**Project Explanation**

**Project Overview**

This project implements an API to retrieve the status of a video translation job. It also includes functionality to simulate real-world scenarios like handling errors during translation.

**Technologies Used**

* **Language:** Java 17
* **Framework:** Spring Boot

**Assumptions and Workflow**

**Key Assumptions**

1. **Video Translation API:**  
   Before creating the /status/{jobId} API, an API /translateVideo was developed to initiate a new translation job. When this API is called:
   * A new job is added to the pool.
   * The job runs for a configurable amount of time (set in the application.properties file in seconds). (By default 50 seconds)
2. **Job Status API:**  
   The /status/{jobId} API can be used to retrieve the status of a specific job using its jobId. The response includes:
   * **pending**: If the job is still in progress.
   * **completed**: If the job translation is finished.
   * **error**: If an error occurred during the translation.

**Simulating Error Use Case**

* An API /delete/{jobId} was added to simulate error scenarios.
* If a video is deleted during its translation phase, the job will transition to an **error** status.

**Cost Considerations for the GET API**

To optimize cost and reduce redundant calls, the following logic is suggested for the front-end:

**Implementation Approaches**

1. **Button with Delay Mechanism:**
   * Add a "Fetch Status" button for jobs.
   * Disable the button for 10 seconds after being clicked to prevent frequent calls.
   * Automatically update the status if the user doesn’t press the button for 20 seconds.
2. **Display Cost Messages:**
   * Show a warning or cost message on the screen if the "Fetch Status" button is clicked too frequently.
3. **Automated Workflow:**
   * Once the user clicks the "Fetch Status" button, automate subsequent status updates at regular intervals and display them on the screen (can be added as an enhancement).

**Logging**

Added logging information to config file, you can change the logging level to INFO, DEBUG, ERROR as per the use case. By Default, I am setting it to INFO level.

**Test Cases**

Wrote 5 Integration test cases that is simulating the API flows, errors use cases in **HeygenApiApplicationTests.java** file.

**How to Run**

1. Download the project ZIP file from the GitHub repository.
2. Import the project into any Java environment such as IntelliJ IDEA or Spring Tool Suite.
3. Run the Spring Boot application to start the server.

**Enhancements**

**Proposed Improvements**

* **Database Integration:**  
  Jobs and their statuses can be stored in a database. The application can periodically update job statuses (e.g., every 10 seconds).

**Code Explanation**

A Separate document to explain code classes is attached with name - Documentation for Code Classes.docx, please go through it for more information.

**Further Explanation**

If needed, we can have a short call to discuss this application in detail.