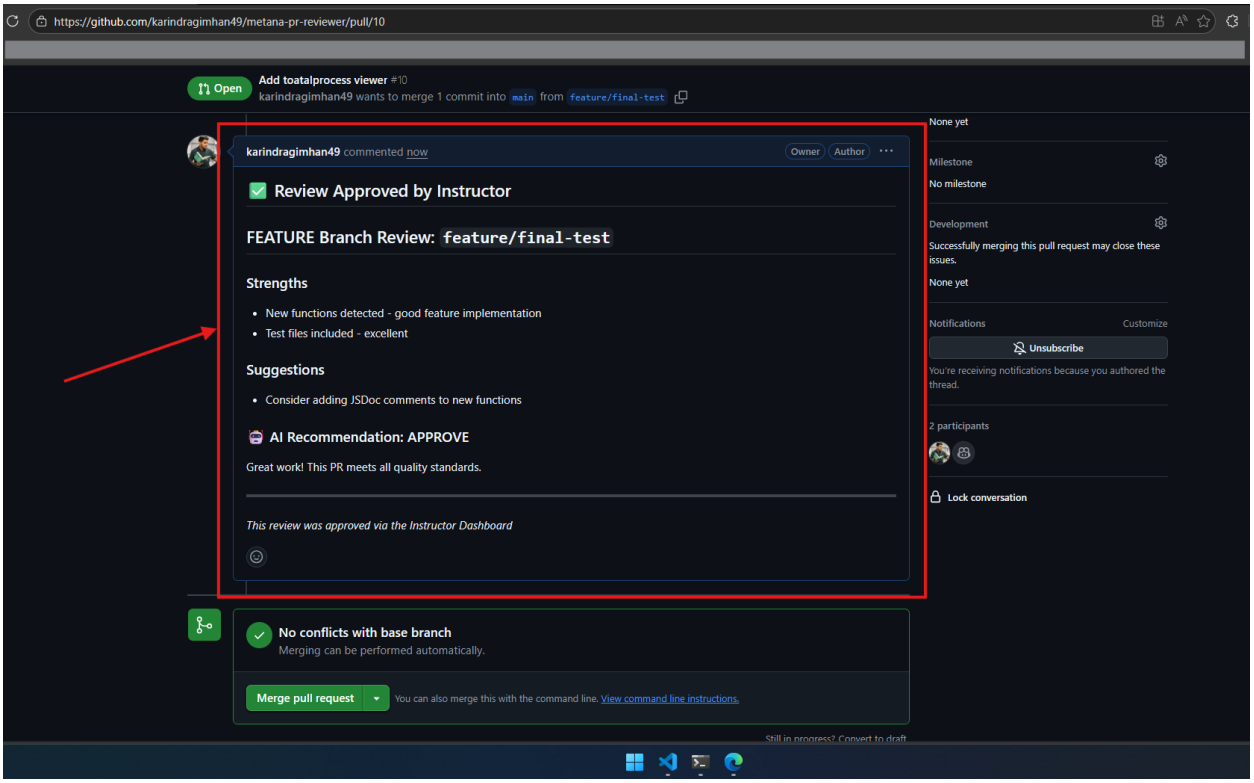

Figure 10 The final "Approved" comment posted by the bot on GitHub.

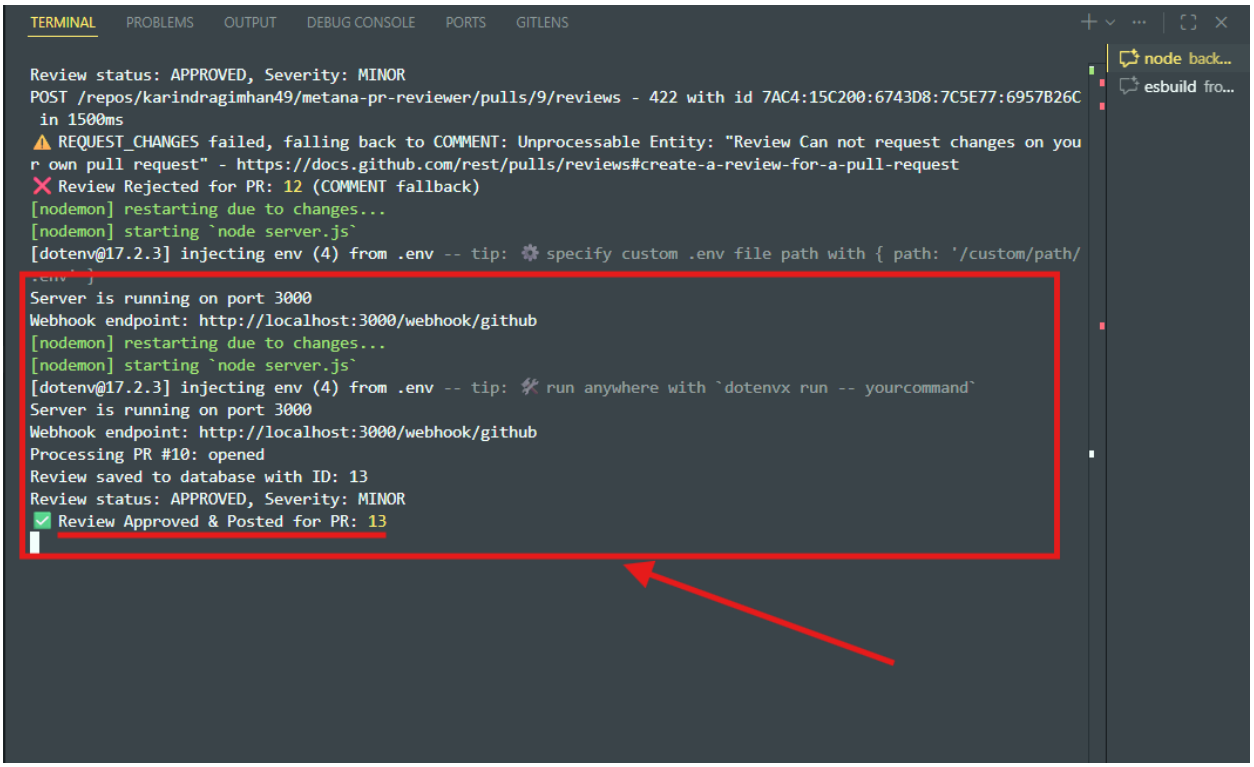

Figure 11 Backend terminal logs confirming the "Review Approved & Posted" event.

PART 3: Rejection Workflow & UI Features

This section shows how the system handles an incorrect PR, how it is rejected, and the key features of the dashboard.

8. Handling Rejections (The "Safety Valve")

A critical requirement was to allow the instructor to block bad code. I tested this by creating a generic PR and initiating the rejection process.

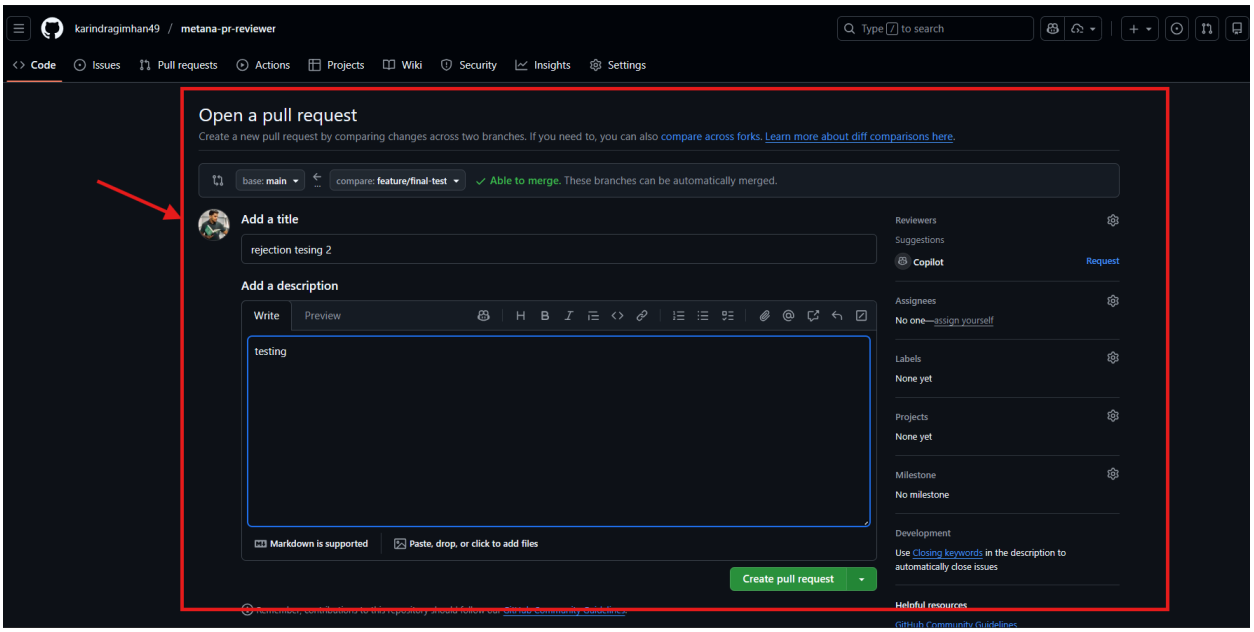

Figure 12 Creating a test PR intended for rejection.

On the dashboard, I selected the "Reject" action. This triggers the backend to attempt a "Request Changes" review on GitHub.

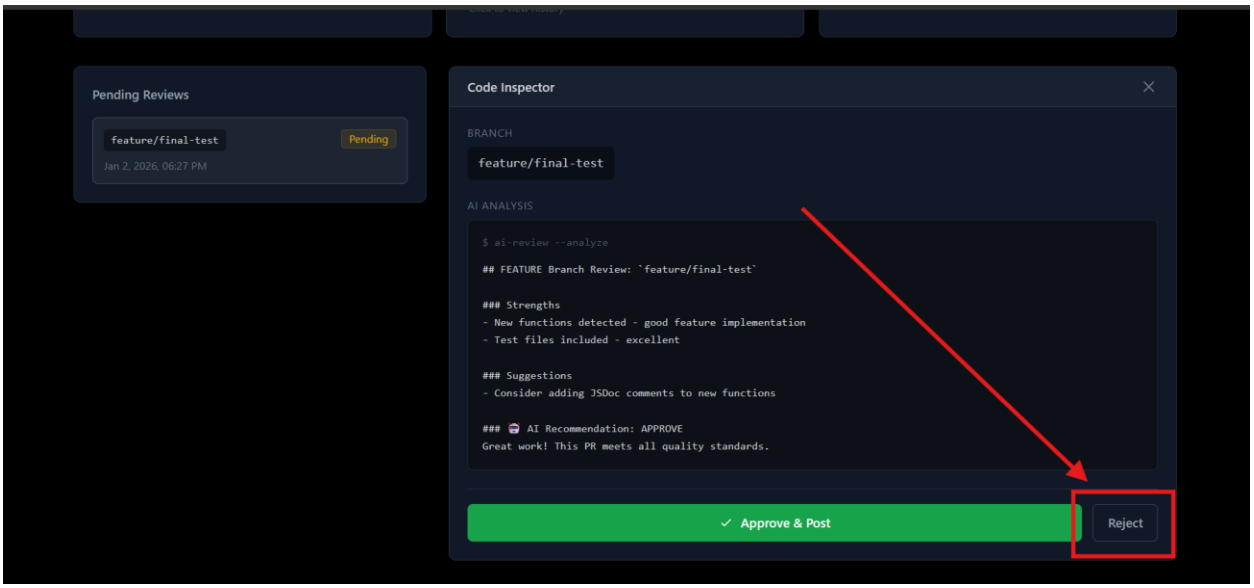

Figure 13 Instructor clicking the "Reject" button on the dashboard.

The system successfully processed the rejection and posted a "REJECTED" status back to the GitHub PR, blocking the merge (visually indicating the rejection to the student).

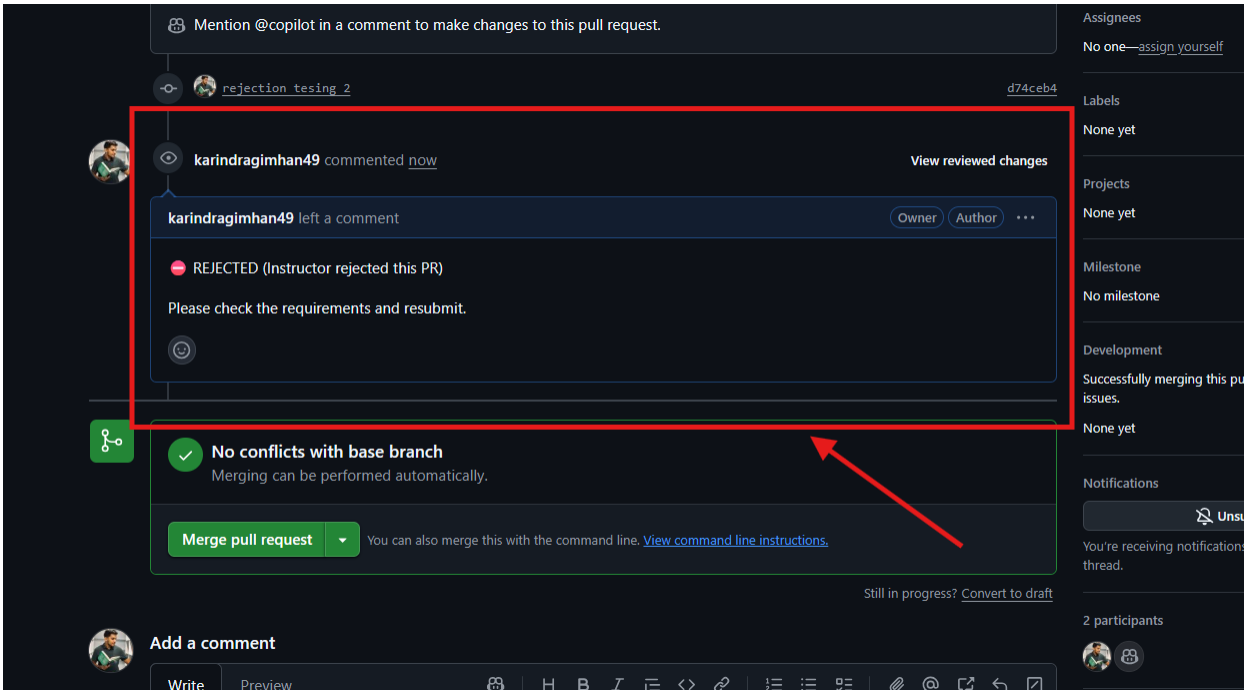

Figure 14 GitHub PR showing the "REJECTED" comment and status.

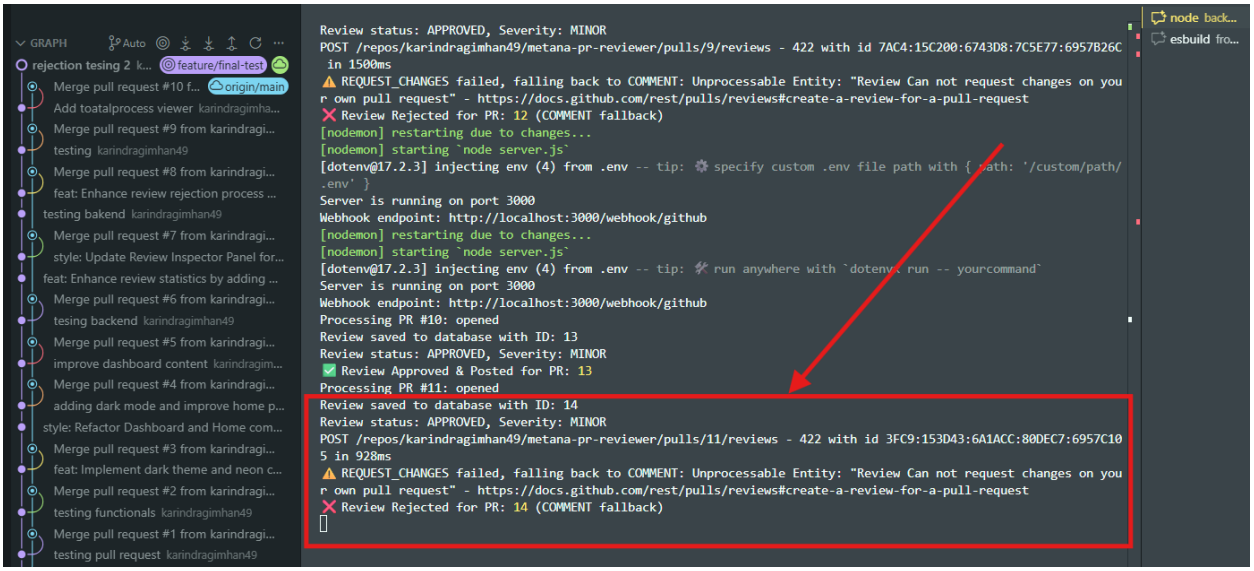

Figure 15 GitHub PR showing the "REJECTED" comment and status.

9. History & Audit Logs

To keep track of past actions, I implemented a History Log. Clicking on the "Total Processed" card opens a modal showing a list of all approved and rejected reviews.

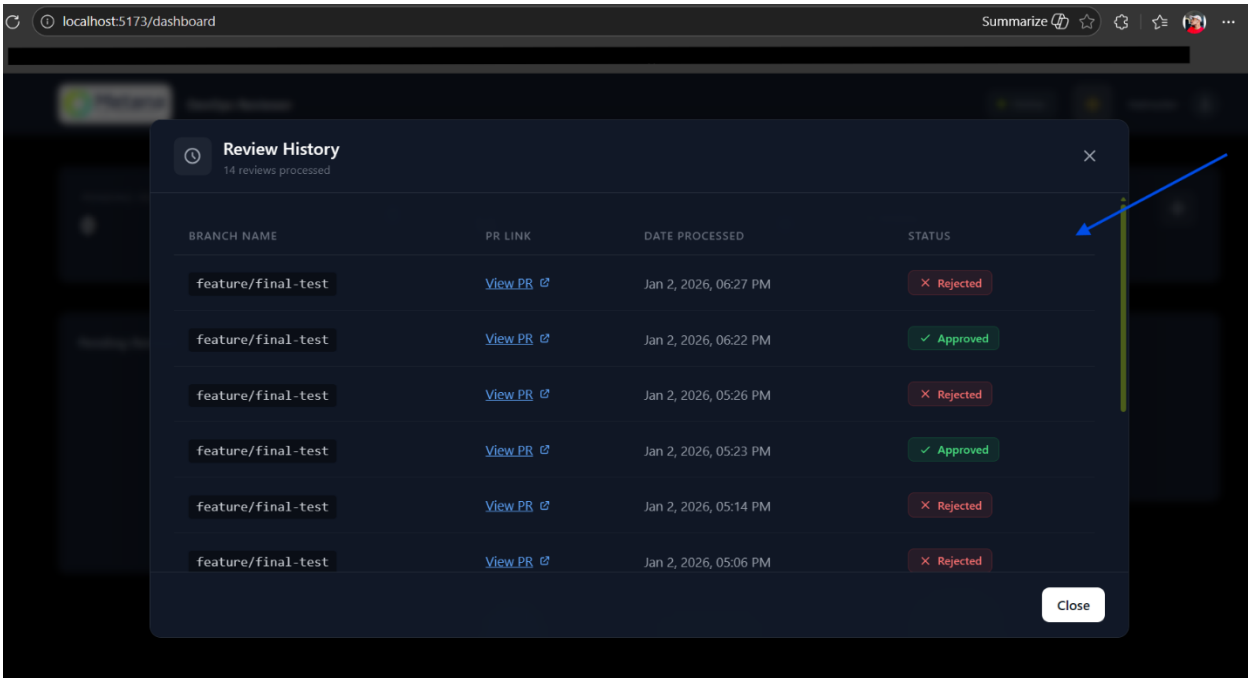

Figure 16 The History Modal displaying a log of all processed PRs.

10. Professional UI/UX (Landing Page)

Finally, to give the product a polished, enterprise-grade feel, I designed a professional Landing Page with a responsive Navbar and modern typography.

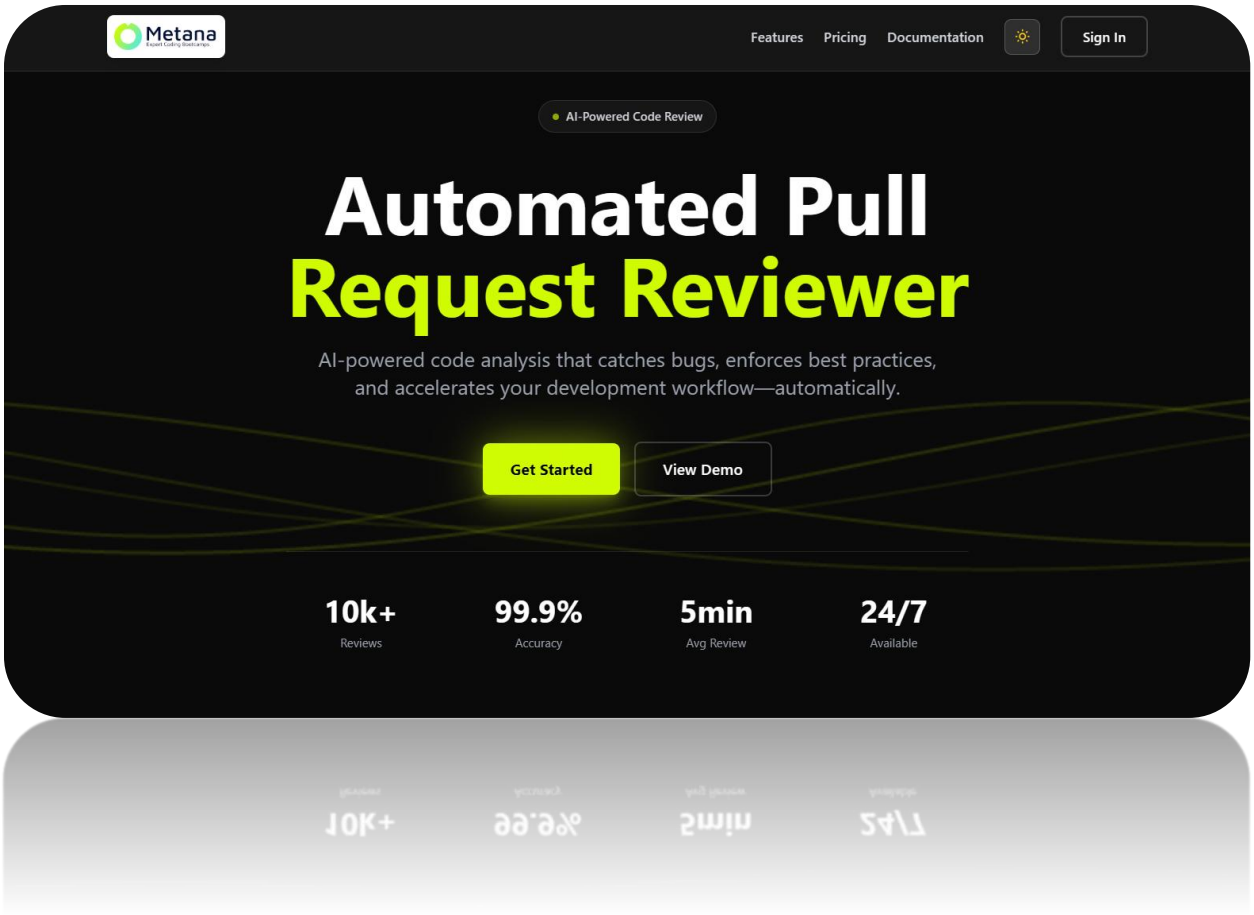

Figure 17 The Modern Landing Page (Dark Mode) serving as the entry point.

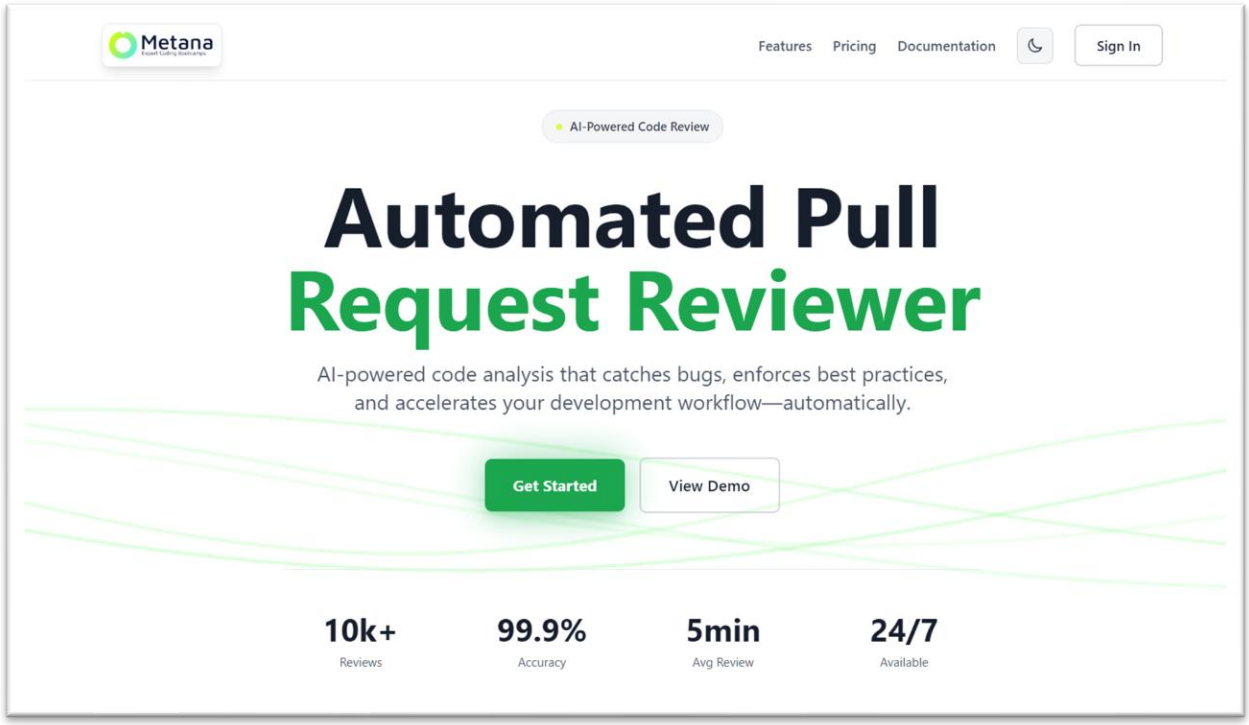

Figure 18 The Landing Page in Light Mode, demonstrating theme adaptability.

The Dashboard itself is fully responsive and maintains a clean "Code Inspector" aesthetic.

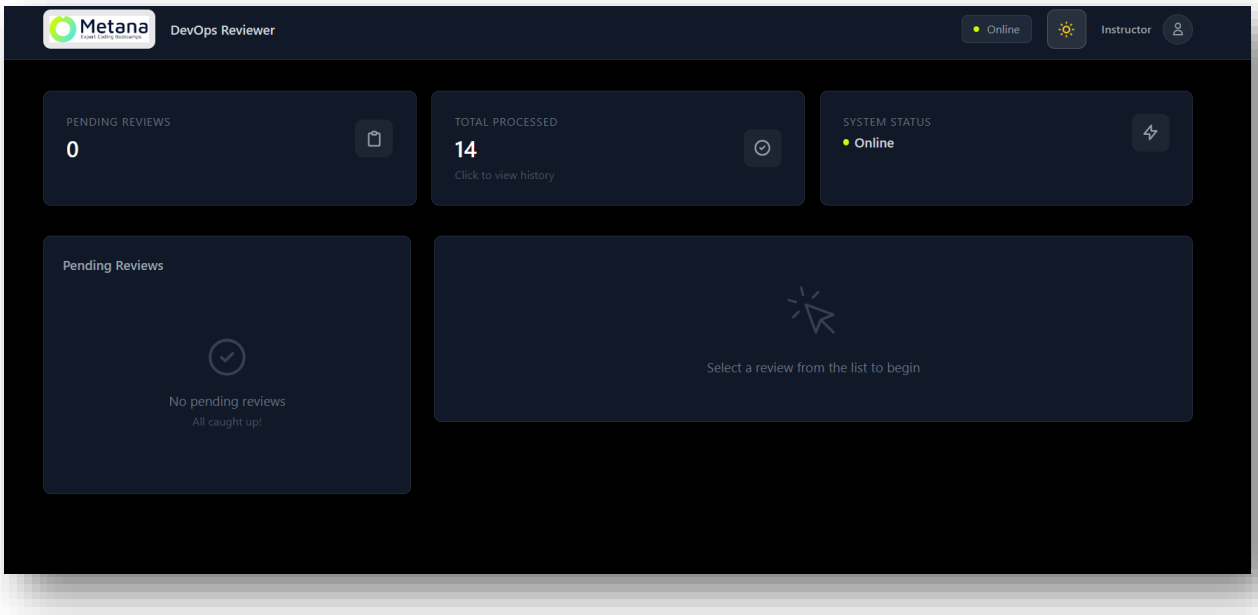

Figure 19 The empty state of the Instructor Dashboard, ready for new tasks.

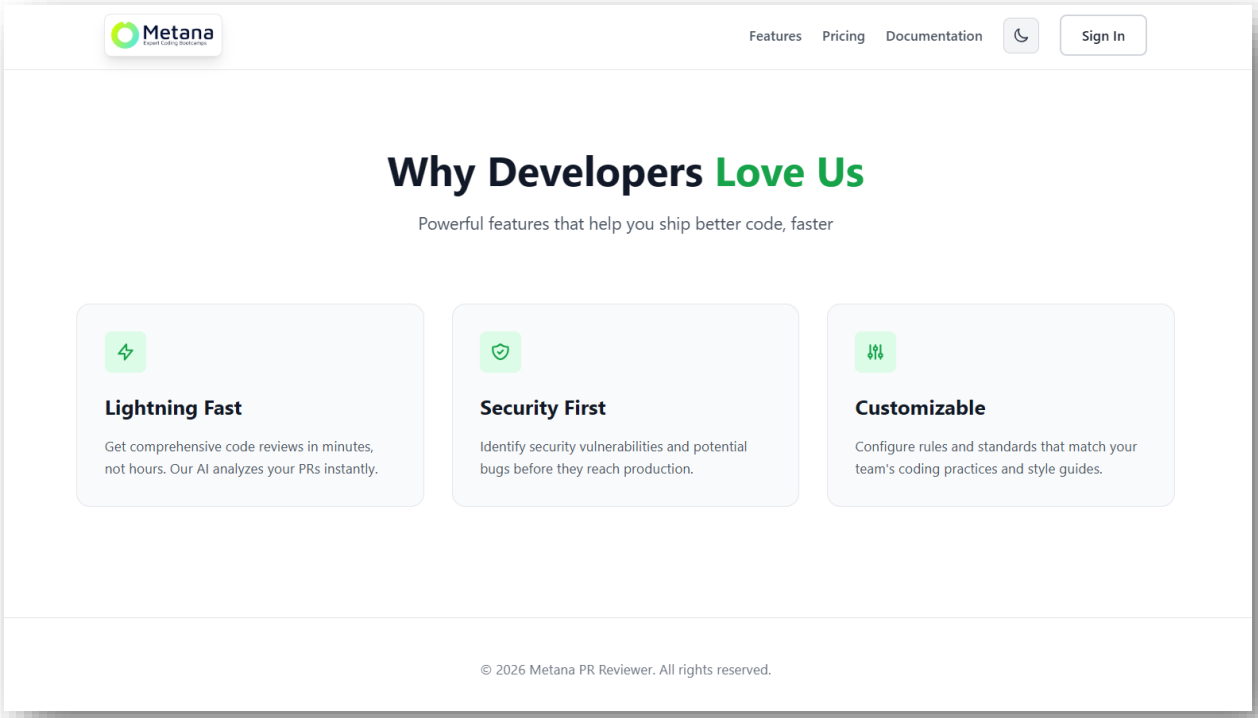

Figure 20 The Landing Page in Light Mode, demonstrating theme adaptability.

Conclusion

This project successfully meets all assessment objectives:

1. **Automated Webhook Triggers** (Demonstrated via Ngrok).
2. **Contextual Analysis** (AI feedback based on branch names).
3. **Instructor Gatekeeping** (Approval/Rejection workflow).
4. **High-Quality Frontend** (Professional UI with History & Themes).

The system is robust, error-tolerant, and ready for deployment.

Appendix:

1. Research & Problem Solving.

During the development phase, a significant challenge was encountered regarding **Infrastructure Connectivity**. Since the backend was running locally (localhost:3000), GitHub's cloud servers could not deliver Webhook events to the application.

To resolve this, I utilized AI as a research assistant to identify the best industry-standard solutions for secure tunneling.

The Research Prompt used:

"I am building a DevOps tool using Node.js and GitHub Webhooks. My server is running locally on port 3000. GitHub cannot send events to localhost.

What are the best tools or methods to expose my local server to the internet securely for testing webhooks? Please compare options like localtunnel, ngrok, and deploying to AWS."

The Solution Implemented: Based on the analysis, **Ngrok** was selected for its robust security features (HMAC verification support) and ease of integration with Node.js. This research step ensured the system was built on a reliable testing infrastructure before final deployment.

2. Sequence diagram