AI Grading Assistant

Components

- PostgreSQL table **ai_queue**, to store assignments queued for processing by the OpenAI API.
- PHP endpoint **POST next_v2_endpoint.php/response/submit**, to submit items for insertion into the ai_queue table.
- React app includes **aiFeedbackItemIds** array for items to be processed by the Grading Assistant. (At this time, the array contains just a single item.)
- PHP class **GenAI**, to perform many static functions:
    - Extract submission data (e.g., student prompt, student response) given the IDs in the ai_queue table (assignment ID, student ID, item ID, &c.).
    - Construct prompt to pass to OpenAI. The prompt is built from standard text maintained by non-tech staff, rubric data, student prompt, student response.
    - Perform OpenAI API call, passing the prompt and receiving the OpenAI response.
    - Extracting the OpenAI response, and passing it to the existing grading class, to be stored as AI feedback data in the Mongo **studentgradeobjects** collection.
- PHP scripts to query the ai_queue table for unprocessed submissions and pass them to the **generate_feedback.php** script for processing. If a record fails processing three times, it is no longer picked up.

Student Submission

The AI Grading Assistant operates on specific problem types (Long Essay, Document-based), each of which is identified using a Rubric ID. This ID is represented with a constant in the code.

When the student submits a response for a problem of one of these types, the problem ID is inserted into the standard submission API call, in an array called **aiFeedbackItemIds**. This results in a record being added to the **ai_queue** PostgreSQL table for processing by the Grading Assistant. Processing is triggered by a cronjob that fires every five minutes.

Cron Job Processing

The cronjob runs a PHP script, which begins by querying the **ai_queue** table for records that aren't marked as having been successfully processed for the Grading Assistant, and that haven't been attempted more than three times. Each row returned by the query is passed to the script **generate_feedback.php**.

The generate_feedback.php script takes several command line arguments, which are needed to identify the item the student submitted:
Course ID, Assignment ID, Task ID, User ID, Item ID

Additionally, the submission ID and AI Feedback ID are needed. These are obtained by calling the **GenAI::queryGradeData** static method. This method is passed the Course ID, Assignment ID, User ID, and Task ID, which are in turn used to gather further data from existing API handlers.

The queryGradeData method begins by calling **RestLessonPlans::queryLessonPlans** to get the Group ID for the querying student grade data. Then, it calls **RestStudentGrades::queryGradeData** to get the submission and feedback IDs. There are four specific IDs: studentGradeId, submissionId, aiFeedbackId, feedbackId.

With this information in place, processing proceeds to generating prompts for the Open AI calls and then retrieving the generated feedback for the student submission.

There are two types of problems to which the Grading Assistant applies: Long Essay, and Document-based. The first step is to determine which of these the present problem is. The generate_feedback.php script calls **GenAI::isLongEssay** with the Item ID.

The isLongEssay static method calls **RestProblems::get_problems**, passing in the Item ID ($problemId). The problem type is determined by inspecting the problem's "rubricId" value. This value is the means of identifying long essay problems, document-based problems, and others. The value for long essay problems is stored in the constant RUBRIC_ID_LONG_ESSAY. If the rubricId matches this value, the isLongEssay method returns true.

Next, generate_feedback.php calls the static method **GenAI:set_to_inprocess**. This simply updates the **ai_queue** record, setting the "in_process" field to true. As long as this is set, the record cannot be included in further cron job queries. This is a necessary precaution, in case certain records are still being processed the next time the cronjob is fired.

Now, generate_feedback.php generates the AI prompt for the submission. The prompt for document-based problems is different from the one for long essay problems, so there are two different processing blocks.

For long essay problems, **GenAI::getPromptDataForLongEssay** is called. It returns the associative array **{ studentPrompt, studentInput }**. The **student prompt** is the specific question the student selected to answer. The **student input** is the essay-format user entered in response to the selected question. These values are obtained by querying the student grade data.

The getPromptDataForLongEssay method calls queryGradeData with the Task ID, Student ID, and Course ID. The result is the submissions the student made for this task / problem. These

submissions are stored in an array of “submissionData”. The “prompt” and “response” properties from the last object in this array are collected and returned.

Next, **GenAI::isPromptInputSupplied** is called with the { studentPrompt, studentInput } associative array returned from getPromptDataForLongEssay. This is to confirm that student input is available before calling the OpenAI API. The isPromptInputSupplied method returns a boolean value.

If isPromptInputSupplied returns **true**, the method GenAI->longEssay() is called. This is what performs the actual API call to OpenAI. This is where the AI model is specified, as well as the key. The API path is **/v1/chat/completions**, and the model is **gpt-4o**.

When the API call returns, the content is extracted from the result using the **GenAI::parseOpenAIResponse**. This is needed because the desired content is stored below the surface of the API response payload: [‘choices’][0][‘message’][‘content’]. Moreover, the content may contain extraneous character sequences (“```json\n”, “```”), which if not removed will cause a JSON parsing error.

Once the AI-generated feedback is received, the method **analyzeStrengthsWeaknesses** is called to perform a further analysis to determine strengths and weaknesses of the student’s input. This again calls the OpenAI API, this time including the AI feedback that was just generated.

When the generated feedback and the strengths and weaknesses analysis have been collected, the results are returned to the **generate_feedback.php** script. These results comprise two specific compilations of data: **prompt_for_openai** and **openai_feedback**. The prompt_for_openai for long essay questions includes the student-selected prompt, the student input, and the grading rubric. This grading rubric is read from an external text file maintained by someone having to do with content and who isn’t a programmer. The openai_feedback is the AI-generated feedback and strengths and weaknesses analysis.

The last step for long essay questions is to validate the results by calling **GenAI::checkValidResult**. This function ensures that student prompt and input are present, there was no error generated by the OpenAI call, and that the submission ID is valid.

If checkValidResult passes, generate_feedback.php calls **GenAI::saveAiFeedback** to add the Grading Assistant feedback to the student’s submission. Finally, it calls **Gen::set_to_processed** to update the ai_queue table to mark the record as processed.

At this point, when the teacher selects the student’s assignment for review and grading, the AI-generated feedback will be displayed, facilitating the teacher’s human-generated feedback.