

# Code Submission Tool Documentation

## Tool Logo:



**Github Link:** <https://github.com/Pyro-Warrior-1884/Assignment-Submission-Tool>

## Tool Summary

This tool is designed to simplify and streamline the process of code evaluation for educators. It provides two primary interfaces:

**Student Submission Portal** – where students can upload their code for evaluation.

**Instructor Dashboard** – where instructors can view submissions, execution outputs, and evaluation results.

When a student submits their code through the submission portal, the system automatically evaluates it and records the outputs. These results are then displayed on the instructor dashboard. Currently, the evaluation focuses on measuring model accuracy, making the tool particularly suitable for assignments involving neural network models. The output status displayed (Success, Failure, or Pending) is determined based on the accuracy of the submitted model when executed with the provided dataset.

## Tool Limitations

- The compiler currently supports machine learning and neural network codes only; other domains are not yet tested.
- The tool runs locally, requiring sufficient computational resources on the host device to execute model training and evaluation.
- All required dependencies and packages must be pre-installed on the host device. Missing packages will cause compilation failures, resulting in a zero score.
- The tool relies on locally stored datasets. If the dataset required by a student's code is missing, the submission will fail to execute.

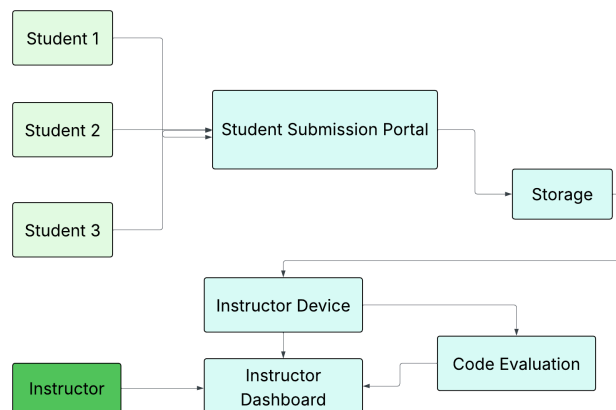
## Tool Features

- **Scalability:** Handles submissions from 60+ students efficiently. Evaluation time varies based on the complexity of the submitted code.
- **Detailed Submission Records:** The instructor dashboard provides:
  - Student name
  - Timestamp of submission
  - Evaluation status (Success, Failure, Pending)
  - Model accuracy
- **Real-Time Updates:** Submissions are updated live, keeping instructors informed of student activity.
- **Pending Status Tracking:** Even if evaluation has not yet occurred, the submission appears in the dashboard with status marked as Pending, clearly indicating it is awaiting execution.
- **Plagiarism Check:** Although not integrated into the primary evaluation pipeline, the tool provides an additional capability to support academic integrity. After code submissions are received, the tool automatically downloads all student code files while evaluation is taking place. These files can then be processed by a dedicated plagiarism detection module, which analyzes the submissions and generates plagiarism scores. The output

highlights the highest similarity levels among students, enabling instructors to quickly identify potential cases of copied work.

- **Report Generation:** Upon compilation, the tool automatically generates a detailed execution report. This report records the outcome of each code cell, indicating whether the execution was successful or if it encountered an error. In the case of errors, the corresponding error messages are captured and documented. All reports are stored locally, providing instructors with a comprehensive record of each submission's execution history for review and reference.
- **Data Export:** The tool provides an option for instructors to download all dashboard data as an Excel sheet, enabling offline review and record-keeping.

## System Overview



The system is designed around two primary user groups: students and instructors. Students interact with the Student Submission Portal, where they upload their code for evaluation. Each submission is stored in the system's Storage module to ensure persistence and traceability. Once submitted, the code is processed by the Code Evaluation engine, which executes the code using predefined datasets and records the results, including model accuracy and execution status.

On the instructor's side, the Instructor Device provides access to the Instructor Dashboard, where results are displayed in real time. The dashboard allows instructors to monitor all submissions, track their evaluation status, and review performance metrics such as accuracy. This clear separation of responsibilities—students focusing on submission, the system handling storage and evaluation, and instructors reviewing outcomes—ensures a streamlined and transparent workflow.

## Prerequisites

Before installing, ensure you have the following:

- Laptop/PC: Minimum 8 GB RAM (suitable for model training).
- Operating System: Windows 10 or later.
- GitHub Account: Required for deployment via Render.
- Google Account: Required for using Render, Vercel, and Firebase.
- Submission Constraint: All submitted codes must be neural network models.

## Installation

## 1. Clone the Repository

Download the project files from GitHub:

**git clone https://github.com/Pyro-Warrior-1884/Assignment-Submission-Tool**

This will create a local copy of the repository on your machine.

## 2. Install Dependencies

- Navigate to the cloned repository.
- Install all required Node.js modules and dependencies.
- Download and configure the necessary Vercel CLI dependencies.

## 3. Deploy Frontend Services (Vercel)

The tool contains two frontend modules:

- submission-frontend → Instructor-facing portal.
- submit-frontend → Student submission portal.

To deploy each frontend:

1. Open the folder in VS Code.
2. Run the deployment command:  
**vercel**  
This will automatically deploy the respective frontend to Vercel.

## 4. Deploy Backend Service (Render)

The backend code is located in NNDL\_backend and must be deployed separately:

1. Push the NNDL\_backend folder into a separate GitHub repository.
2. Go to Render, sign in with your GitHub account, and connect the new repository.
3. Deploy the backend service from there.

## 5. Configure Firebase Database

1. Create a Firebase project and set up a real-time database.
2. Generate the API service key for your Firebase project.
3. In the backend project, create a .env file and add your service key as follows:  
FIREBASE\_SERVICE\_KEY=<your-service-key>  
(You may choose a different variable name, but ensure it matches the references in the code.)

## 6. Running the Compiler

The system requires a Go-based compiler to process submitted notebooks.

1. Open the file **run\_go\_compiler.bat** and update the **cd** command to point to the location of the **go\_jupyter\_converter** directory.
2. Open VS Code, navigate to the **go\_jupyter\_converter** folder, and start the compiler by running:  
**node connection.js**
3. Keep this process running. The compiler will sequentially execute student submissions and update their status.

- If stopped, no further submissions will be compiled.

## 7. Using the Tool

- Share the Vercel-deployed [submit-frontend](#) link with students for submissions.
- Instructor submissions and results will appear in the Vercel-deployed [submission-frontend](#) link.
- Initially, student submissions will be shown with status: Pending. Once the compiler processes them, results will be updated automatically.

## 8. Output Storage

- Decoded Notebooks: Saved in the [decoded\\_notebooks](#) folder.
- Execution Reports: Saved in the [outputs](#) folder.

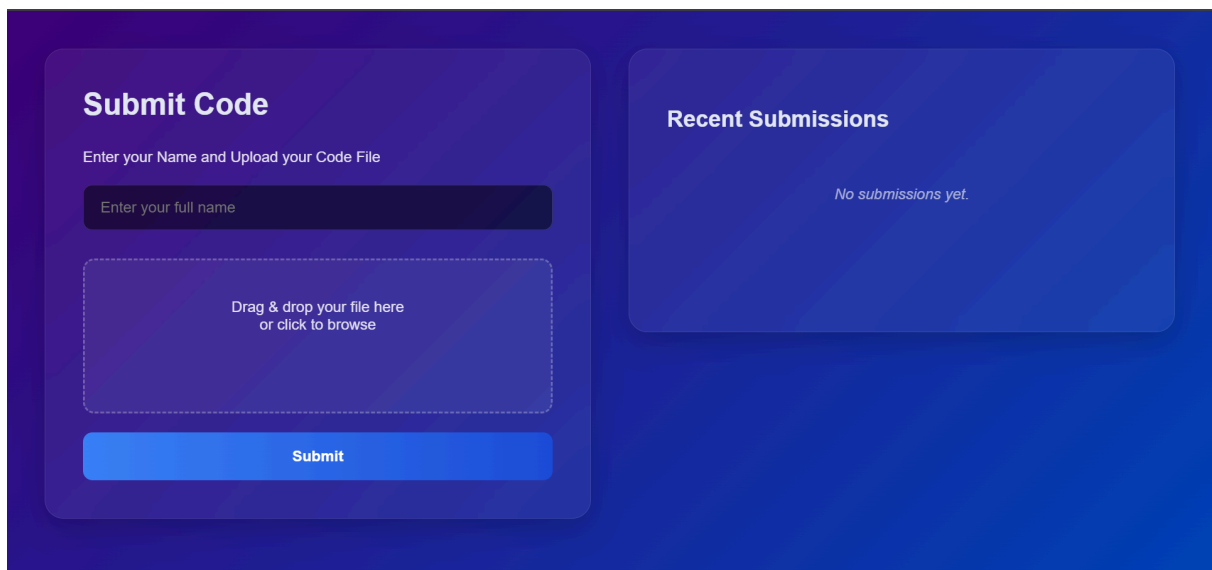
Both folders are located in your home directory. If you wish to change the folder names or locations, you may edit the corresponding paths in [connection.js](#).

## Usage Guide

### Student Workflow:

Students interact with the **Submission Portal** to upload their code. The process is as follows:

1. Open the submission site.

The image shows a web interface for submitting code. It has a dark blue background. On the left, there's a 'Submit Code' section with a title, a subtitle 'Enter your Name and Upload your Code File', a text input field for 'Enter your full name', a dashed box for file upload with the text 'Drag & drop your file here or click to browse', and a blue 'Submit' button. On the right, there's a 'Recent Submissions' section with the text 'No submissions yet.'

2. Enter the required details (e.g., student name).

3. Upload the code file.

4. Click **Submit**.

If the submission is successful, a confirmation message will be displayed.

### Submit Code

Enter your Name and Upload your Code File

Drag & drop your file here  
or click to browse

Submit

### Recent Submissions

Johnathon Doe	12-09-2025 01:57
<pre>CB.EN.U4CSE22505.ipynb {   "cells": [     {       ...and 168 more lines     }   ] } Status: Pending</pre>	

In some cases, the submission may take additional time to process; during this period, students should wait until their submission appears in the recent submissions list.

### Submit Code

Enter your Name and Upload your Code File

Selected: CB.EN.U4CSE22505.ipynb

Submit

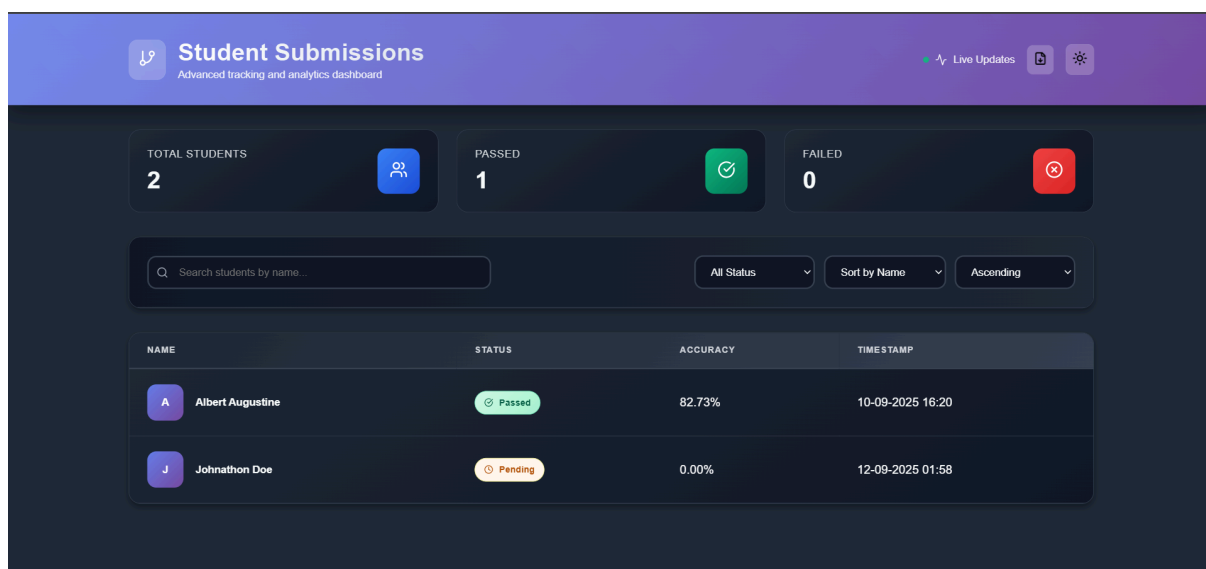
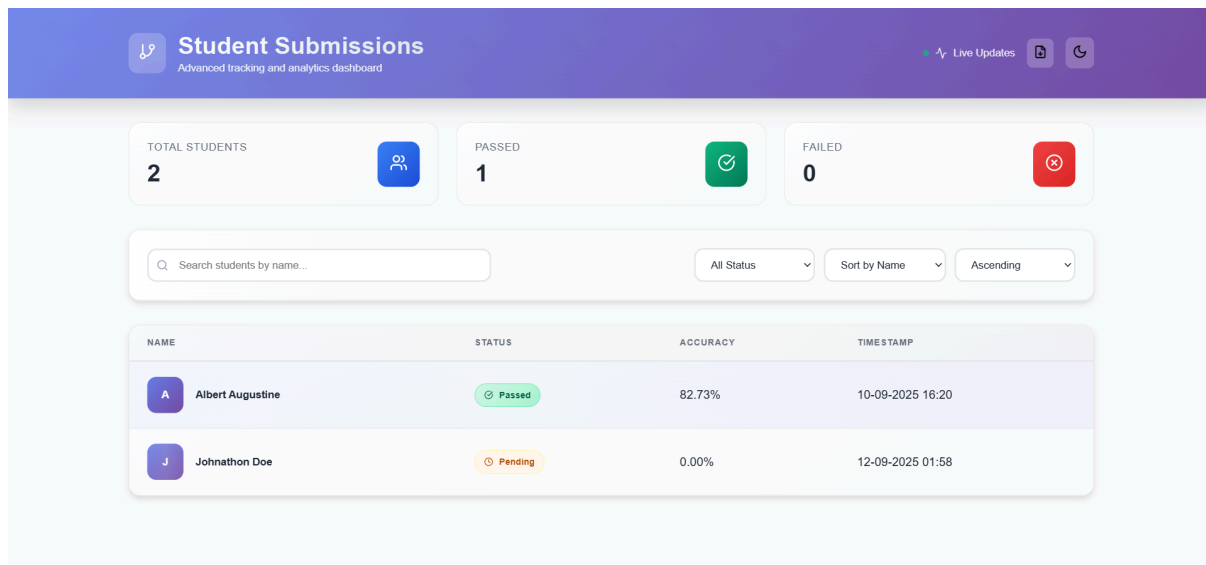
### Recent Submissions

No submissions yet.

Once submitted, the code is recorded and queued for evaluation.

### Instructor Workflow:

Instructors access the **Dashboard**, which supports both light and dark mode. The dashboard provides real-time visibility into all student submissions.



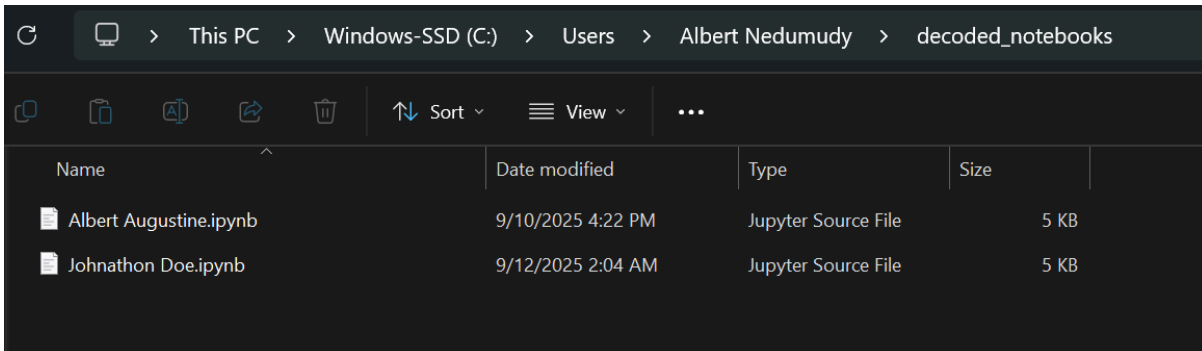
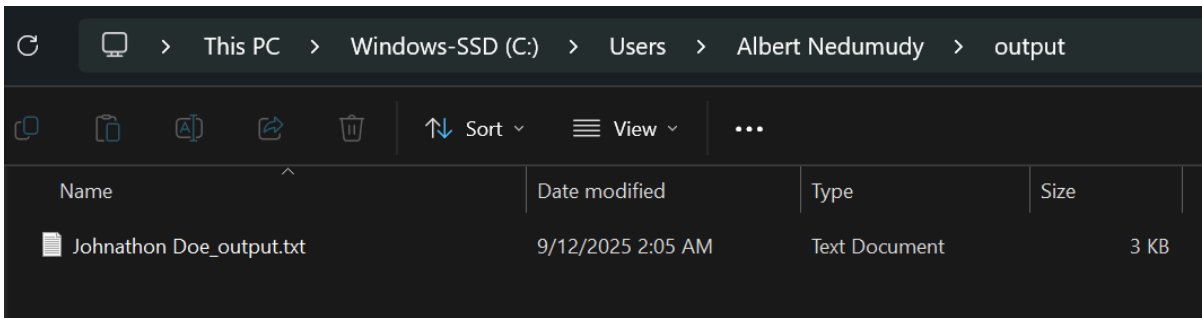
## 1. Activating the Compiler

- Navigate to the directory containing the compiler connection file.
- Run the compiler to begin evaluation.
- Submissions will then be processed sequentially, with results automatically updated on the dashboard.

```
PS C:\Users\Albert Nedumudy\Desktop\Main HQ\Work\NNDL_Project> cd .\go_jupyter_converter\  
PS C:\Users\Albert Nedumudy\Desktop\Main HQ\Work\NNDL_Project\go_jupyter_converter> node .\connection.js  
[x] Processed: Johnathon Doe | Status: Success | Accuracy: 82.37
```

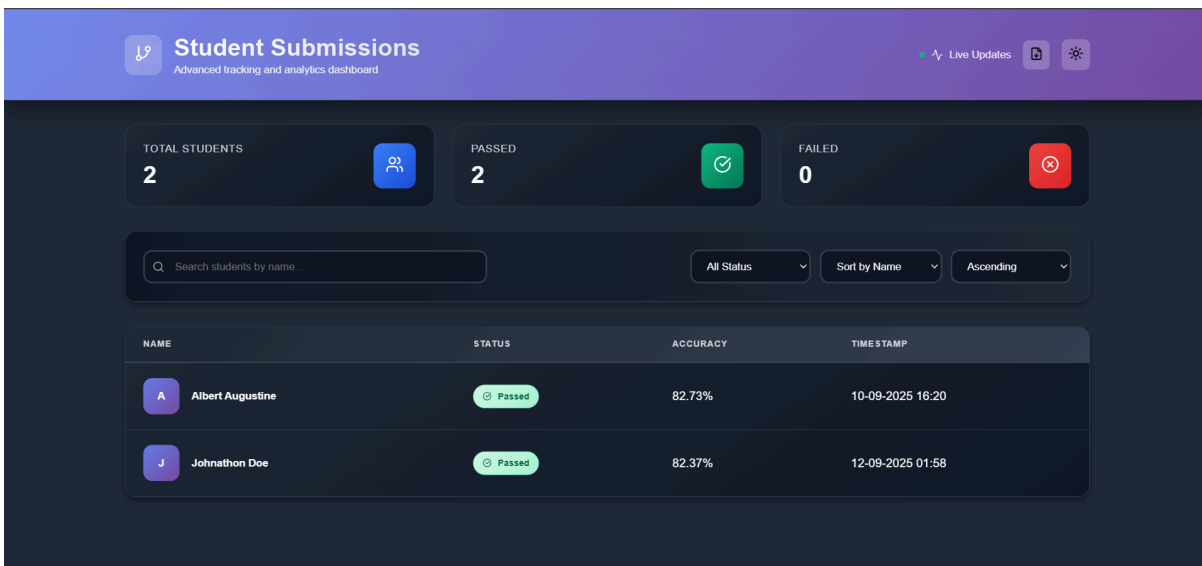
## 2. Viewing Reports and Student Code

- Execution reports are stored in the **outputs** folder at the project root.
- Student-submitted code files are stored in the **decoded\_notebooks** folder at the same level.



### 3. Monitoring Submission Status

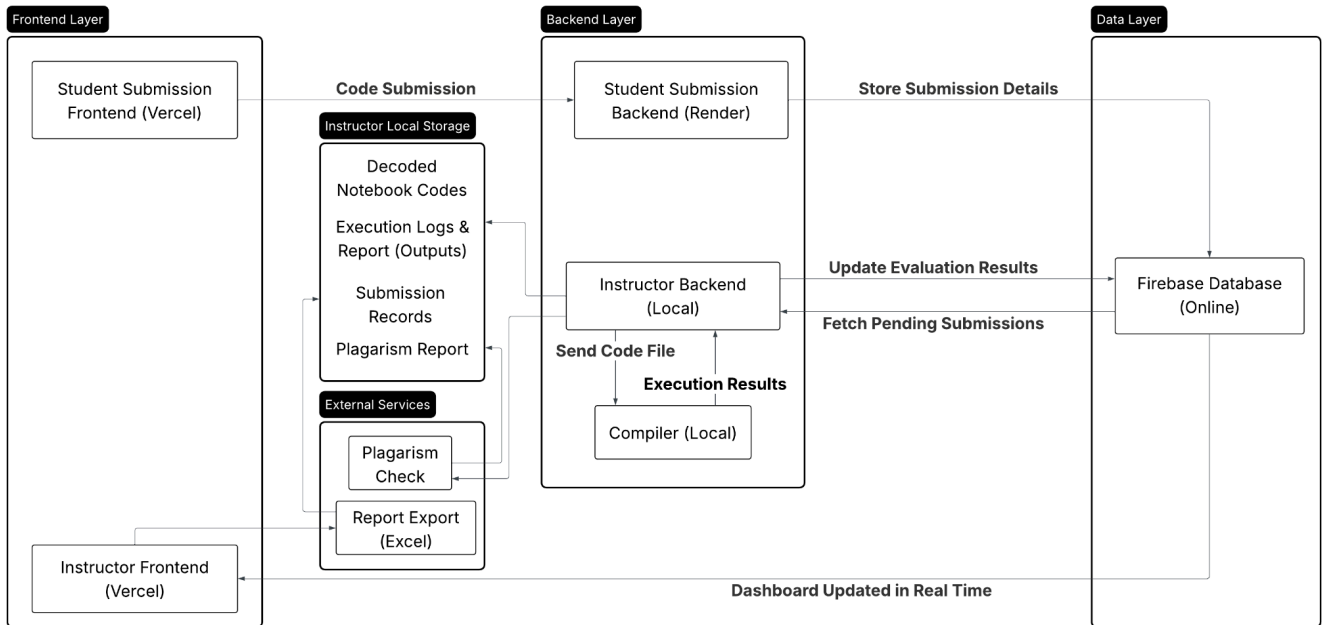
- After compilation, the dashboard reflects updated statuses for each submission (*Success*, *Failure*, *Pending*).



The dashboard also provides additional functionality:

- **Search:** Look up students by name.
- **Sort:** Organize submissions by name, accuracy, or timestamp (ascending or descending).
- **Filter:** View submissions by status (Success, Failure, Pending).

## System Architecture Design



## Overview

The system is designed using a three-layered architecture consisting of the Frontend Layer, Backend Layer, and Data Layer, ensuring clear separation of responsibilities and smooth interaction between components.

The **Frontend Layer**, deployed on Vercel, provides user interfaces for both students and instructors, enabling code submission and real-time monitoring of results.

The **Backend Layer** manages the processing workflow, where student submissions are routed through the online backend service (Render) and executed locally by the instructor's backend and compiler. This design supports scalable submission handling while maintaining secure execution.

The **Data Layer**, powered by Firebase, ensures persistent storage of submission records, evaluation outputs, and status updates, which are reflected back to the dashboard in real time.

Additionally, local storage and external services (such as plagiarism checks and report exports) complement the core layers, enhancing the system's utility for academic evaluation.

## Tech Behind Layers

### Frontend Layer

#### Student Submission Portal (submit-frontend)

The student-facing application is implemented using **React.js with Vite** for optimized builds and fast rendering. Styling is handled through **CSS** ([App.css](#), [index.css](#)), and the project is deployed via **Vercel** for scalable hosting.

#### Key functionalities:

- File upload interface implemented with [react-dropzone](#).
- Form handling and validation via [react-hook-form](#).
- Submissions are transmitted to the backend using **REST APIs** with JSON payloads over **HTTPS**.
- Core logic resides in [App.jsx](#) and [main.jsx](#), which bootstrap the React app.

#### Instructor Dashboard (submission-frontend)



The instructor-facing frontend is also built with **React + Vite** and styled using custom CSS. This dashboard provides real-time visibility into submissions and evaluation results.

#### Key functionalities:

- Data synchronization achieved through **Firebase Firestore**, enabling instantaneous updates without page reloads.
- Interface components (`StudentSubmissions.jsx`, `StudentSubmission.css`) render tables, filters, and submission details.

**Deployment:** Both frontends are deployed on **Vercel**, ensuring global accessibility with minimal configuration overhead.

### Backend Layer (NNDL\_Backend)

The backend consists of multiple services working in coordination:

#### 1. Submission Handling Backend

- Acts as the entry point for student uploads.
- Receives submissions through REST endpoints and stores them for further processing.
- Built using **Node.js** and hosted on a scalable environment (Render).

#### 2. Instructor Backend

- Implemented with **Node.js** and the **Firebase Admin SDK**.
- Responsible for receiving processed results and publishing them back to Firestore.
- Uses `firebase.js` to initialize Firestore connections and `index.js` as the main server script.

#### 3. Execution and Evaluation (Go Jupyter Converter + Python Runner)

The evaluation pipeline is managed by the **Go Jupyter Converter** module, which integrates with Python execution scripts.

#### Components:

- `parser_checker.go`, `dataset_rewriter.go`: Handle notebook parsing and dataset path adjustments.
- `run_code.py`: Executes submitted notebooks, captures output, and manages errors.
- `python_checker.go`: Measures execution time of Python scripts.
- `connection.js`: Bridges notebook processing with Firestore, recording accuracy results.

The pipeline converts uploaded notebooks into executable code, runs them in a controlled environment, and publishes results back to Firestore.

#### 4. Plagiarism Detection Service

- Implemented in Go (`PlagCheck/main.go`).

- Uses the `go-diff/diffmatchpatch` library to compare source files against a repository of submissions.
- Returns a plagiarism score as JSON, allowing instructors to detect potential academic dishonesty.

## Data Layer

The system leverages **Firestore** as its primary datastore.

### Key uses:

- Stores student submissions and metadata (author, commit hash, timestamp).
- Maintains execution reports and evaluation logs.
- Records plagiarism scores alongside execution outcomes.

Because Firestore is a real-time database, updates made by backend services are immediately reflected in the instructor dashboard without requiring manual refresh.

## Configuration & Quality Controls

- **Configuration Management:** Some backends rely on hardcoded paths (e.g., Python interpreter in `python_checker.go`), while others use structured config files (`config.go`).
- **Deployment:**
  - Frontend: Vercel.
  - Backend: Combination of local servers and Render-hosted services.
- **Security:** Firebase authentication and Firestore access rules must be enforced to ensure data integrity and privacy.

## End-to-End Workflow

1. **Submission:** A student uploads code via the submission portal.
2. **API Handling:** The backend stores the submission and queues it for processing.
3. **Execution:** The Go Jupyter Converter + Python runner executes the code, logs outputs, and calculates accuracy.
4. **Plagiarism Check:** The submission is compared against other files for similarity scoring.
5. **Storage:** Results (execution + plagiarism scores) are written into Firestore.
6. **Dashboard:** The instructor dashboard automatically reflects the results in real time.

## Future Enhancements

At present, the system is designed to execute only locally drafted code and does not support code submissions created and executed on external platforms such as Google Colab. Additionally, since the compiler operates in a local environment, the overall performance is dependent on the instructor's hardware specifications. Systems with limited processing power or memory may experience delays or reduced efficiency when handling multiple or computationally intensive submissions.

