1. Upload Images of student tests
2. State # of pages per student
3. Upload images of answer key (let me know how to make these into URL)
4. Copy paste the question list:

[

{ "question\_id": "1", "max\_mark": 1 },

{ "question\_id": "2", "max\_mark": 1 },

{ "question\_id": "3", "max\_mark": 1 },

{ "question\_id": "4", "max\_mark": 1 },

{ "question\_id": "5", "max\_mark": 1 },

]

1. I put the human graded list of questions not given 100%:

[

{ "question\_id": "3”,

{ "question\_id": "5",},

]

1. Which AI models to send to (number of times)
2. A results page that shows by question:
   1. GROUP CLAUDE: Try 1, Try 2…
   2. GROUP 5 mini: Try 1, Try 2….
3. Statistics:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Try 1 | 2 | 3 | Avg. |
| Claude | 2 | 3 | 4 | 3 |
| GPT 5 |  |  |  |  |

H: 3, 5

AI: 2, 3, 6

3 discrepencies

Open R API key: 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

## Dmitry’s comments:

API application is written using Django, and it is responsible for handling assessment-related logic, such as uploading pdfs, storing assessment settings, and tracking the upload status

AI application is written using FastAPI, and is responsible for communications with LLM (questions list generation, grading).

Some of the example files below will contain Django or FastAPI related logic. All this can be converted to a single style using AI, I provide these files just to give the general idea of how the upload+grading flow works.

#### Upload images of student tests

We upload not images, but PDFs. However, I believe this step can be omitted and done as the last one if there still will be a dev time, as for testing purposes we can upload images.

On FE we uploda the file using [this](https://docs.dhtmlx.com/vault/initialize.html) library, but if there is no requirements on the way this upload flow looks, then cursor can easily generate the input form for pdf files (@simple-file-upload.html).

Then this pdf file is uploaded using form data to server. Server is written using Django + DRF, so we accept file as rest\_framework.serializers.FileField(required=True). The file is validated, but for testing purposes we can skip those validations.

After file is uploaded, we are saving it to a system temp folder and use the [pdf2image](https://pypi.org/project/pdf2image/) service. Note that it uses poppler-utils package under the hood, so you would need to install this one as well (the documentations contains installation steps for mac/linux).

Example file (@file-upload.py). Note that in the end of a post method I added the example of how we save the images (using django models + S3 as a storage). If you will build withotu django, this can be done using library boto3.

#### State # of pages per student

On FE this could be a simple <input type=”number”/>.

On BE simply save the number to DB. We have a table in DB that stores upload-related settings (let’s call it UploadSettings), such as assessment\_id, pages\_per\_student, feedback\_lenght, etc.. Send this number to LLM later.

#### Upload images of answer key

Same logic as with uploading student answers. You can save images to a separate table, or create a flag in an existing table (UploadedImages), like is\_correct\_answer\_image: bool. So when you save the image, you can later extract the field value and separate which images belong to students, and which ones are for correct answers.

#### Store upload settings

Same as in # of pages per students.

On FE can build a simple form with radio buttons, and then send a choice to BE.

On BE the selected settings can be saved in UploadSettings table. Then, depending on the values of these settings, a prompt to AI can be selected.

#### Send request to LLM

I previously shared the code that is responsible for preparing the data for request for LLM.

From API module we send the following data to AI module: [schema](https://drive.google.com/file/d/1O3XYc4ae49ZfX64EnLfvyn5whuGK_BYX/view?usp=drive_link).  
  
from pydantic import BaseModel

class ImageSchema(BaseModel):

id: int

page\_number: int

aws\_path: str

class QuestionItemSchema(BaseModel):

question\_number: str

max\_mark: float

class ImagePromptWithImagesSchema(BaseModel):

external\_assessment\_id: int

external\_assessment\_settings\_id: int

external\_assessment\_file\_id: int

pages\_per\_student: int

prompt: str | None = None

images: list[ImageSchema]

model\_assessment: list[str] | None = None

question\_list: list[QuestionItemSchema]

class Config:

"""Prevents automatic aliasing of conflicting field names"""

protected\_namespaces = ('protected\_',)

from\_attributes = True

* external\_assessment\_id is a identifier of an assessment.
* external\_assessment\_settings\_id is identifier of the settings object. Need to have it to correctly create cropped images. Every setting object represents an upload (same assessment may have multiple uploads).
* external\_assessment\_file\_id is an identifier of the uploaded file.

Right after the request is accepted, the method create\_prompt of file [external\_assessment\_cases](https://drive.google.com/file/d/14J6YlhM2FDiwsFpaFBDusgAnkUZCmv5E/view?usp=sharing) is called. (@external\_assessment\_cases.py)

* It saves the accepted data to DB. This is needed because different requests are made as a separate processes, and sharing lots of data between those can be an overhead, so we put data to DB and when batch process is started – it extracts the required data.
* It transforms the data to a format that is sent to LLM.
* It calls the image prompt service to start the file-related operations.

Image prompt service file can be found [here](https://drive.google.com/file/d/1DTJtIbCTHpAGS-t1NtZY63v8TbJyRDKp/view?usp=sharing) (@image\_prompt\_service(1).py). You should’ve already seen it, everything is the same. We split code to batches and send every batch as preprocess-process chain. I think for testing purposes you do not need the token calculation logic. All batches are executed in parallel. File with batch tasks can be found [here](https://drive.google.com/file/d/1DTJtIbCTHpAGS-t1NtZY63v8TbJyRDKp/view?usp=sharing).

Instead of calling chat\_gpt.ask\_chatgpt you can insert your own logic that will make a request to OpenRouter.

#### Processing AI response

Once responses for all batches are received and saved in DB (again, to avoid passing big data between the processes), we [handle](https://drive.google.com/file/d/1DTJtIbCTHpAGS-t1NtZY63v8TbJyRDKp/view?usp=sharing) a response. Response schema described [here](https://drive.google.com/file/d/1TGgSq-hm6caW7iQur67d3bMJoZzffAP7/view?usp=drive_link).  
  
from pydantic import BaseModel

class ExternalAssessmentStudentAnswerSchema(BaseModel):

question\_number: str

mark: float

feedback: str

class ExternalAssessmentStudentResultSchema(BaseModel):

first\_name: str

last\_name: str

answers: list[ExternalAssessmentStudentAnswerSchema]

class GPTGradingResultResponseSchema(BaseModel):

result: list[ExternalAssessmentStudentResultSchema]

As was mentioned in one of the emails, AI module has it’s own status, separate from the status that is stored on API module. But again, I don’t think you need multiple modules.

#### Creating results

After this whole process goes the student results creation. I believe for testing purposes it will be enough to save the data as is, without adding too much logic. However, I do not know what you guys agreed upon, so if you need some other details – let me know as well.

Those are the data levels that we have:

* Assessment.
* Upload.
* Student.
* Answer.

For each level we have a table in DB. So once you save the results, to DB, you can display all student results for an assessment.