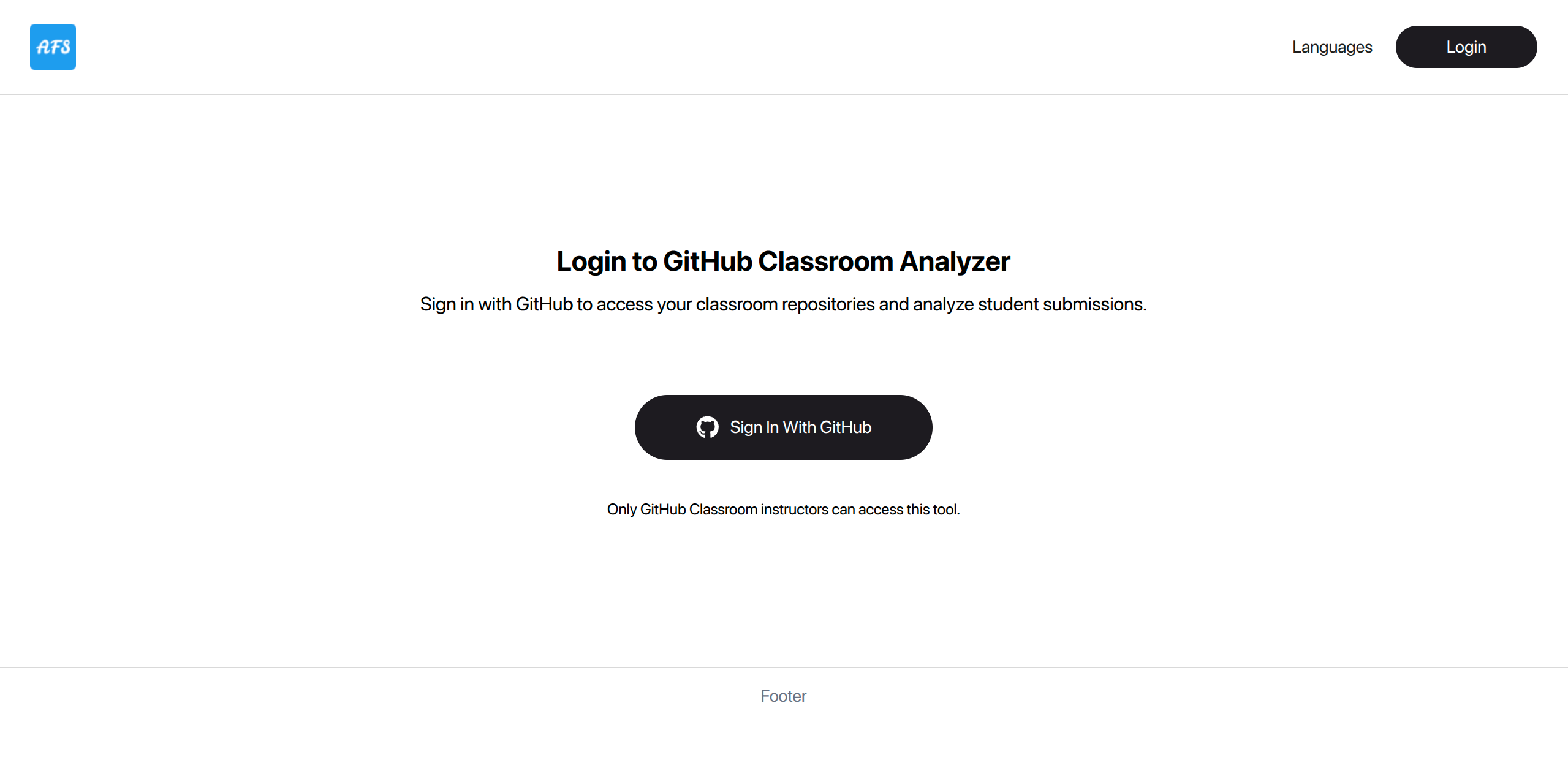
AI Feedback System

|  |  |
| --- | --- |
| **Professor** | **Jukka Nevalainen** |
| **Students** | [Helmi Griffiths](https://github.com/HelmiGr) |
| [Ville Matilainen](https://github.com/vima20) |
| [Sam Chou](https://github.com/FuzzyKala) |
| Repository | [Project Repository](https://github.com/Summer-project-25-AI-Feedback-system/AI-Feedback-System) |
| App Link | [WebSite](https://ai-feedback.live) |

### **1. Introduction to the AI Feedback System (AFS)**

The **AI Feedback System (AFS)** is a web-based application designed to help educators efficiently manage and evaluate student assignments. Built with a modern **React.js** and **TypeScript** frontend and a **Node.js** and **Express.js** backend, AFS automates the process of fetching, analysing, and providing feedback on student code using AI.



### **2. Core Functionality**

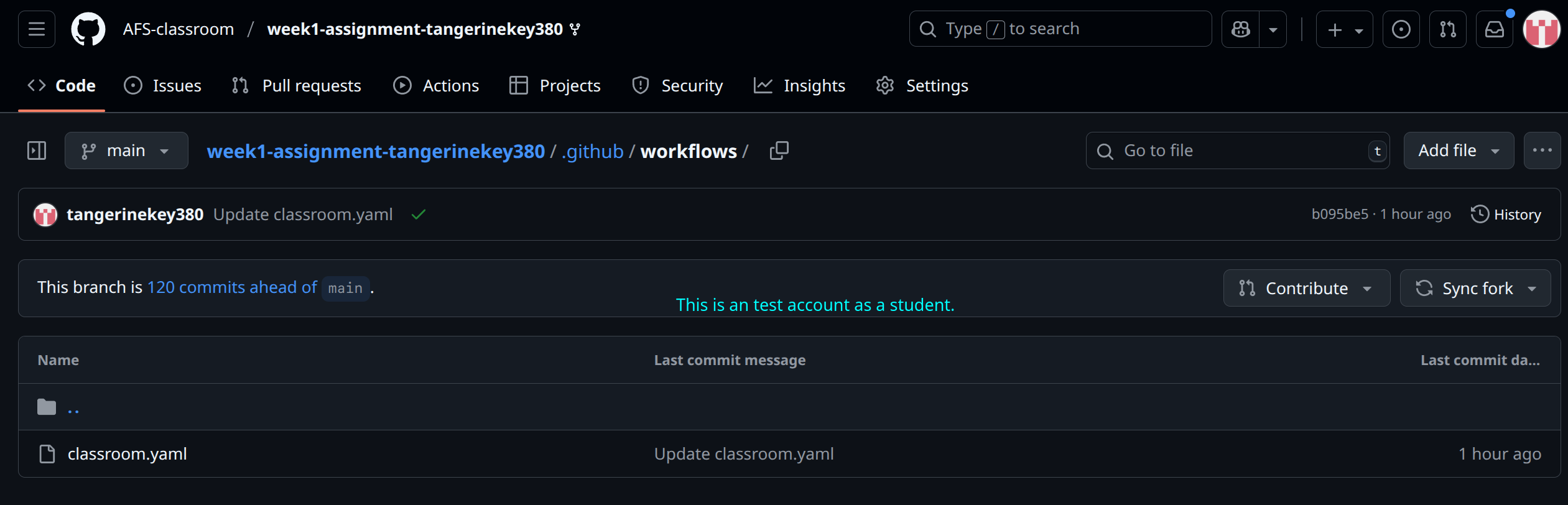
AFS integrates with **GitHub Classroom** to streamline the grading process. When a professor creates an assignment, GitHub Classroom creates a main repository. When a student accepts the assignment, it automatically forks this repository to create a private, dedicated one for the student.

The AFS system works by:

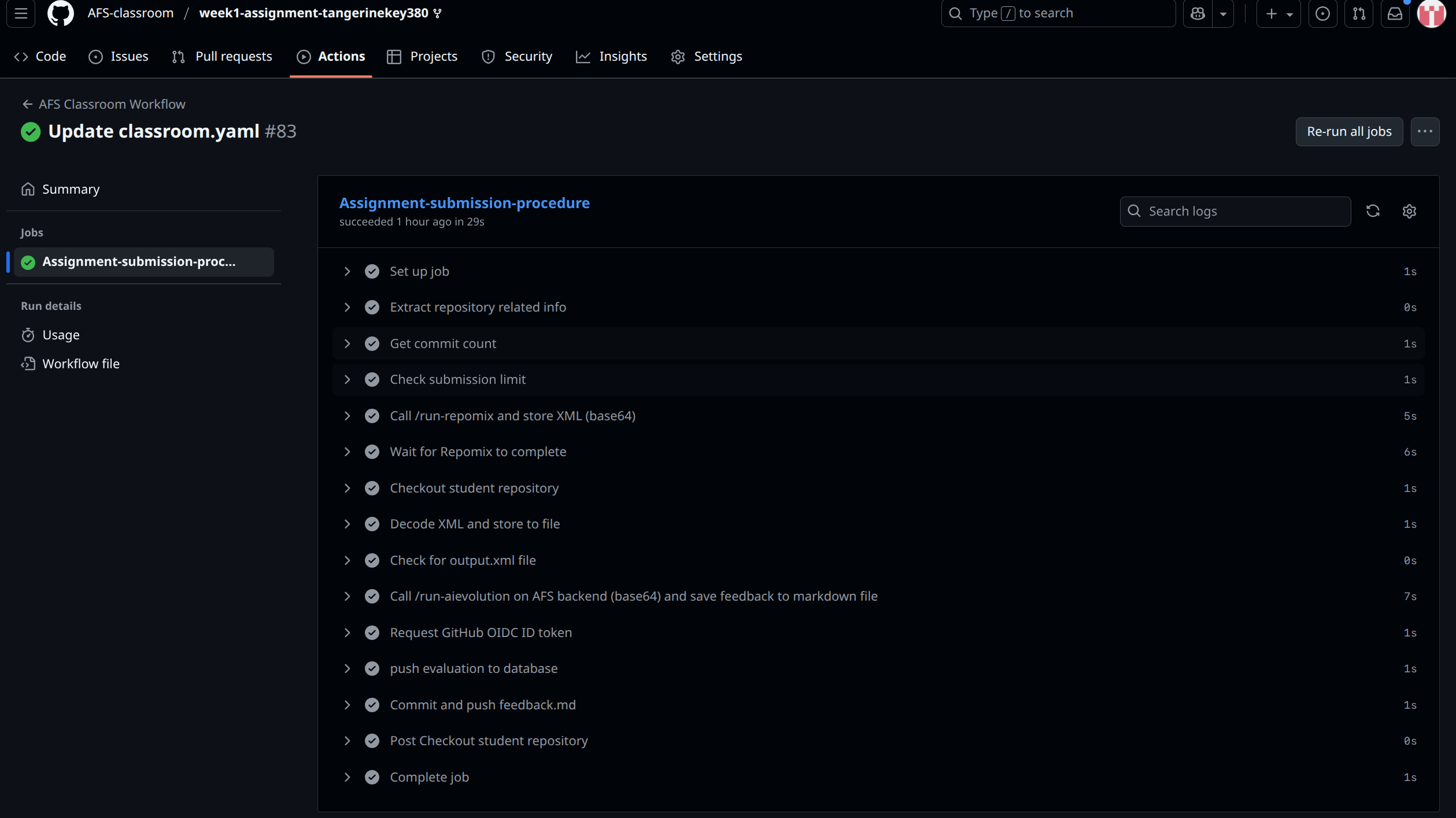
* **Repository Synchronization**: When a professor logs into AFS, the system uses the GitHub classroom API to access and display the organizations and assignments they have created. This provides a centralized dashboard for all their coursework.



* **Automated Evaluation:** A key feature of AFS is its automated feedback loop, which is triggered by a **GitHub Actions** workflow file (classroom.yaml) present in each student's forked repository. This workflow automatically runs whenever a student pushes new code.



* **AI Submission**: The GitHub Action first checks if the student has exceeded the professor-set submission limit. If not, it uses **Repomix**, a tool that simplifies the entire code repository into a single, structured **XML file**. This XML file is then sent to a third-party AI API (e.g., OpenAI) to generate feedback.

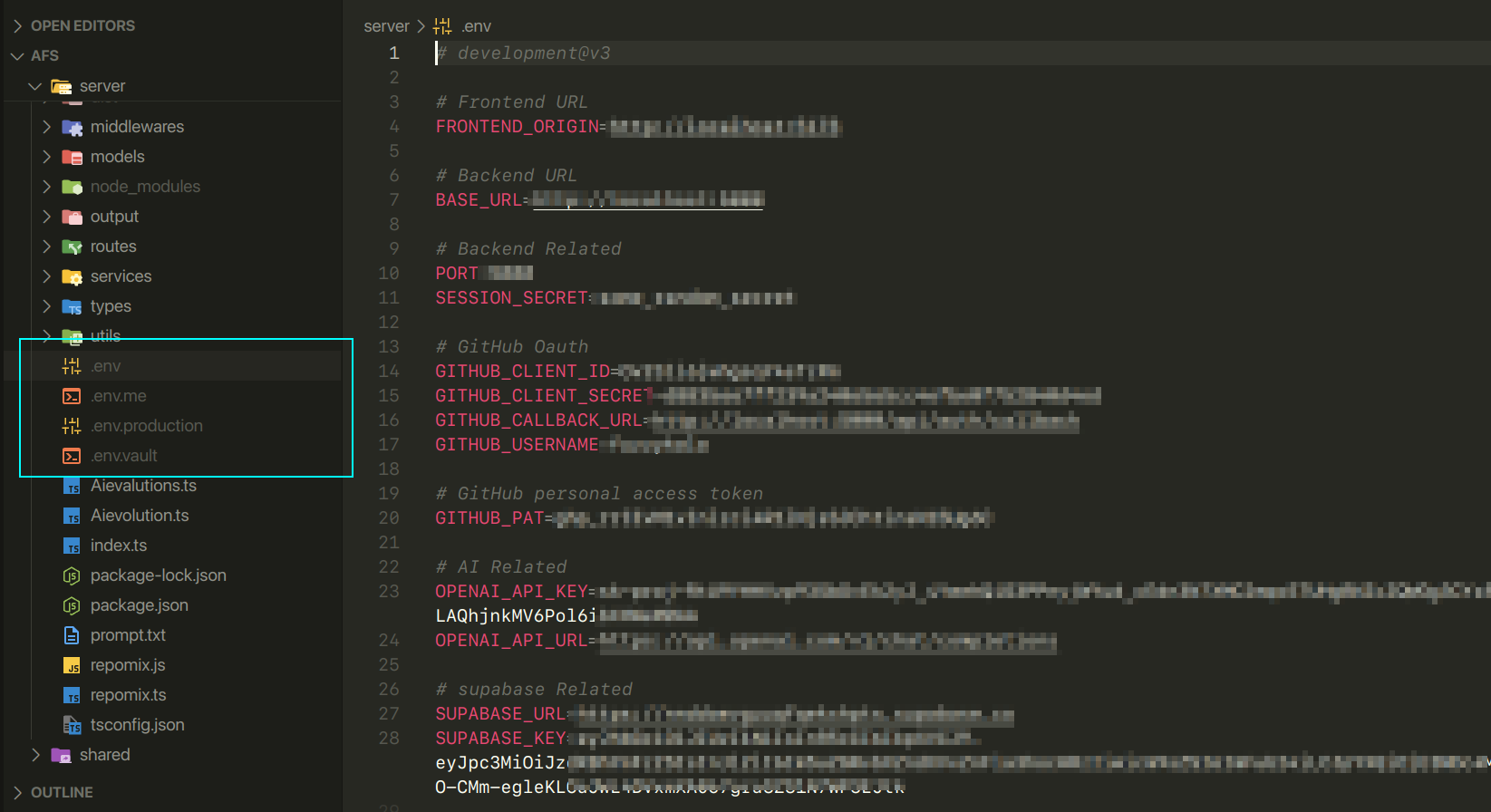


* **Feedback Delivery**: The AI provides detailed, constructive feedback on the student's code. AFS receives this feedback and saves it in a **Markdown file** (ASSIGNMENT\_EVALUATION.md) directly within the student's repository for easy access. The feedback is also stored in the AFS database for the professor to review.

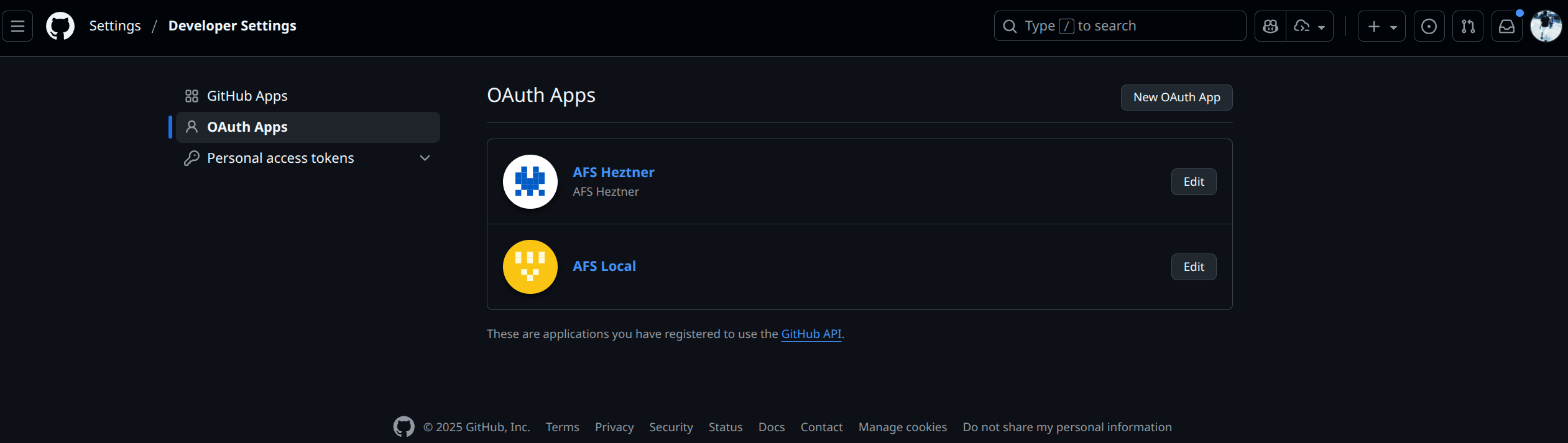


### **3. Key Technologies**

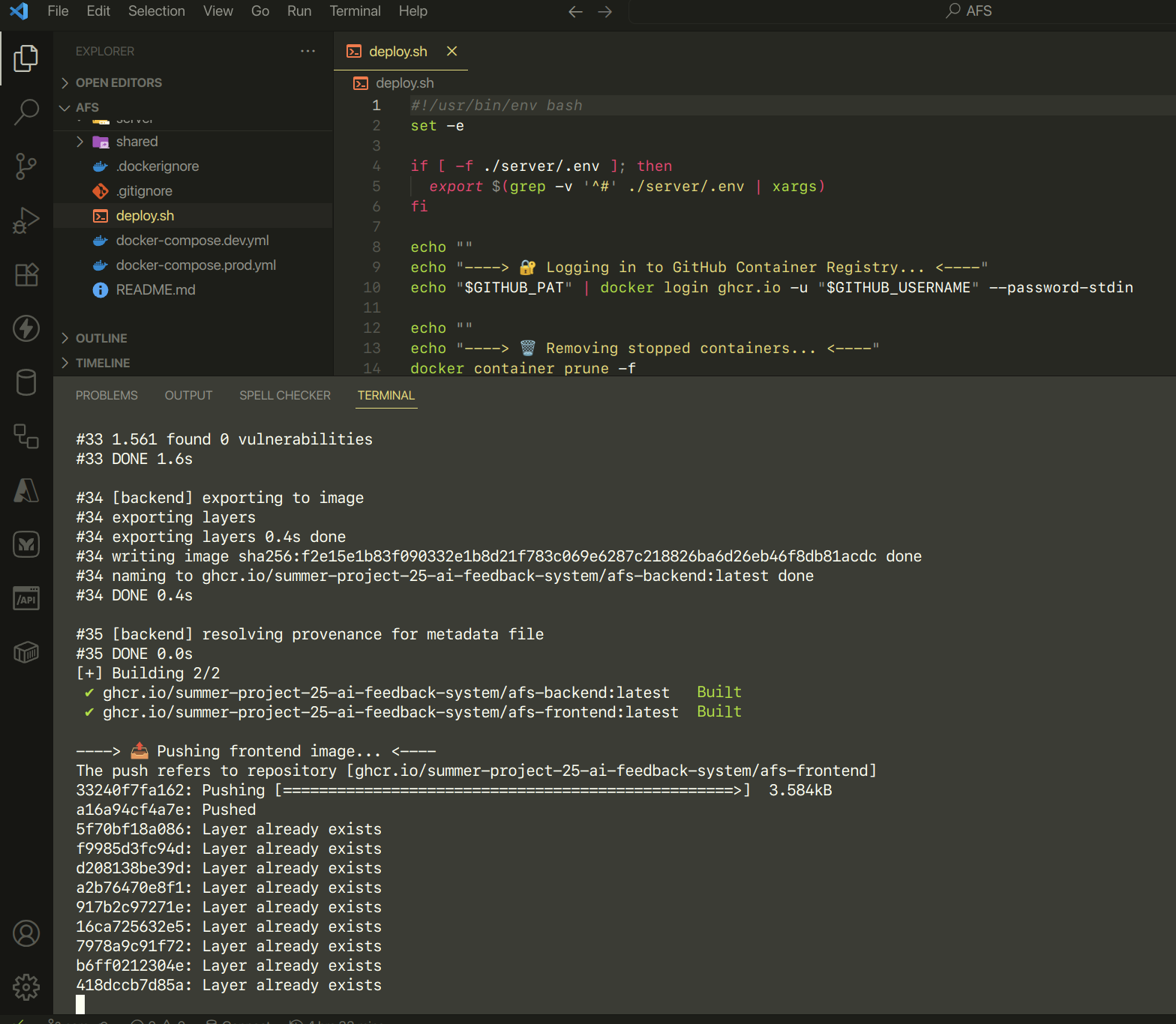
* **Frontend (User Interface):**
  + **React & TypeScript**: AFS is built with **React.js** for a dynamic user interface. **TypeScript** is used for type safety, which helps reduce errors and improves code robustness.
  + **Vite.js**: This build tool is used for the frontend, providing a fast development server with features like **Hot Module Replacement (HMR)** for instant UI updates.
* **Backend (Server Logic):**
  + **Node.js & Express.js**: The backend uses **Node.js** as its runtime environment and **Express.js** as a lightweight framework. This setup efficiently handles API requests, routing, and connections to external services like GitHub and the AI API.
  + **DotEnv**: The **DotEnv** library is used to manage environment variables, enabling seamless transitions between development and production environments.



* + GitHub OAuth App: **GitHub's OAuth** protocol, along with **Passport.js**, is used for secure user authentication in the backend.



* Deployment**:**
  + **Docker**: AFS uses **Docker** to containerize the backend and frontend into portable images.
  + **GHCR (GitHub Container Registry)**: **GHCR** is used to manage and host the Docker images. This allows for automated server deployment using a **docker-compose** file and shell scripts.



* **Core Tools:**
  + **GitHub Classroom**: A service that automates the distribution and collection of assignments.
  + **GitHub Actions**: A platform for automating software workflows directly within a repository, used here to trigger the code evaluation process.
  + **Repomix**: An open-source tool that packages an entire code repository into a single, AI-friendly XML file for efficient analysis.

### **4. User Workflow**

#### **Professor's Workflow**

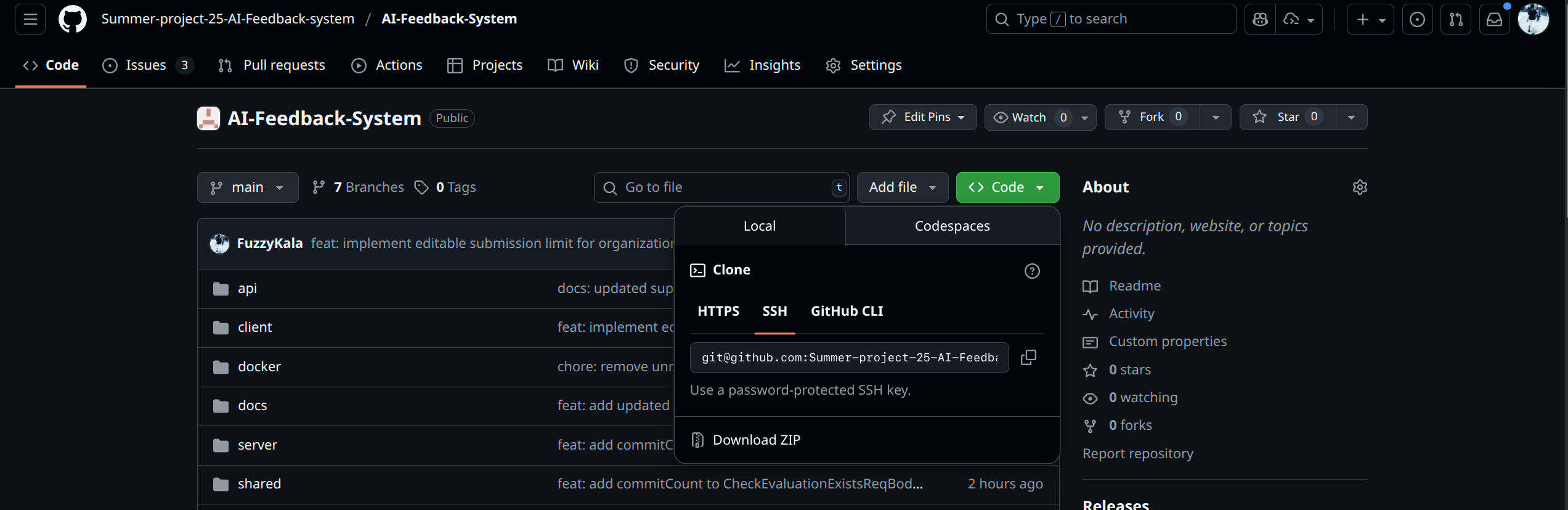
1. **Login**: Access the AFS web application and log in securely with your GitHub account.
2. **Dashboard**: View all your organizations and assignments from GitHub Classroom in a centralized dashboard.
3. **Manage Submissions**: From the dashboard, you can view student repositories and their automated feedback. You can also **set a submission limit** to control how many times a student can receive automated feedback.

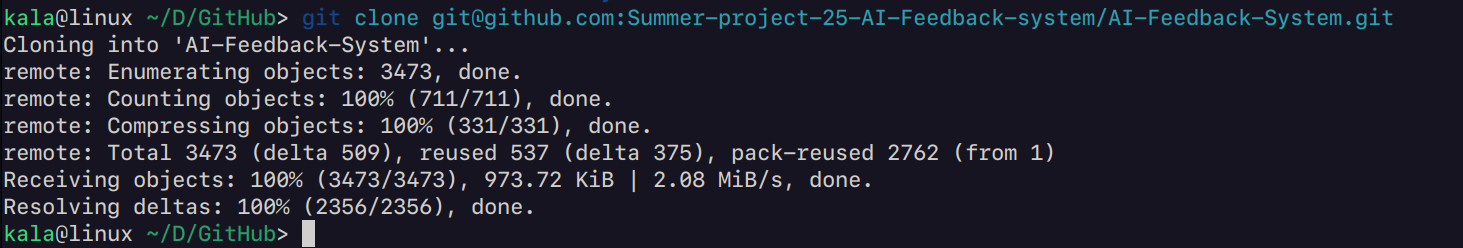
#### **Student's Workflow**

1. **Accept Assignment**: Accept the assignment link provided by your professor. GitHub Classroom will automatically create a new repository for you.
2. **Start Coding**: Work on the assignment in your local copy of the repository.
3. **Push to GitHub**: To receive feedback, **commit** and **push** your code to your GitHub repository.
4. **Receive Feedback**: The push action automatically triggers the AFS workflow. After a few moments, a new file named ASSIGNMENT\_EVALUATION.md will appear in your repository, containing the AI-generated feedback.

### **5. Local Deployment & Implementation:**

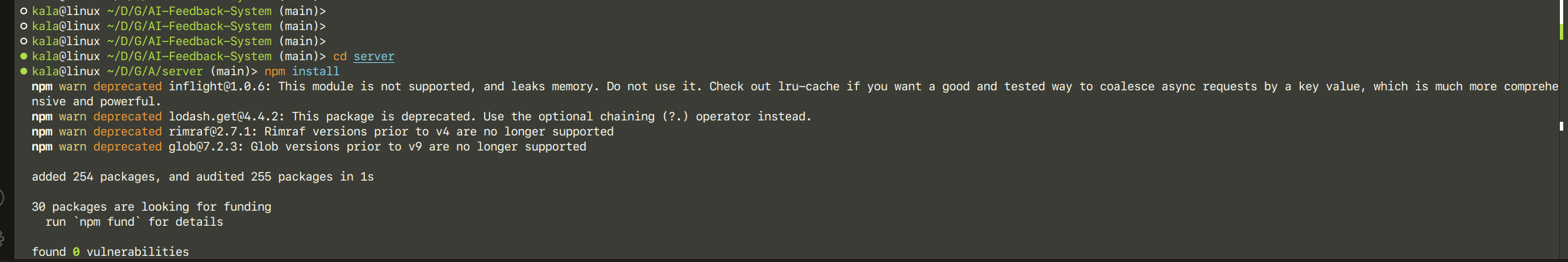
#### **Step 1: Get the Repository:**

Clone the repository from GitHub using either HTTPS, SSH, or the GitHub CLI.



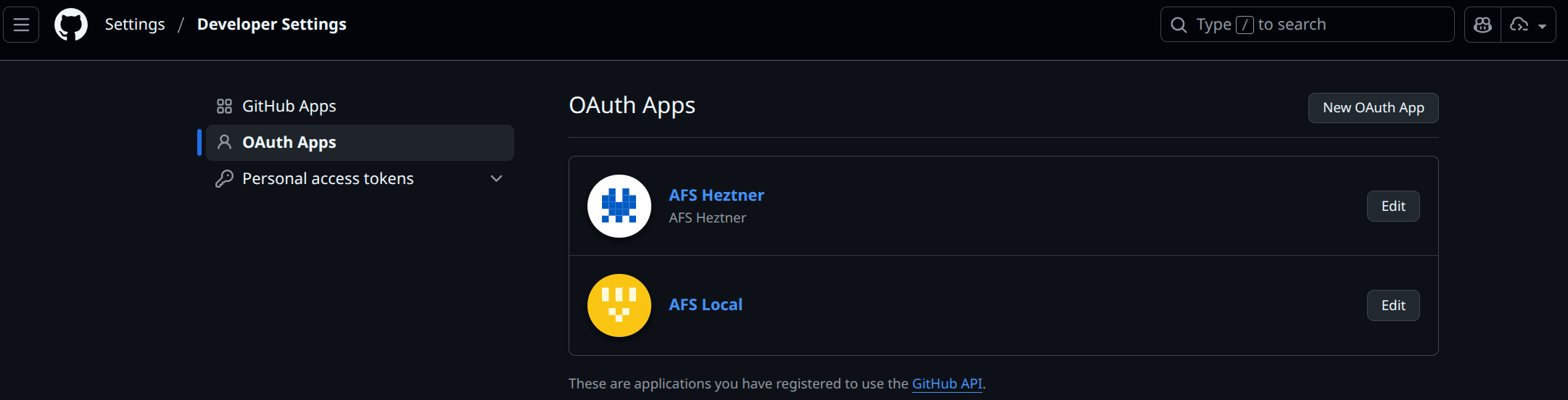
#### **Step 2: Install dependencies:**

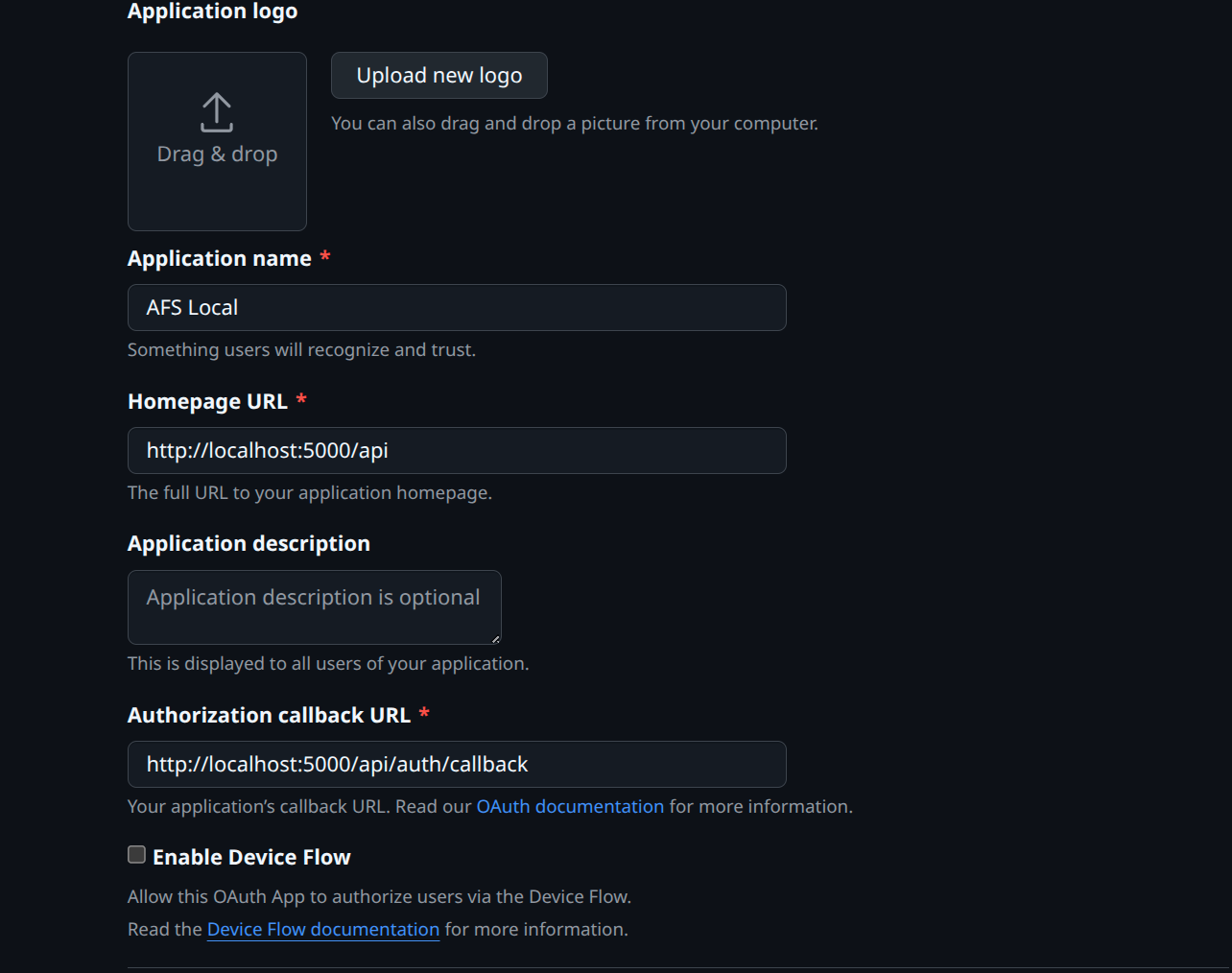
Navigate to both the client and server folders and run npm install to install all project dependencies.



#### **Step 3: Configure GitHub OAuth Apps & PAT (Backend)**:

Go to **GitHub Settings > Developer settings > OAuth Apps**. Create a new OAuth app for your project.





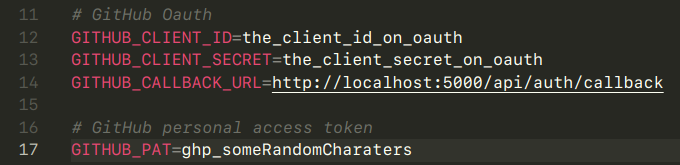
So far, the .env file would look like this:



Next, go to **Personal access tokens (classic)** and create a new token. Be sure to select the necessary scopes.



Add these new credentials to your backend .env file. You will also need to add your GitHub username for use in Docker deployments.

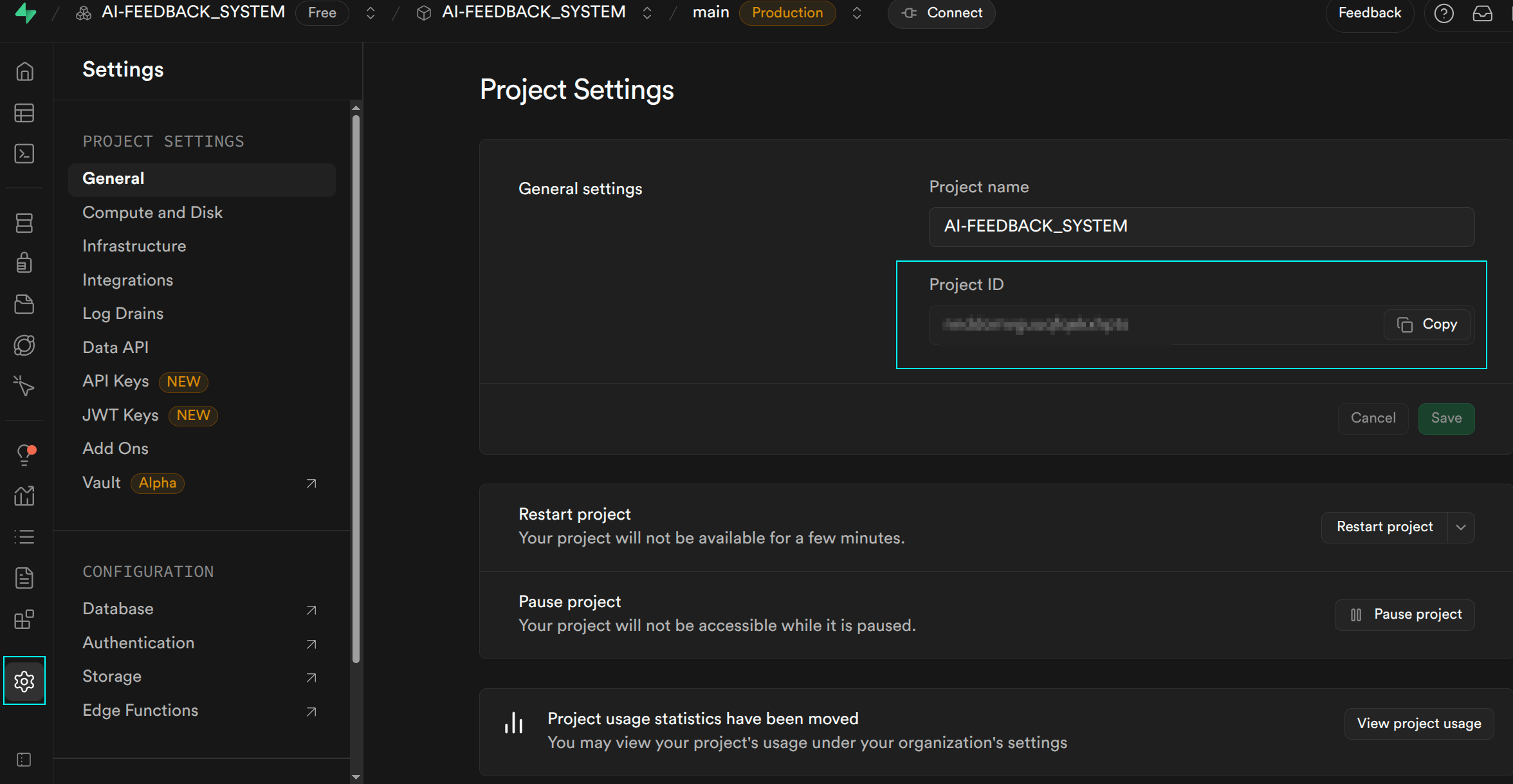


Also, add a variable GITHUB\_USERNAME for docker deployments uses, we will introduce this later, this should be your GitHub username.

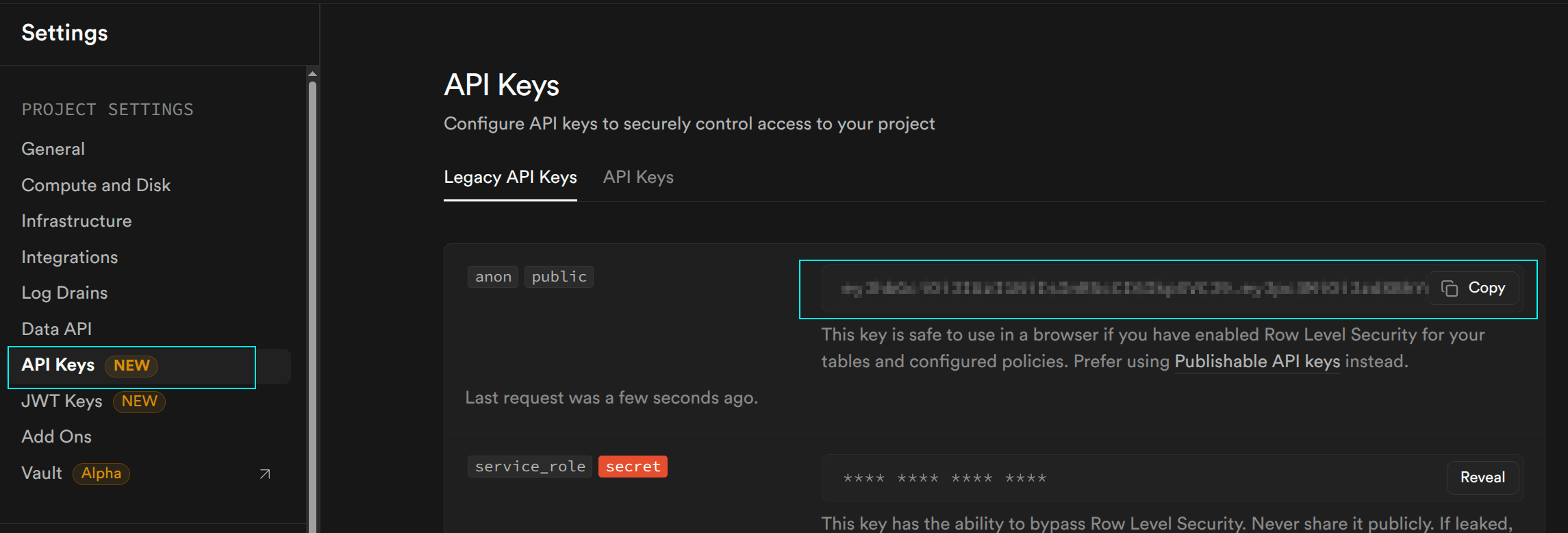


#### **Step 4: Configure Database Variables (Backend)**:

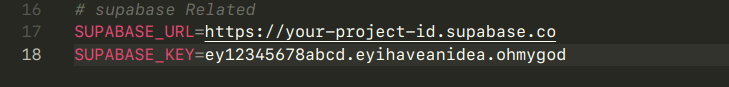
This project uses **Supabase** as its database. Go to your Supabase project's **Settings > General** to find your **Project ID**.



In **Settings > API Keys**, find your API key.

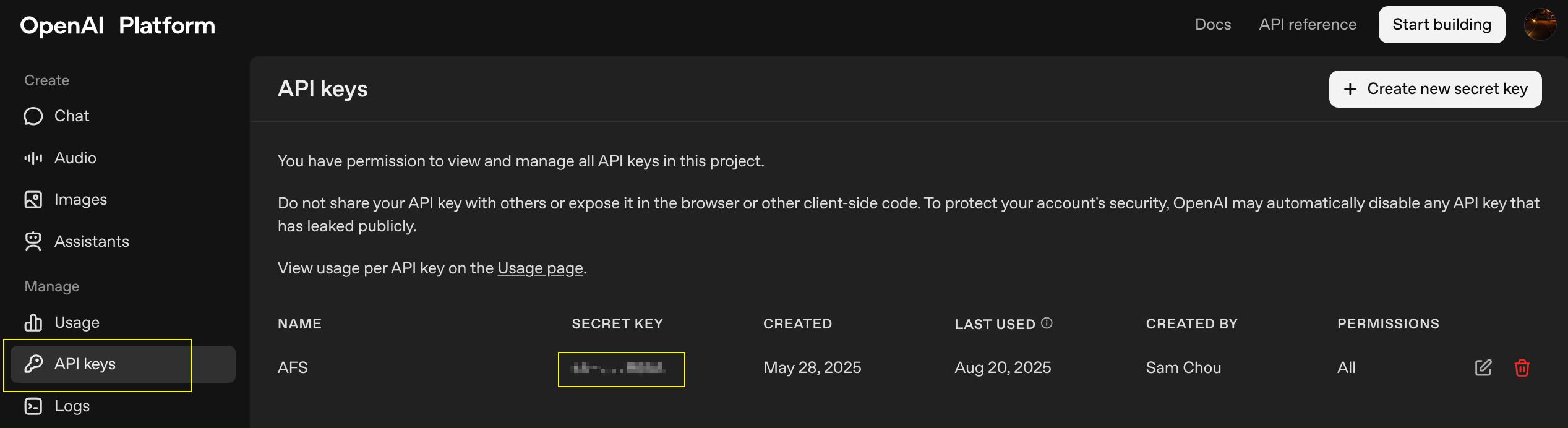


Add these credentials to your backend .env file.



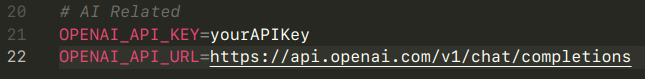
#### **Step 5: Configure AI Variables (Backend)**:

The system uses the OpenAI API. Obtain your **API Key** from the OpenAI Platform.



Set your OPENAI\_API\_URL to <https://api.openai.com/v1/chat/completions>.

Add these variables to your backend .env file.



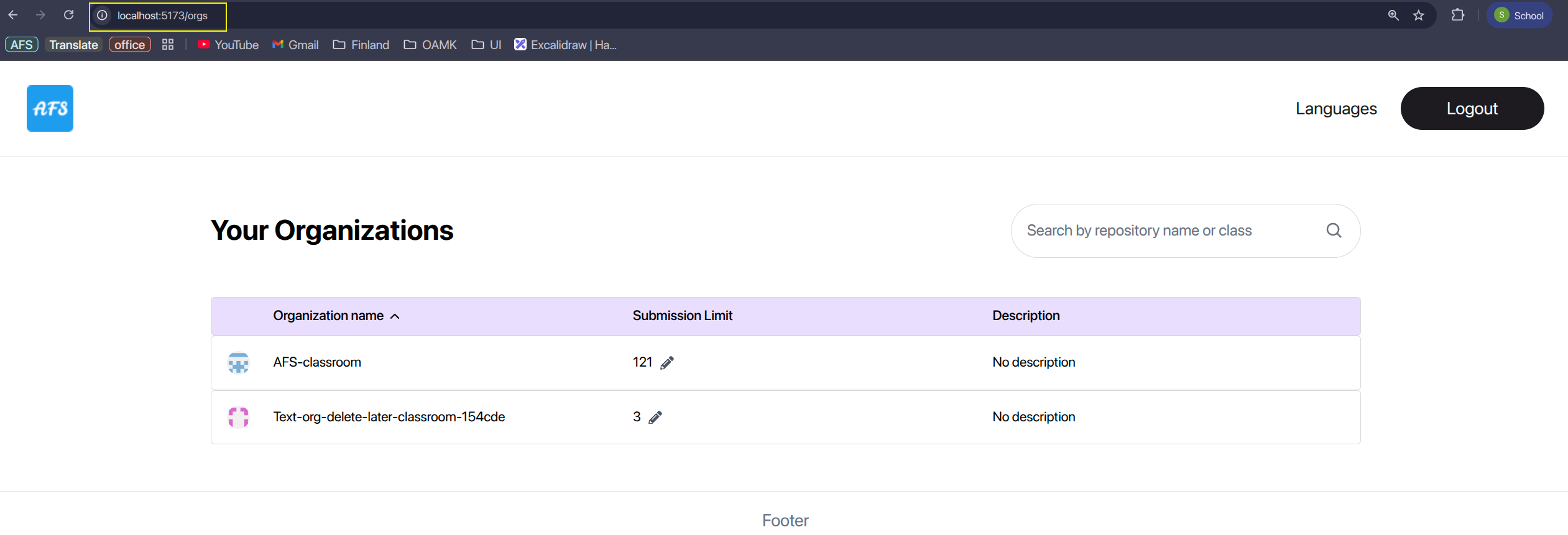
#### **Step 6: Configure Frontend Environment Variables**:

For Vite.js to recognize environment variables, they must be prefixed with VITE\_. Set the VITE\_API\_BASE\_URL in your frontend .env file to the address of your backend server..

#### **Step 7: Run Locally**:

In one terminal, run npm run dev in the /server directory.  


In a separate terminal, run npm run dev in the /client directory. The application should now be running on your local machine.



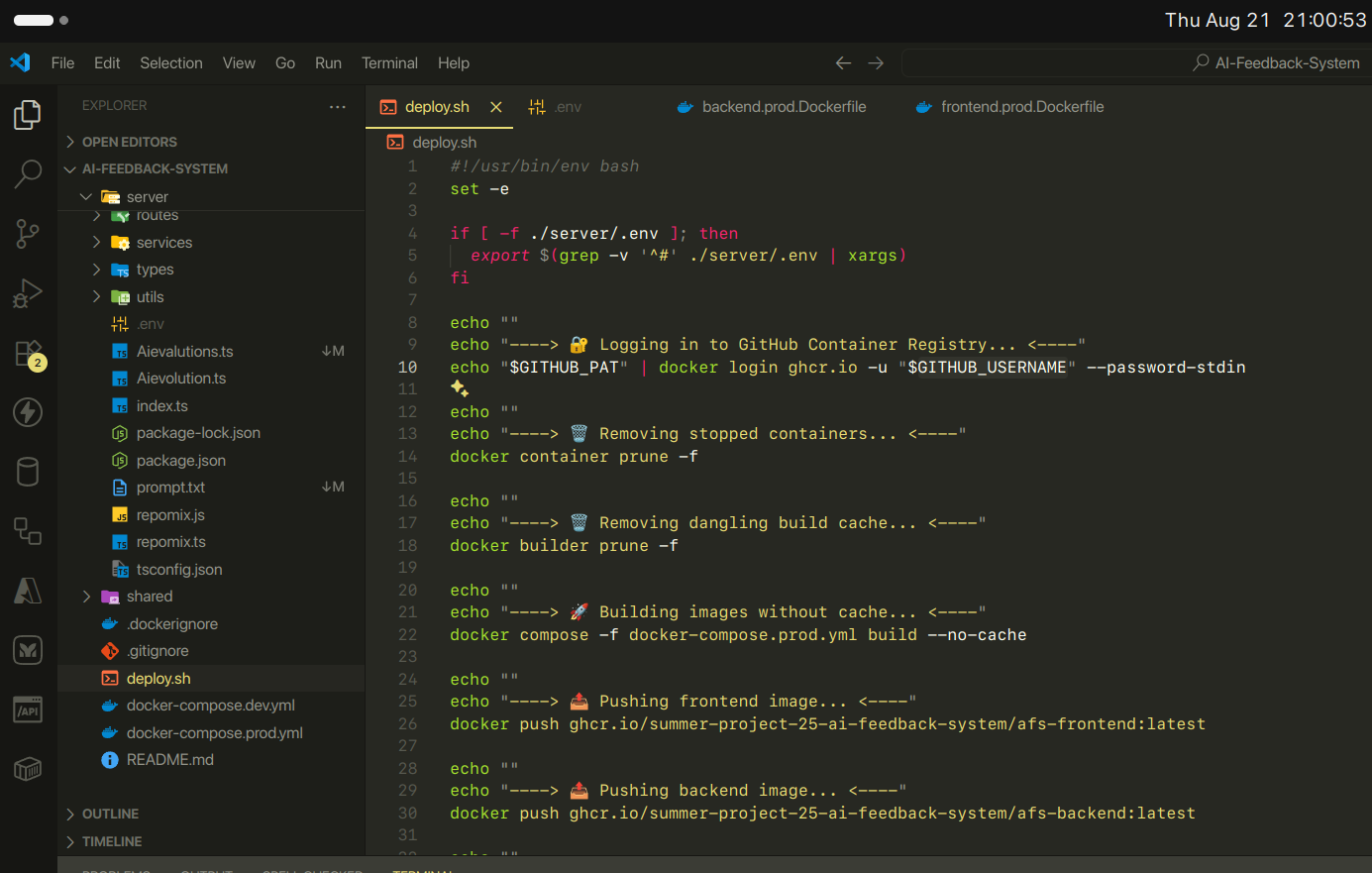
### **6. Remote Deployment & Implementation:**

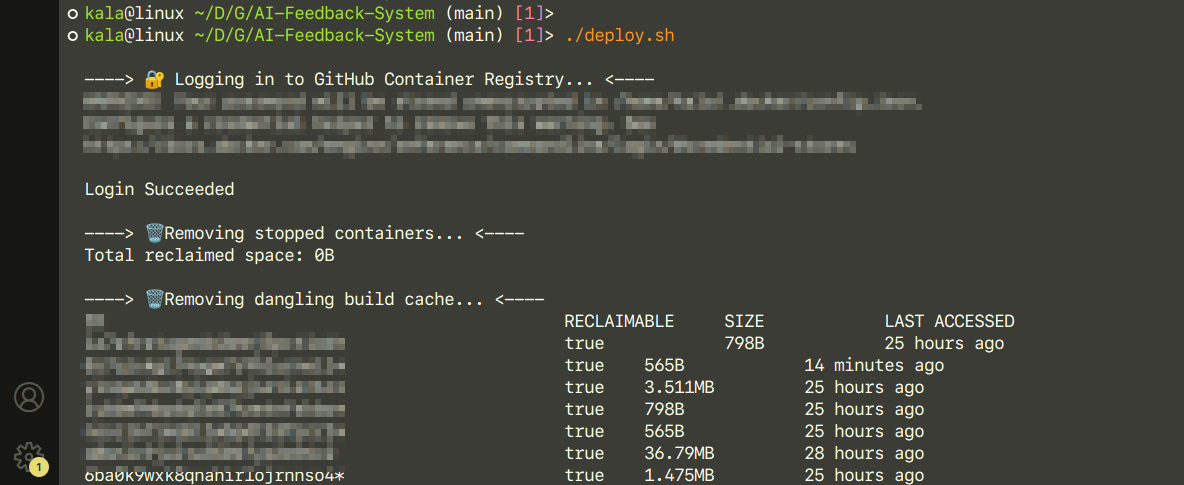
This guide demonstrates how to deploy the AFS backend and frontend using **Docker** and **GHCR** on a dedicated server.

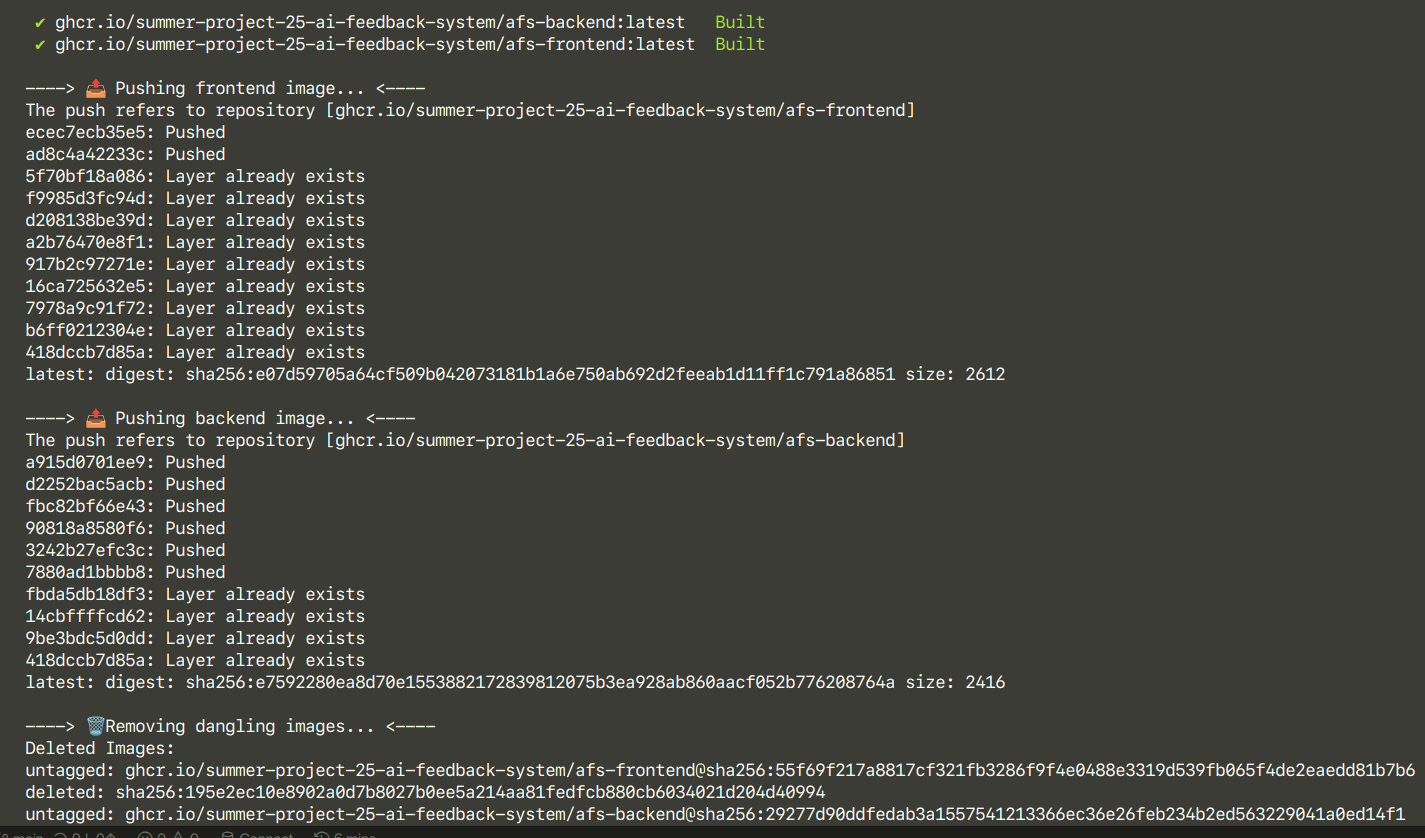
#### **Step 1: Build and Push Images to GHCR**:

Run the provided deploy.sh bash script from the project's root folder.

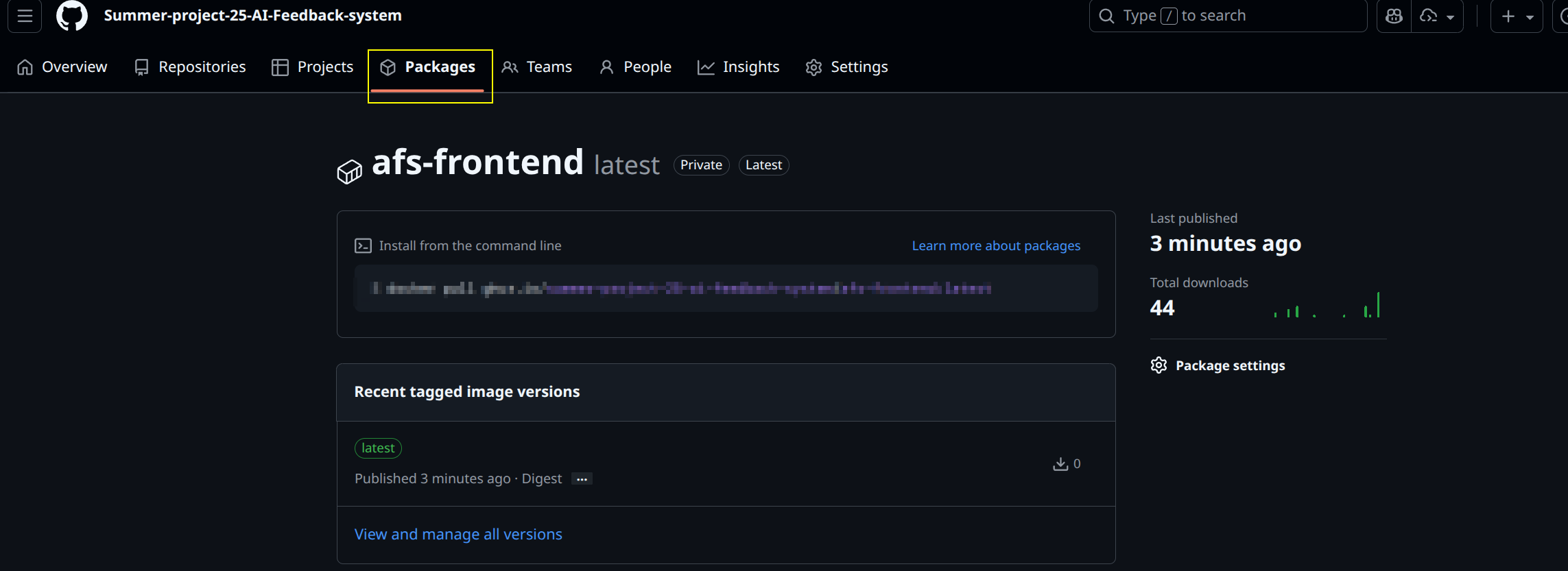
This script will log into GHCR using your GitHub PAT and username, build Docker images for both the frontend and backend, and push them to your repository on GHCR.

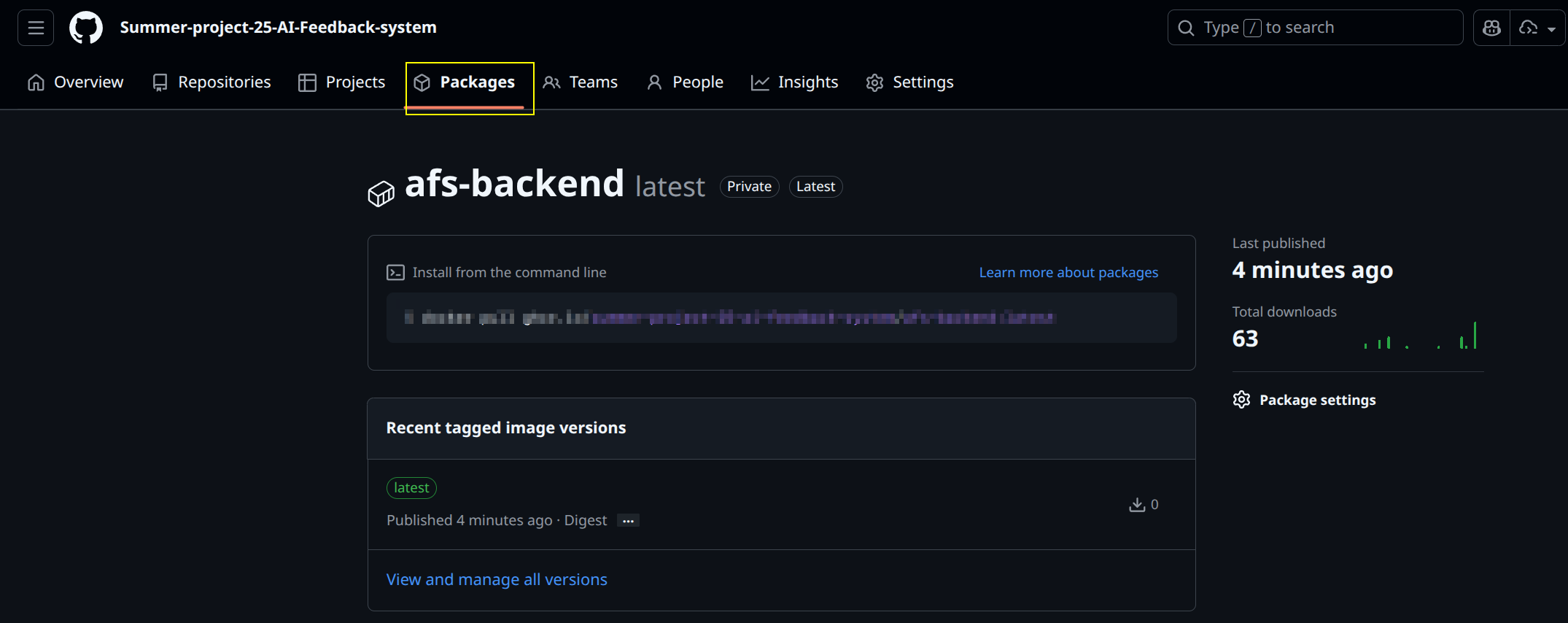




After building successfully, you should be able to see the following photo.  


And you can see the backend and frontend images as packages in your repository or organization.

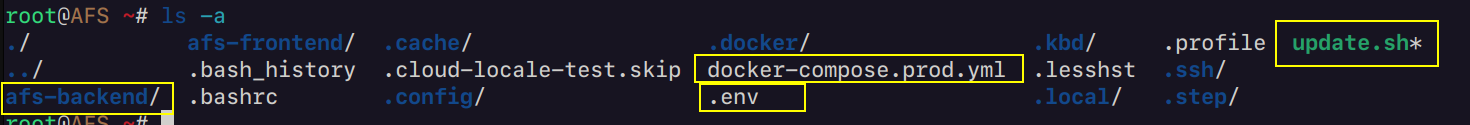




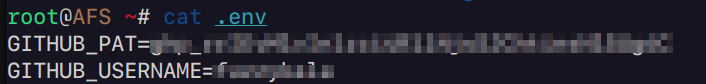
#### **Step 2: Prepare Deployment Files on the Server:**

On your server, create the following files and folders:

* A root .env file to store your GitHub user-related variables.
* An afs-backend folder containing an .env.production file with all backend environment variables, including the production-specific GitHub OAuth credentials.
* A docker-compose.prod.yml file, copied from the project's deployment folder.
* An update.sh script, also copied from the deployment folder, to automate the deployment process.

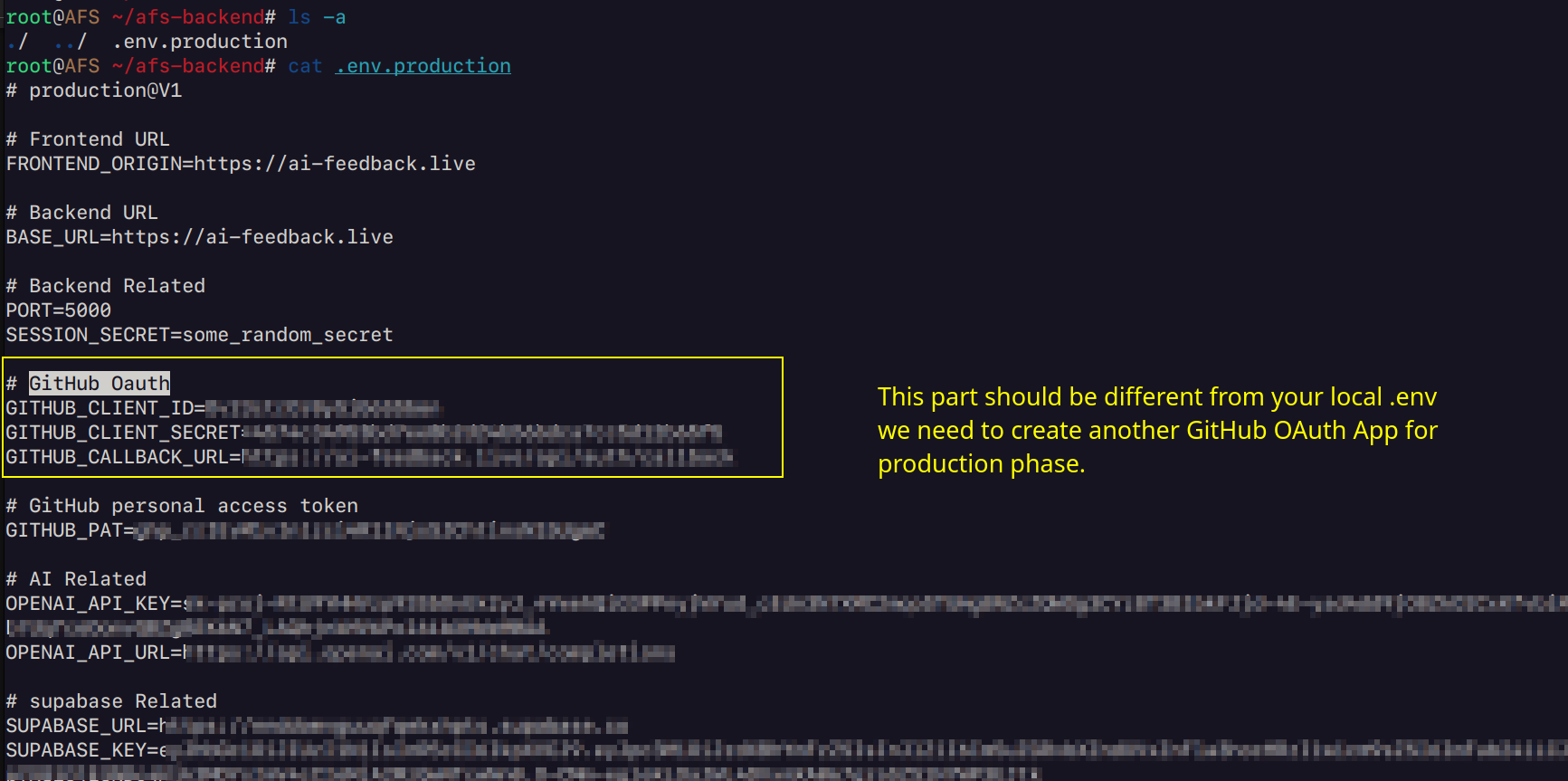


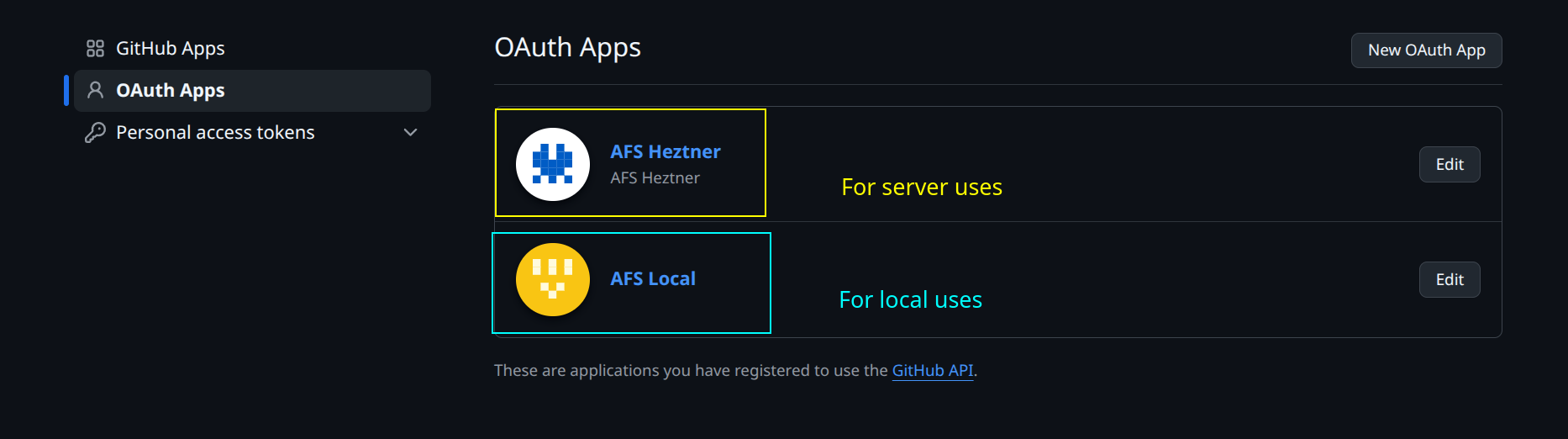
.env file should be look like the same as part of your local development .env.



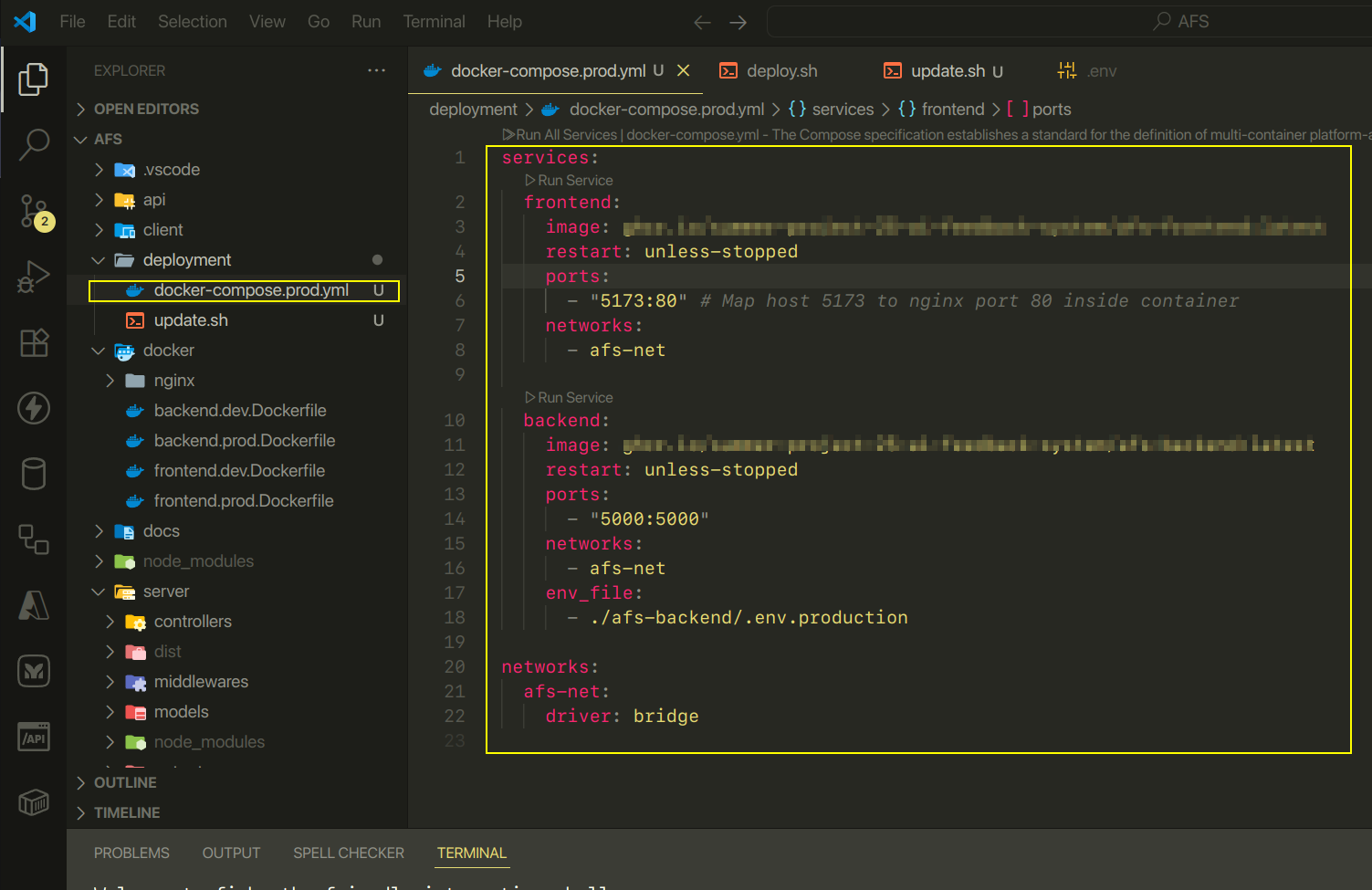
**afs-backend folder (Important):**

In this folder, create .env.production for server in the production phase. It should basically the same as your local development .env, but the GitHub OAuth part should be different.

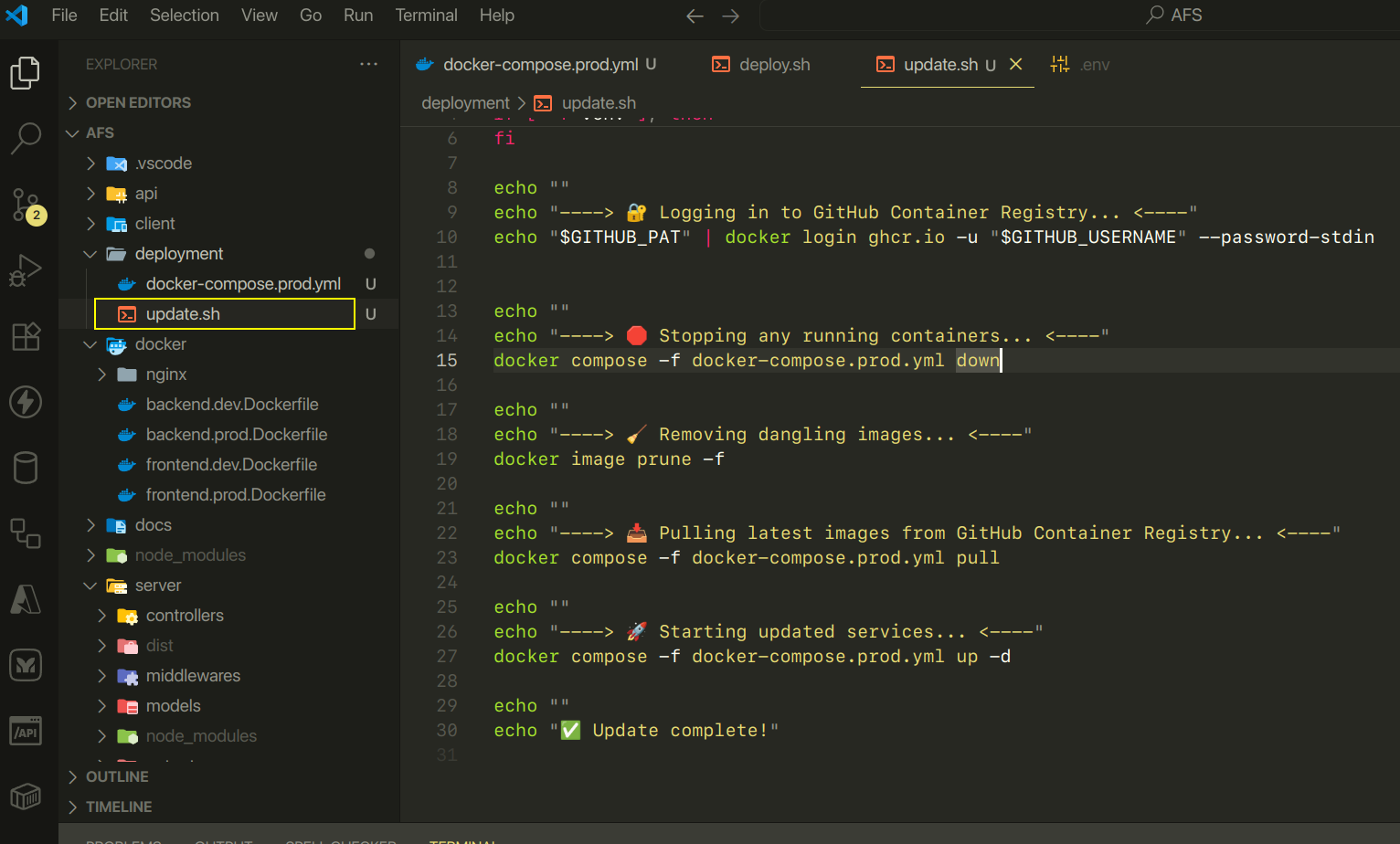




**docker-compose.prod.yml**: copy the file in the deployment folder and paste it into the server one.

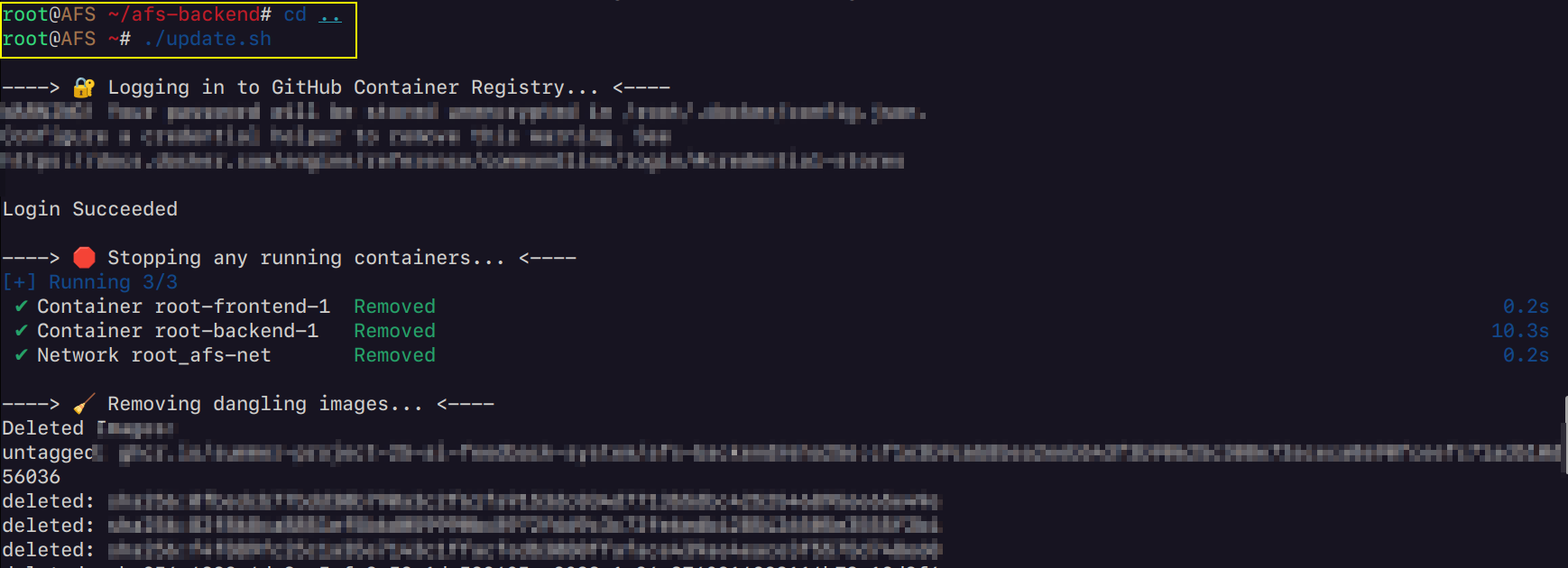
Caution: this file is different from the docker-compose.prod.yml under the root folder.  


**update.sh**: copy the file in the deployment folder and paste it into the server one.



#### **Step 3: Deploy the Images**:

From the server's root directory, run the ./update.sh script.



This script will pull the latest images from GHCR and deploy them using the docker-compose.prod.yml file. If successful, you will see a confirmation message on your console, and the application will be live.  
