Brief Product description

Our web application serves as a comprehensive tool for creating and managing quizzes in a classroom environment. The process of generating quiz questions leverages large language models (LLM), drawing insights from the class textbook and the specific chapter covered in the class. Instructors have the flexibility of creating and editing quizzes, accordingly, ensuring a better experience for their students. Meanwhile, students can easily access and take quizzes within the system, with the added functionality of flagging inaccuracies in the generated questions.

Product functionalities

Sprint#1 Functionalities

- **Instructor US#10: Access Previously Generated Questions**

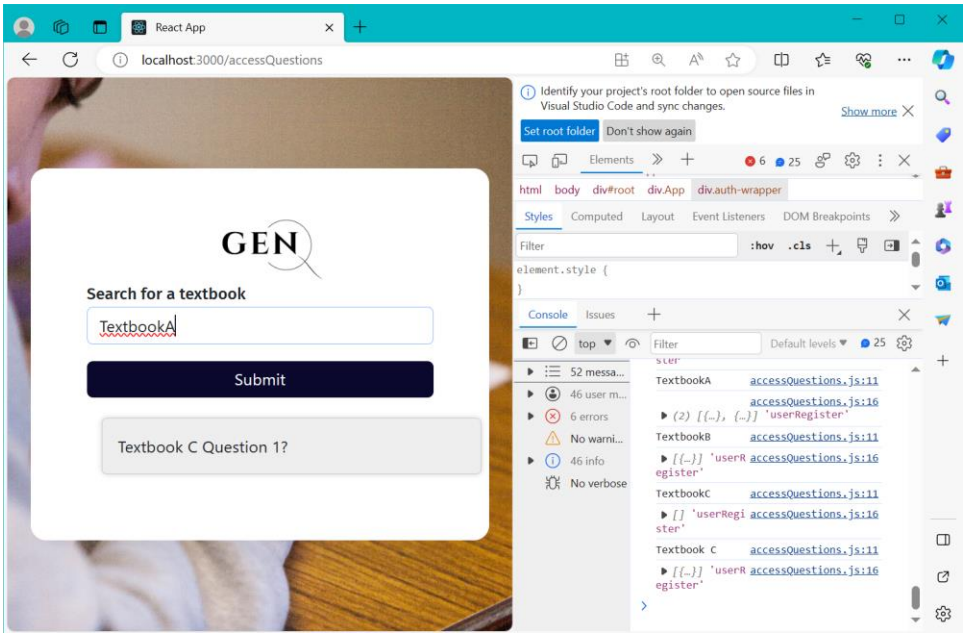


Image 1: Here, we see that we enter TextbookA as the name for the textbook to search in the search bar (disregard the text below the submit button as this is from a previous query).

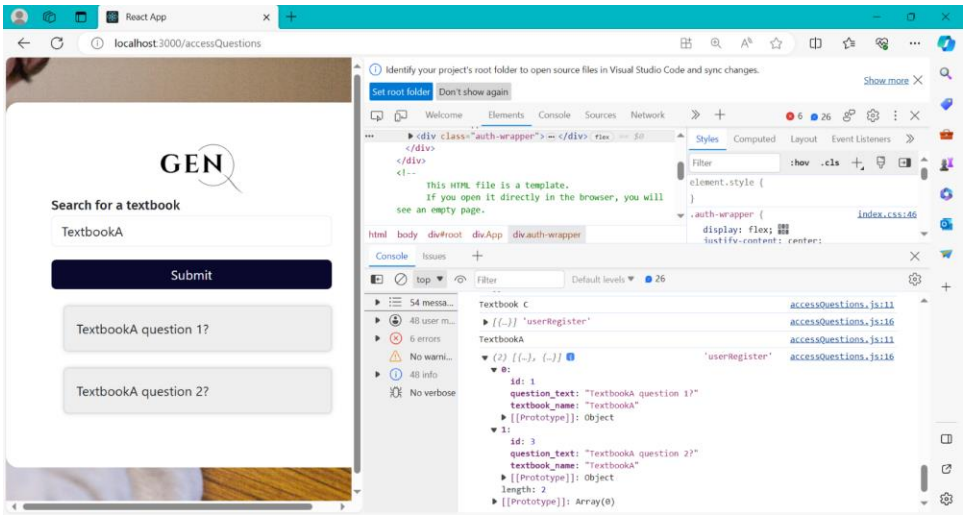


Image 2: When we hit submit, we see on the bottom right in the console that we log the response from the backend server, which is the questions associated with the textbook, and we also see that those questions (only their text) are shown below the submit button.

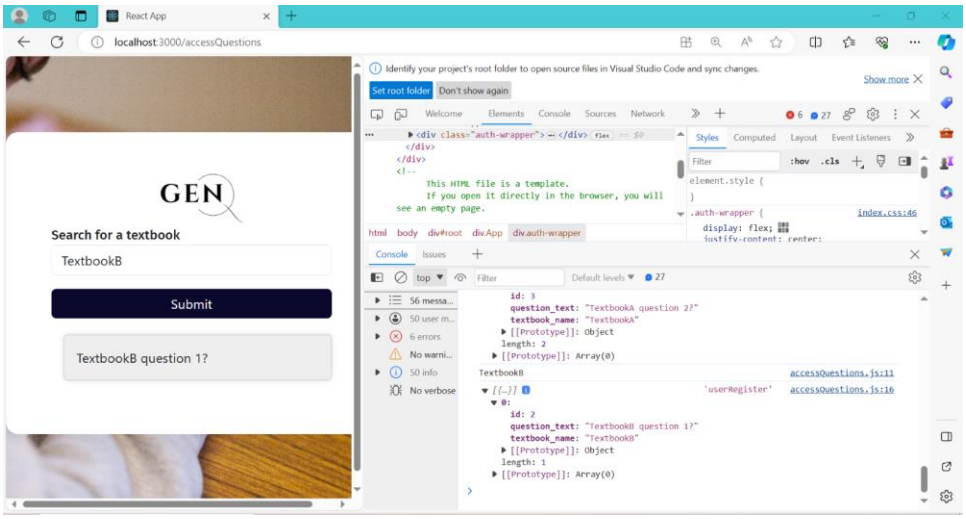


Image 3: Similarly, when we search for “TextbookB”, we get a response from the backend server with the appropriate questions (as shown in the bottom right corner of the console), and those questions are presented below the submit button.

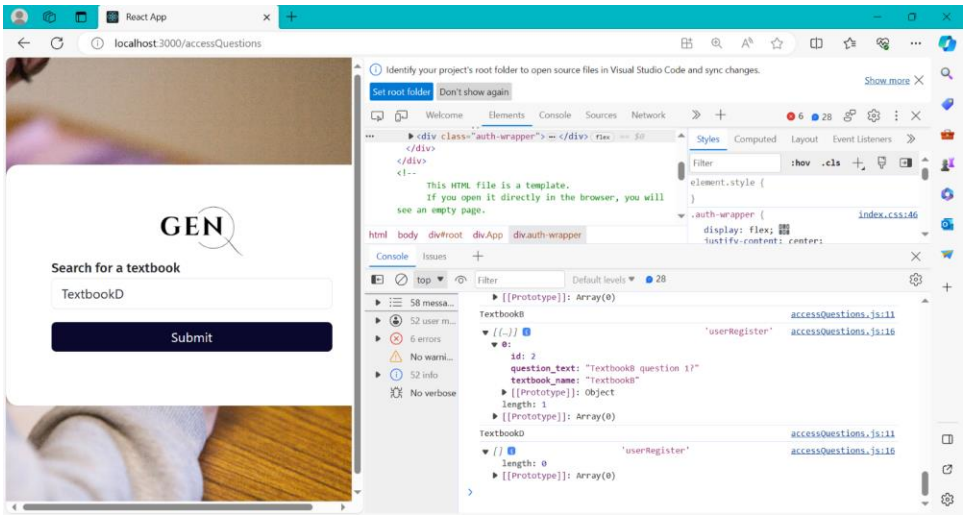


Image 4: Here, we see that when we search “TextbookD”, which is a textbook name not present in the database, we get an empty list of questions back from the backend server (as shown in the bottom right corner of the console). Thus, there are no questions present below the submit button.

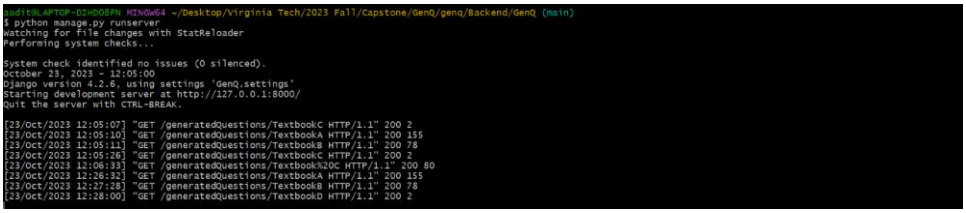


Image 5: Shown above is the log from the backend server that shows it is receiving the requests.

- **<u>Instructor US#3: Create various Quiz Versions</u>**

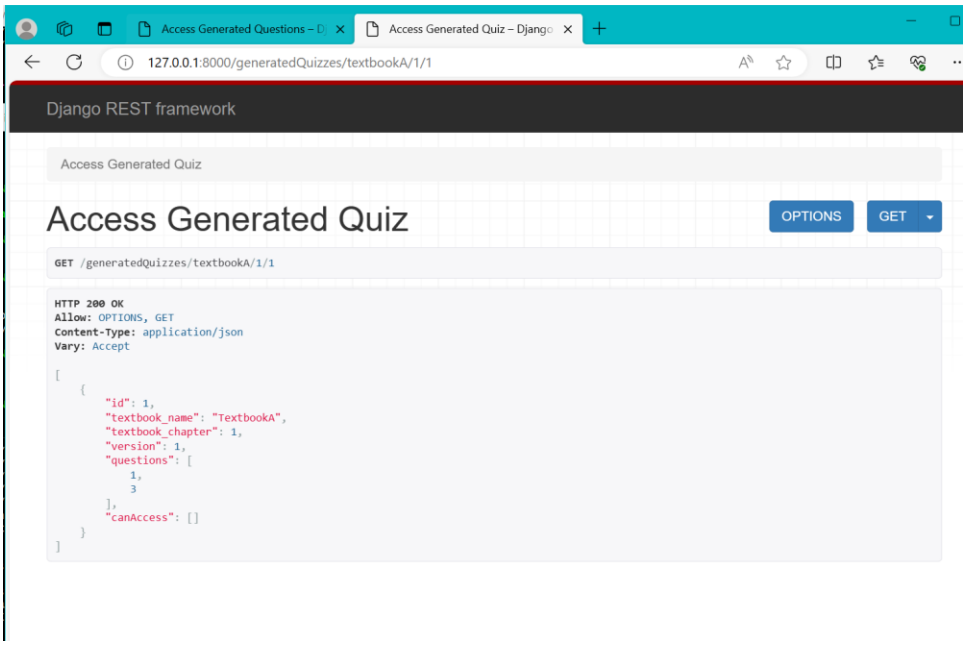


Image 1: The GET API for a quiz, given the textbook name, textbook chapter, and the version. Here we see that we get a response back from the backend with the ID for the questions associated with that quiz.

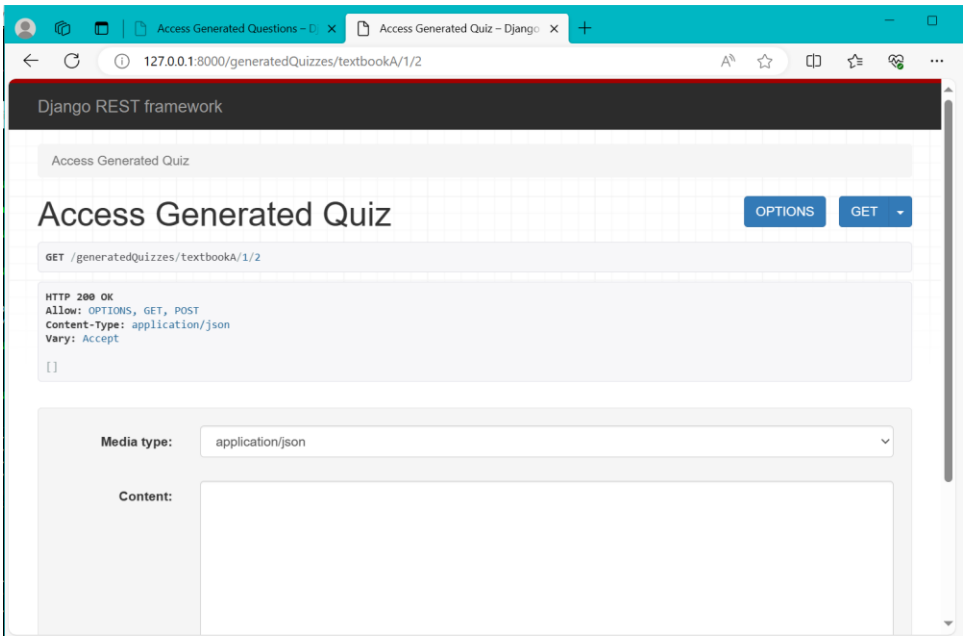


Image 2: Here, we see that when we call a GET request with the same textbook name, chapter, and the 2nd version (which doesn't exist), we get an empty list back from the backend server indicating this quiz does not exist

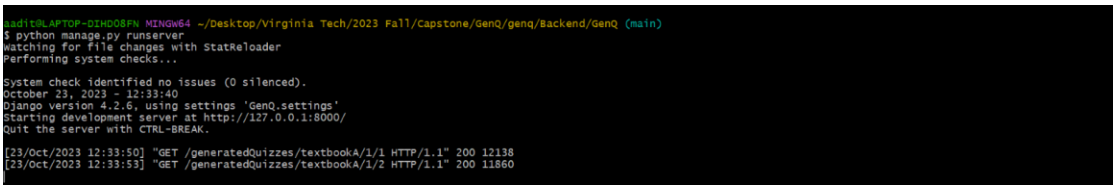


Image 3: Here, we see the log from the backend indicating that it is receiving the query from the Django API.

- **<u>Instructor US#2: Generate Question</u>**

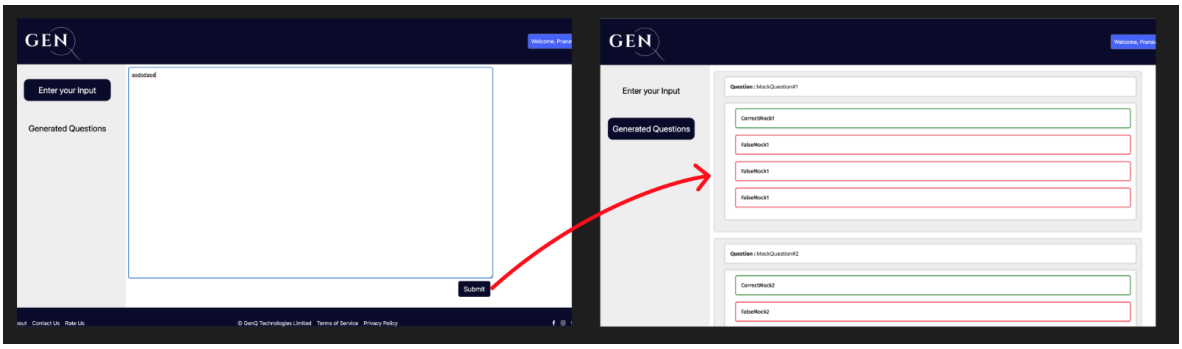


Image 1: After entering the generate questions page, the user would be promoted to enter the textbook information (will be changed in next sprint) after which they would submit it to start the generation process. Then the page would move to the generated questions page, the user (instructor) can read through all the generated questions and options.

Image 2: Once the user has finished looking over the generated questions, they could save the questions.

<u>Instructor US#1: Upload Textbook:</u>

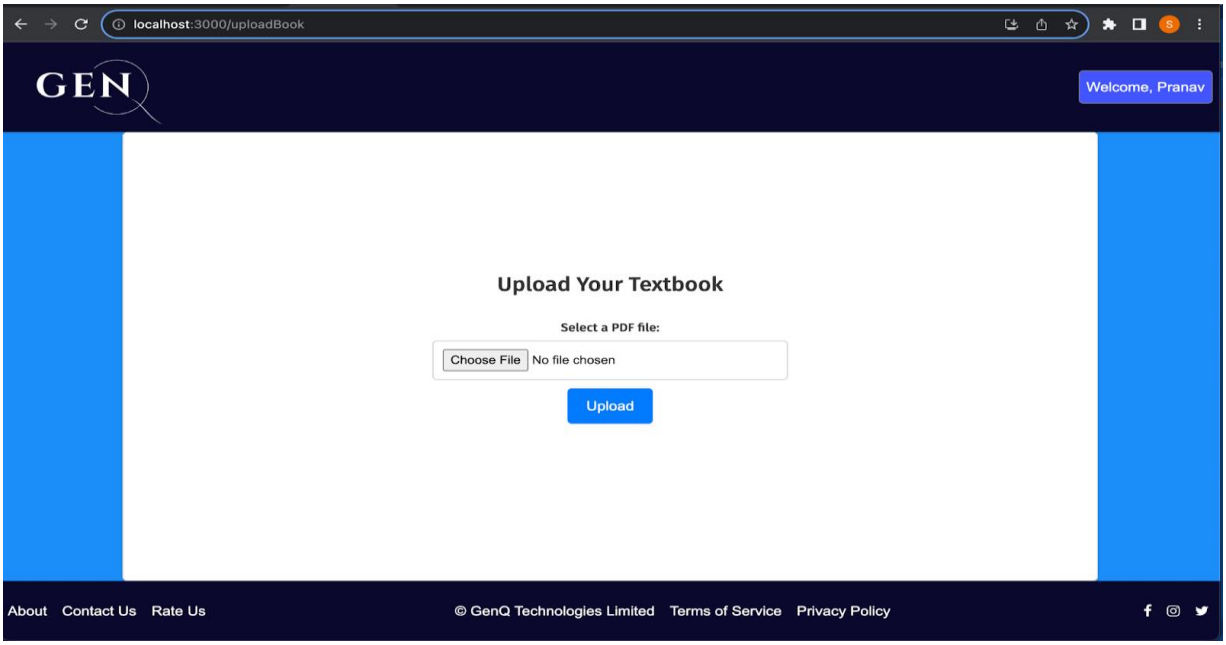


Image 1: Upon going to the uploadBook route, the prompt along with necessary buttons are displayed to upload the textbook. The ability to choose a pdf file of a textbook and then upload said textbook is given in the form of buttons.

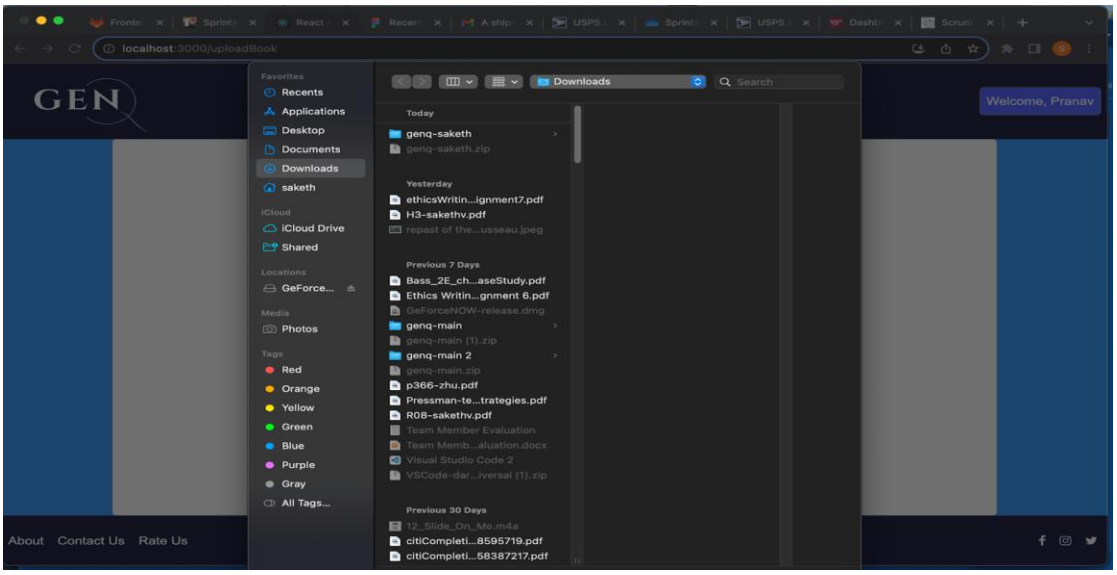


Image 2: This image represents the screen that is displayed when the Upload button is selected. The files that are available are pdf files and any other file type will not be available for the user to pick. A requirement for the system was to allow only pdf files for the textbook. Upon selecting confirm, the pdf will be sent to the server where if processed appropriately, will return a success message to the platform. Otherwise, a failure message will be displayed.

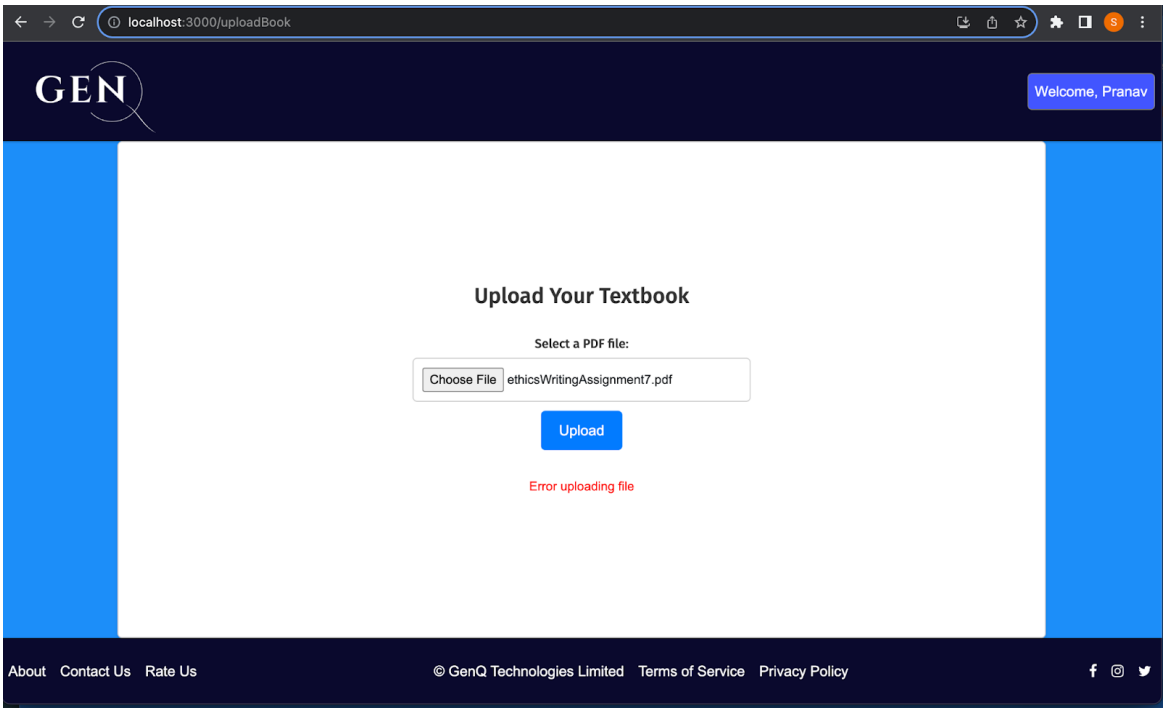


Image 3: Above is an error message being displayed on the platform in response to the server saying that there was a problem in the transmission of the pdf file from the user to the software. This provides a check in the system to ensure that all the data that is stored on the server is appropriate to the software.

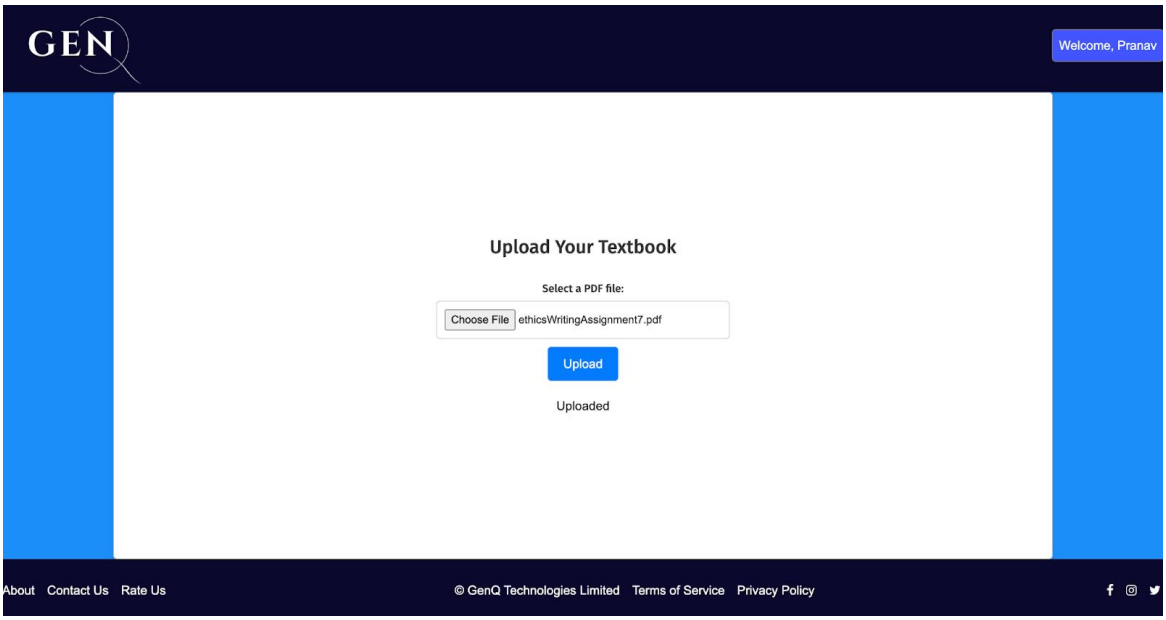
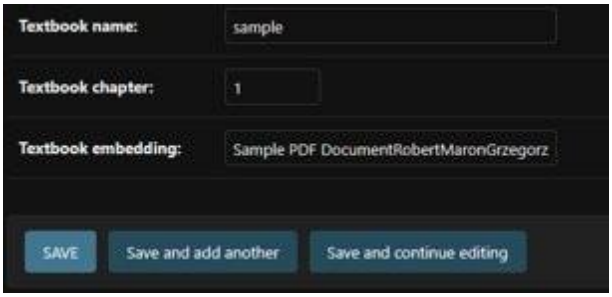


Image 4: Here is an example of a pdf document being successfully uploaded to the platform with a return message from the server stating “Uploaded” as a reassurance to the user. This completes a positive working use case of the upload textbook.

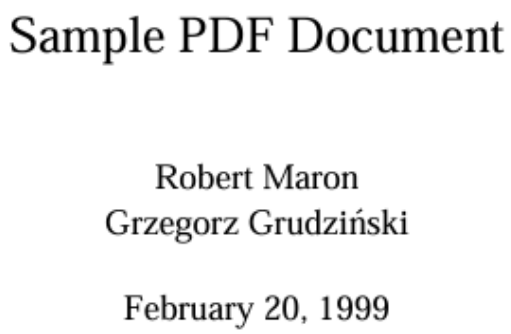
A Python JSON object with the textbook information is submitted through the API to the /upload/ endpoint.

```
{
        data : {
                'name': 'sample'
                'chapter': '1'
                }

        files : {
                'textbook': Python File Object
                }

}
```

When viewing the database, we can see a corresponding textbook object is created. The text from the file object is extracted and stored in a field.



We can see that the text in the database matches the text from the sample PDF shown below. One improvement would be to save the file server-side and store a path to the file in the database to improve performance.



- User Authentication

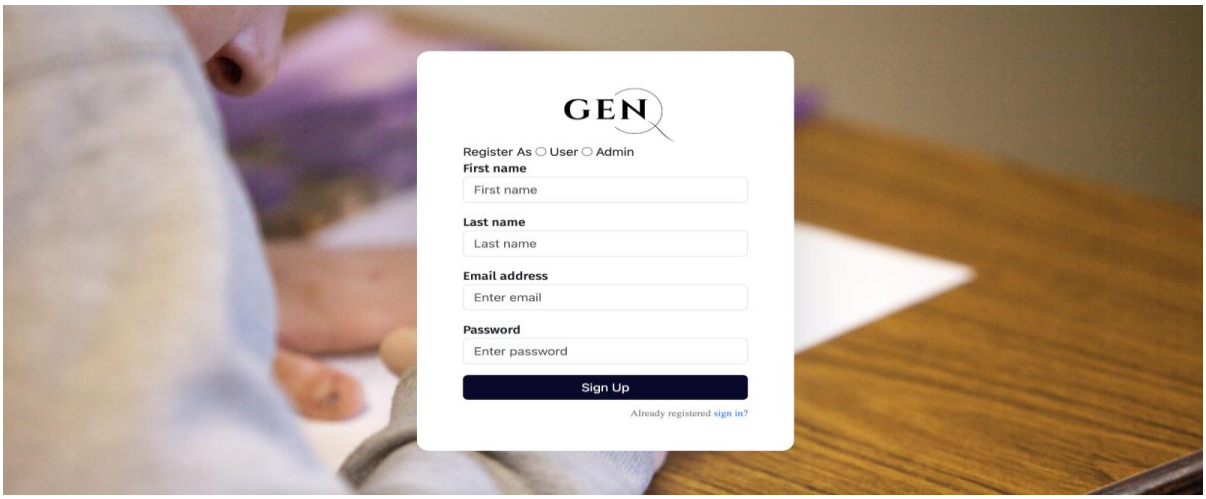


Image 1: This page is our User Registration Page. When the user is registering for the first time on the website, he needs to choose if he is an instructor or a student and enter other relevant information like Name, Email and Password. This data is stored in the database and will be used to authenticate the user during login.

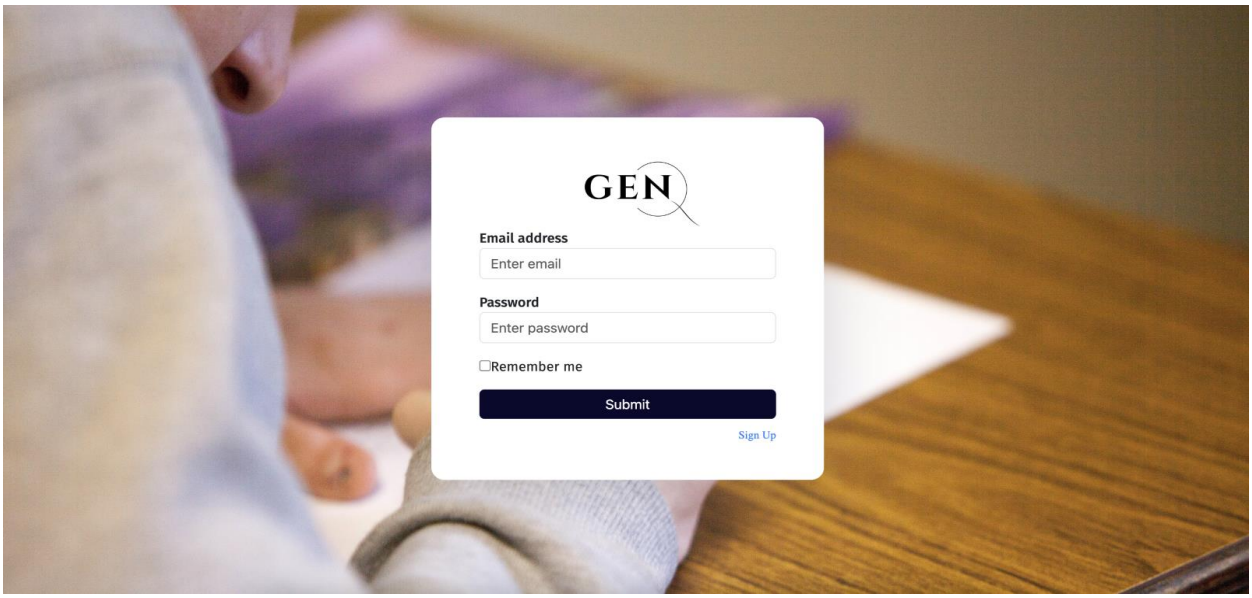


Image 2: This page is our User Sign-up Page. When the user needs to login on the website, he needs to enter data such as email and password. The data stored in the database during registration will be used to authenticate the user during login.

- **LLM API**

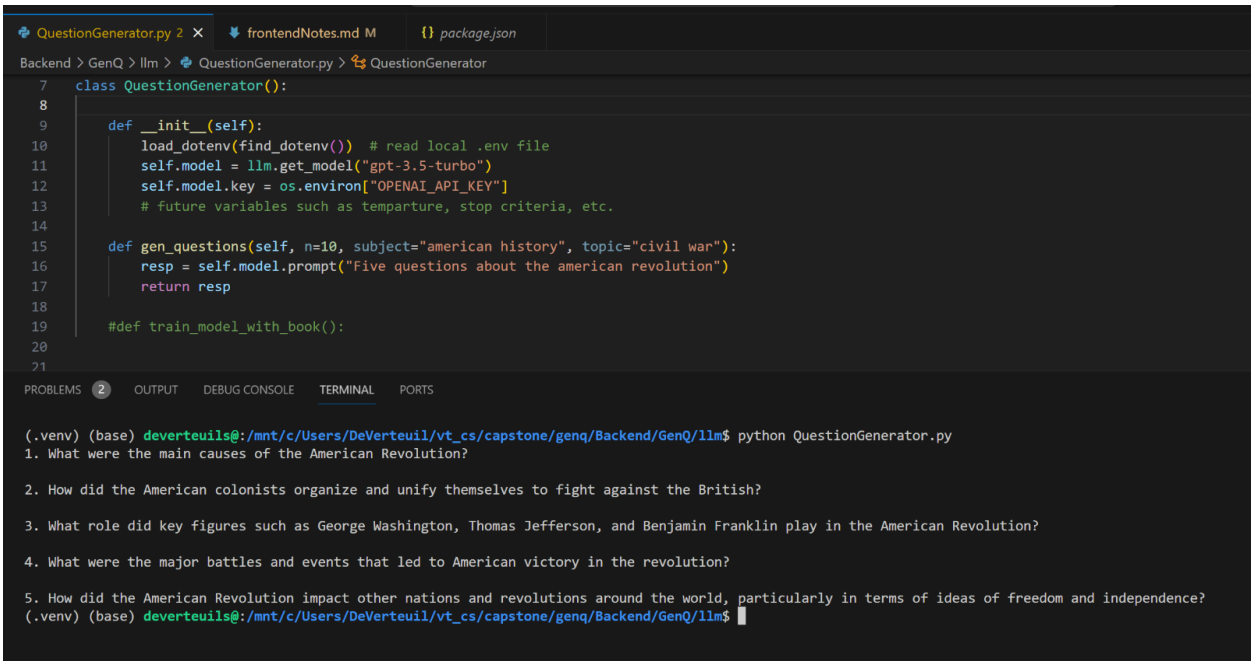


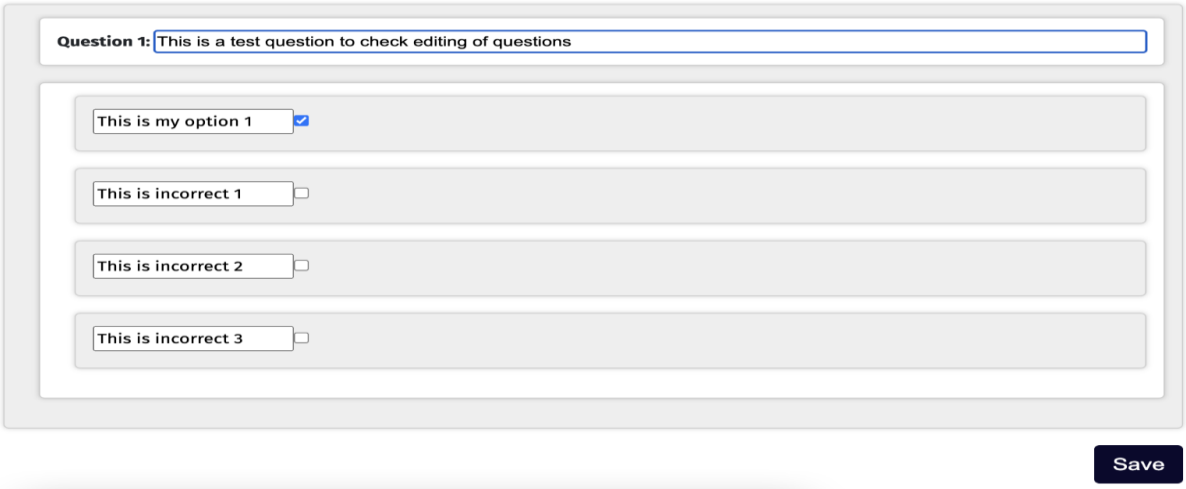
Image 1:

As can be seen in the screenshot above, a mock LLM has been created. At present, the user is not able to access these generated questions. The prompt is hard-coded, and the output is printed to the terminal for the time being. In the next sprint, we plan to integrate the LLM response with our front end. Additionally, we will allow for specific textbook based downstream training of the LLM so that the user can receive robust questions based on their desired input.

- Sprint#2 Functionalities

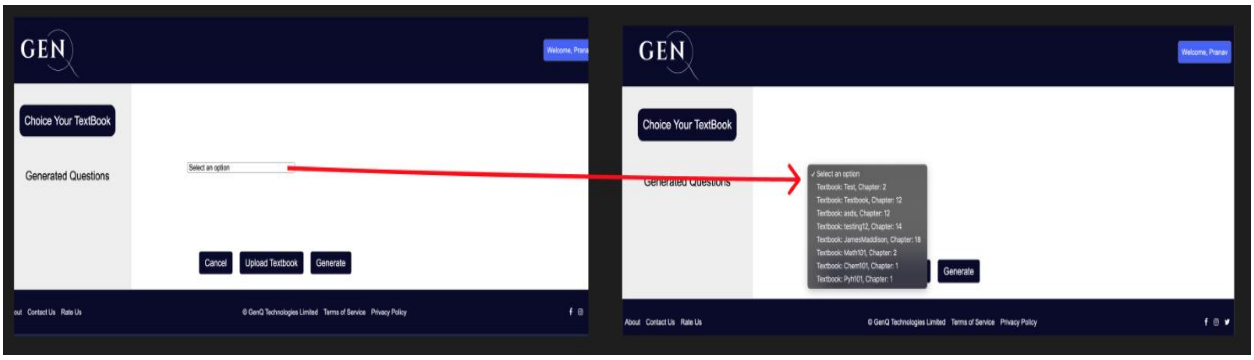
a. **Instructor US#6: Instructor Edit Generated Questions**

Once the questions have been generated, Instructor sees the questions and options in an editable format. Instructor has the option to edit the question and also the choices can be edited by the instructor. The figure below demonstrates the page which has generated questions.

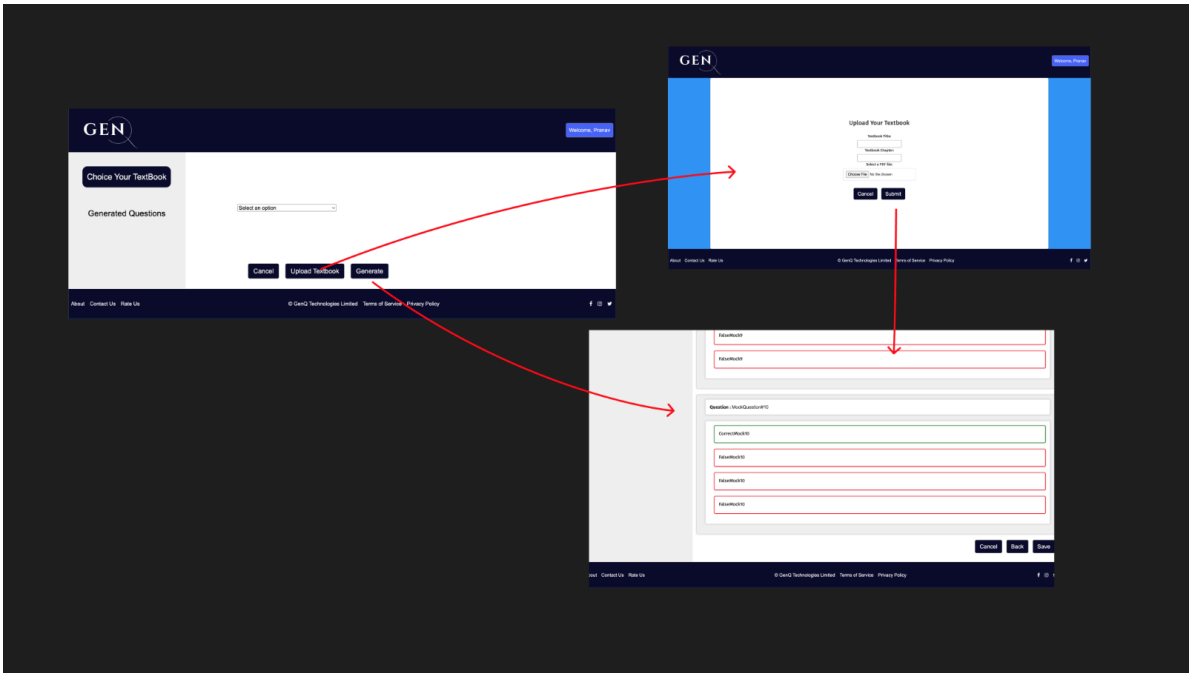


After Instructor edits the question and options and saves them, they are saved into the database and are now viewable as the edited questions and options.

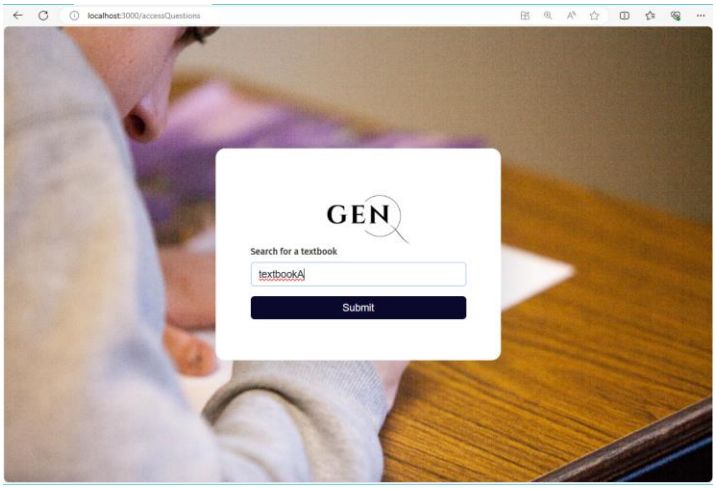
a. **General User Story: Integrating Generating Question with Uploaded textbook & Integrate LLM API with Model**
b. **A new process has been created to capture all the textbooks that exist in the backend, where the user has the choice of either uploading a new textbook or choosing an existing uploaded textbook. The figure below is an updated page with a list of textbooks for the instructor to choose from.**



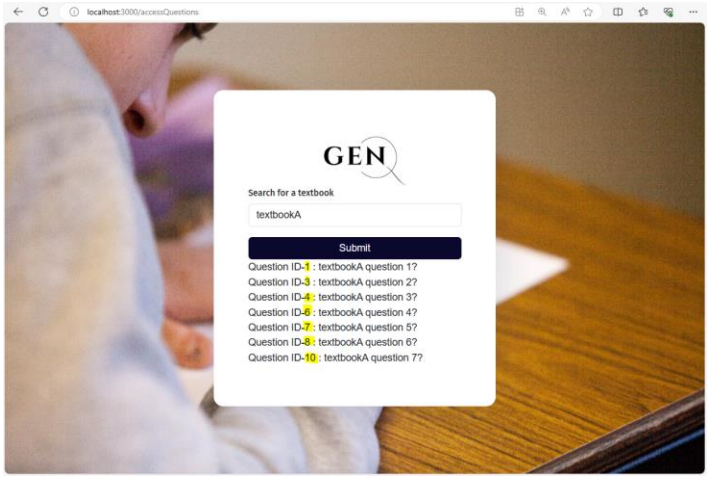
The other option is uploading a new textbook and instantly generating questions as shown below.



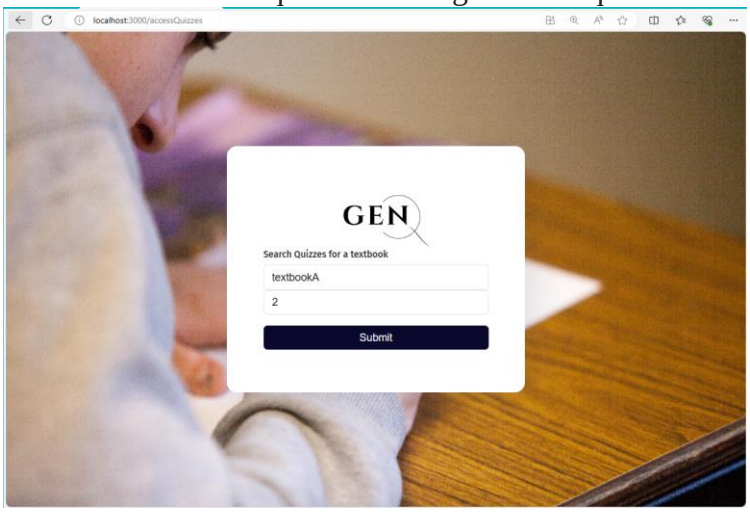
c.Instructor US#3: Instructor Quiz Versioning



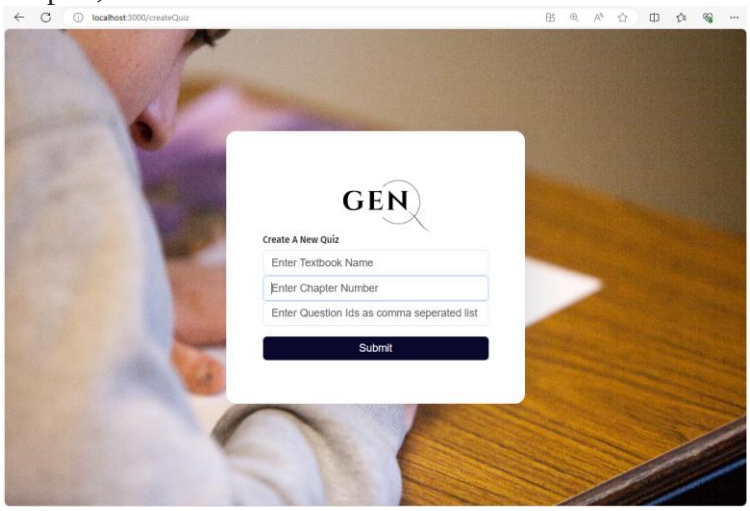
First, the instructor enters the textbook name to find all the questions available



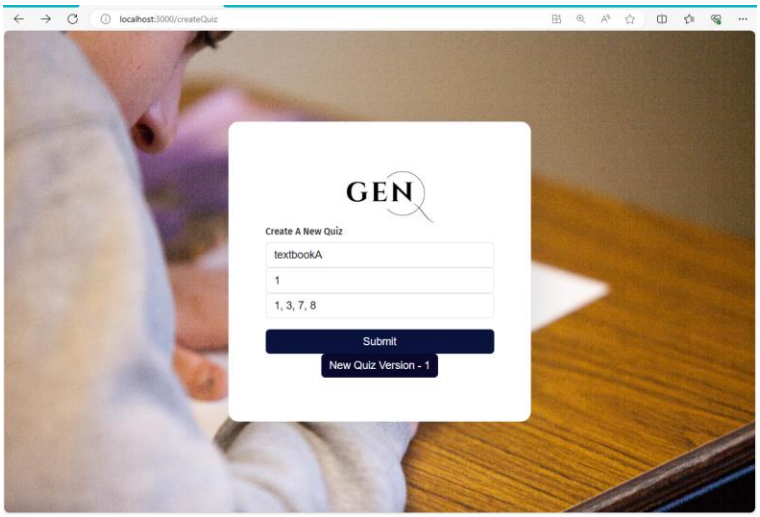
This returns a list of questions along with the question ID associated with that textbook



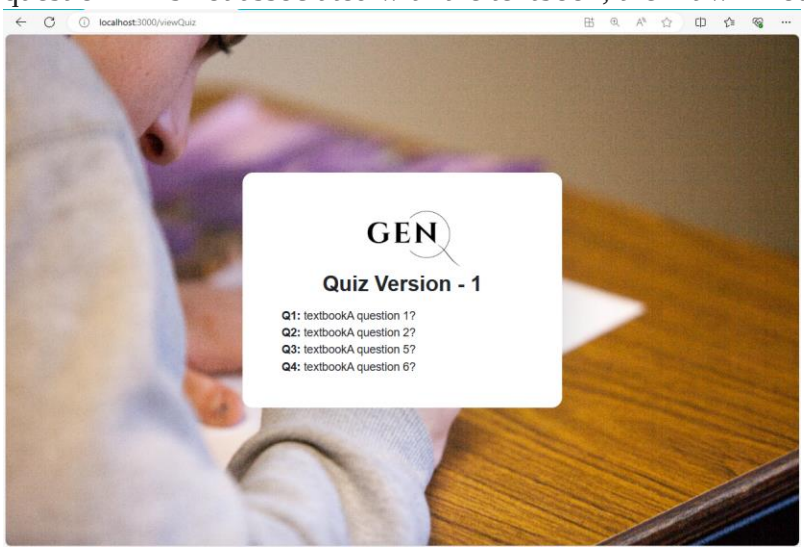
If we check in our database, we see that if we search for quizzes for any textbook name and chapter, we have no results.



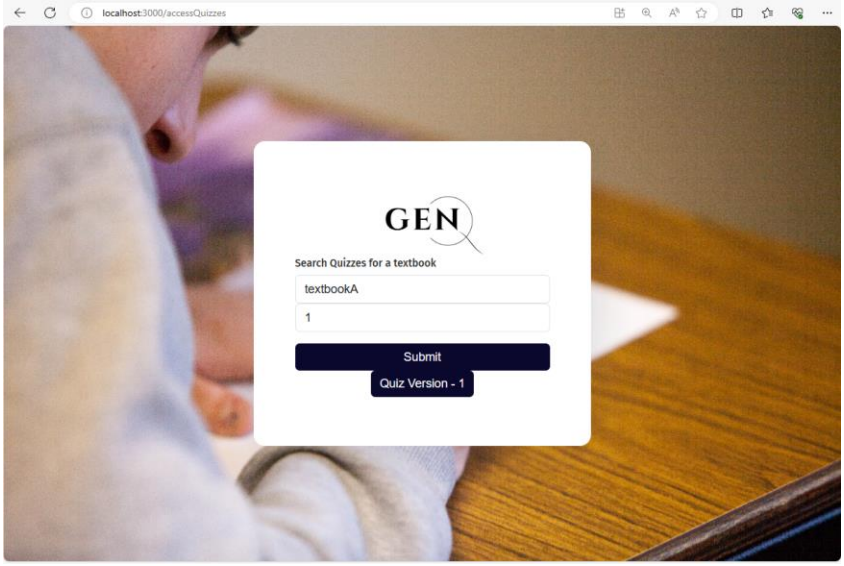
We then navigate to the “createQuiz” page, which provides the above form.



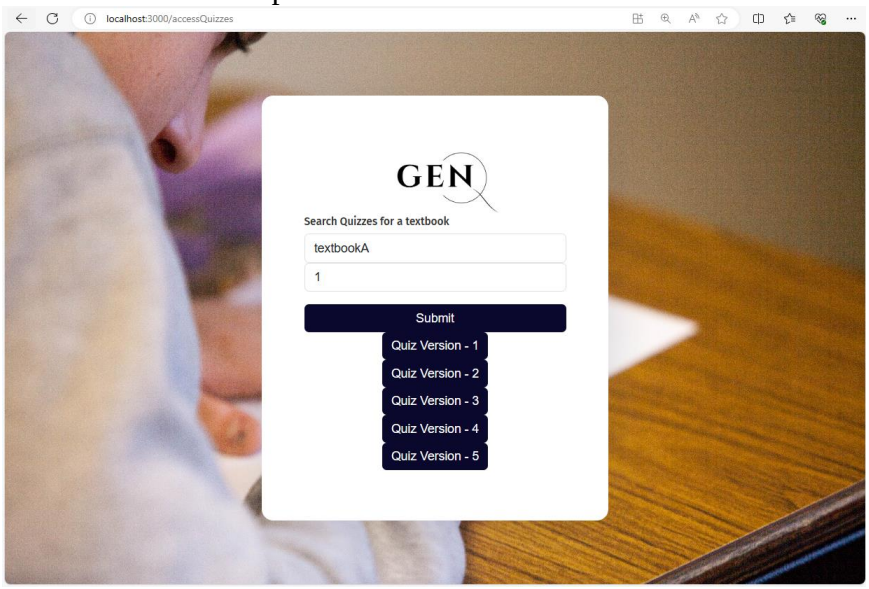
Once you enter the textbook name, chapter number, and the question IDs you want to include in the quiz, you hit submit, and get a link to the new Quiz. Note that as a safety check, if the question ID is not associated with the textbook, then it will not be add to the quiz.



Clicking on the button takes you to a new page which allows you to view the quiz. This includes the version, and the questions associated with it.



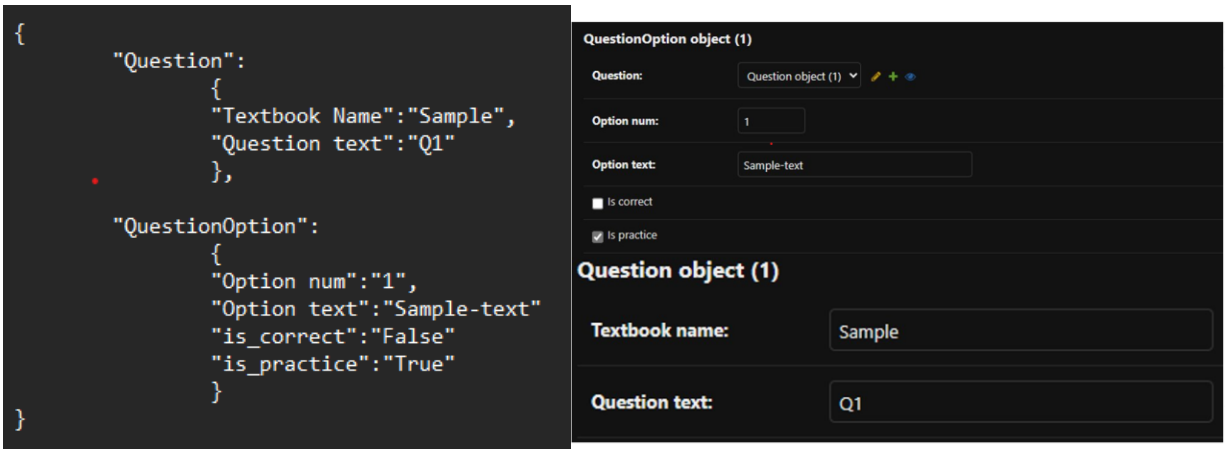
If we return back to the “accessQuizzes” page, and search for the textbook and chapter, that new version now shows up when we search.



We can create multiple versions of the quiz, and when we search for the textbook and chapter, they all show up as a link so that you can look at that quiz version.

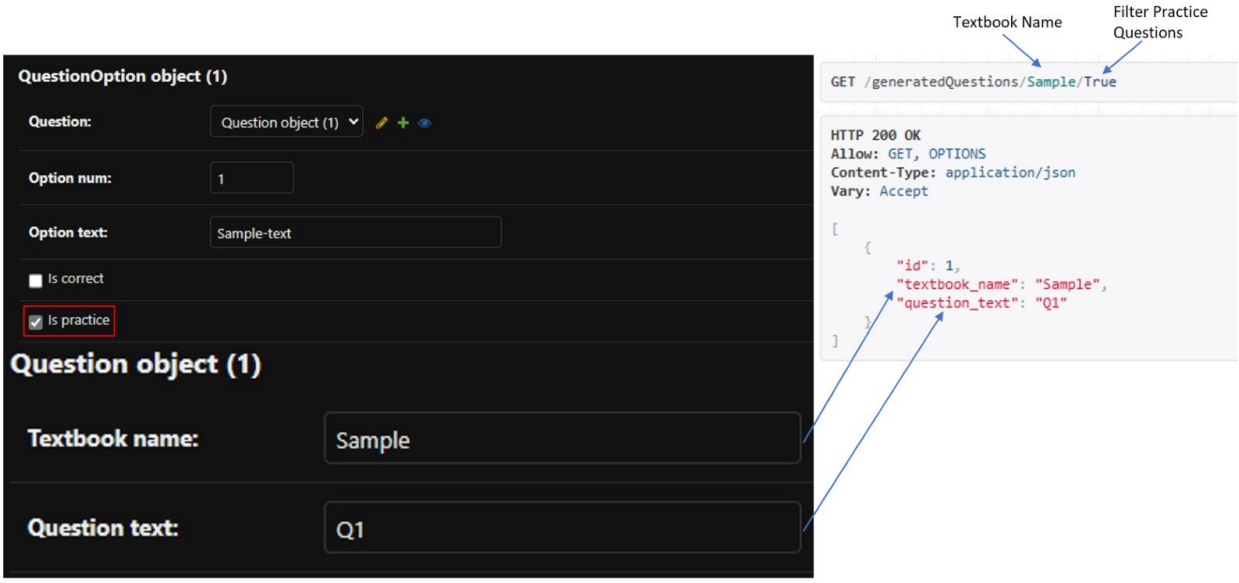
d. Instructor US#5: Instructor Choices Student Sample Questions

i. **First the instructor submits an API request to either create a new practice question or update and existing question and mark it as practice.**



ii. **(Left) Question data in Post request. (Right) Resulting question and option in the database.**

iii. **We can see in the database that the question is successfully marked as a practice question. The question fields match the information passed in the API request. Once the question is marked as practice, instructors can filter questions by textbook and if the question is marked as a practice question with an API request.**

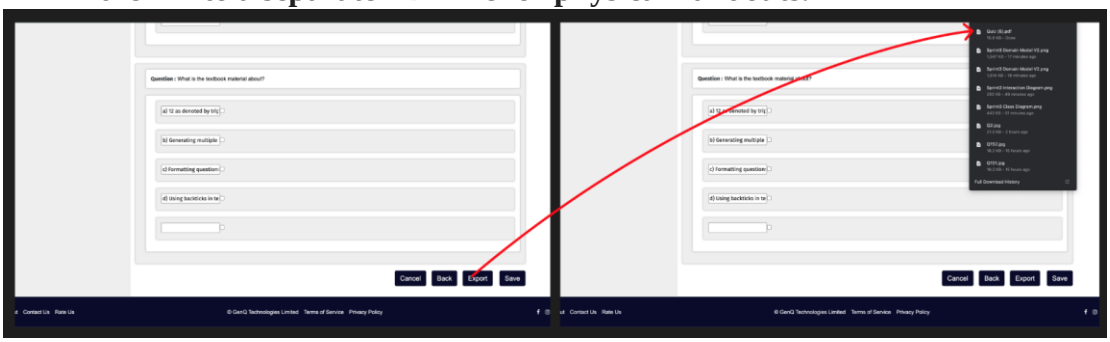


iv. **On the left we can see the question option and associated question which has the textbook "sample" and text "Q1". The question option marks the object as a practice field. On the right side the GET request passes in the desired textbook name and if to return practice questions. The correct question is returned from the API request.**

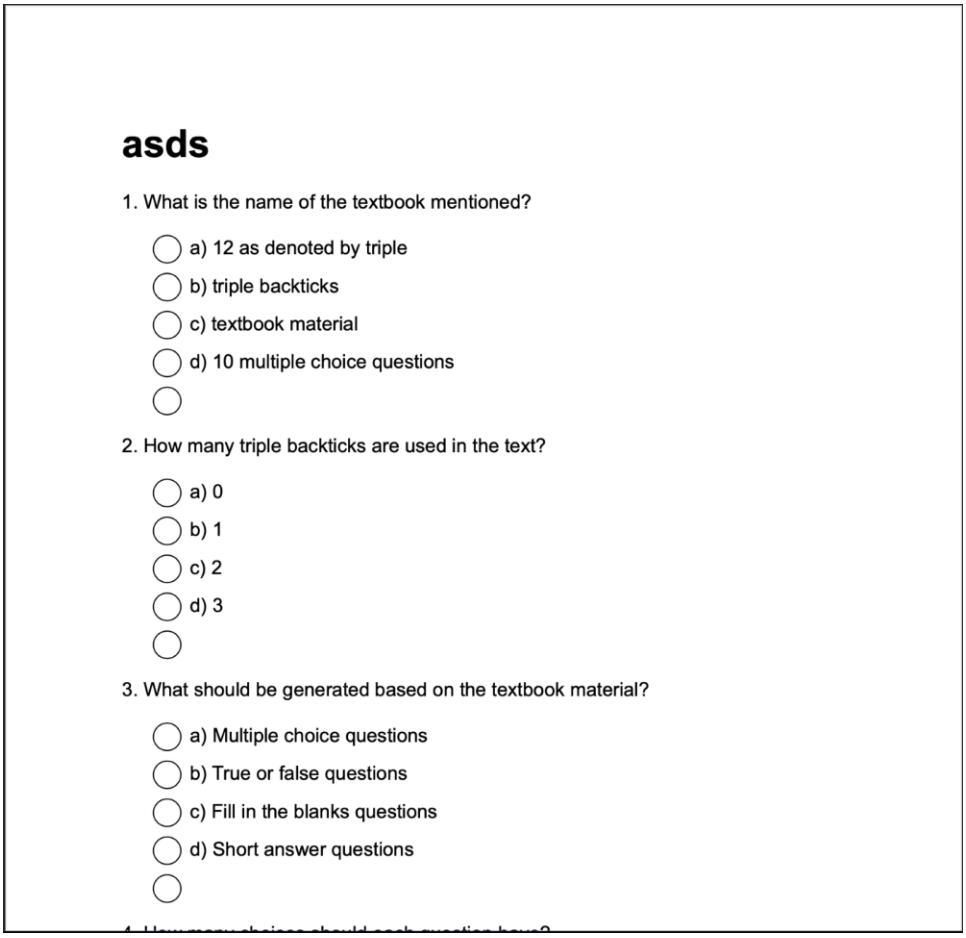
- Sprint#3 Functionalities

  Student US#1: Instructor Exports Quiz Questions

  After the user generates quiz question, they would have the option to export them into a separate PDF file for physical handouts.



Below is a sample of the generated PDF.



asds

1. What is the name of the textbook mentioned?

   a) 12 as denoted by triple

   b) triple backticks

   c) textbook material

   d) 10 multiple choice questions

2. How many triple backticks are used in the text?

   a) 0

   b) 1

   c) 2

   d) 3

3. What should be generated based on the textbook material?

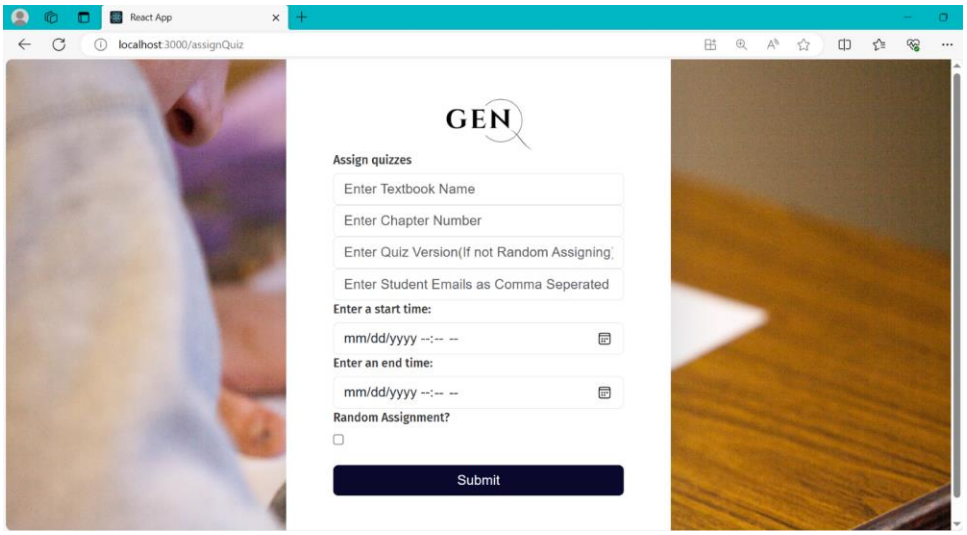
   a) Multiple choice questions

   b) True or false questions

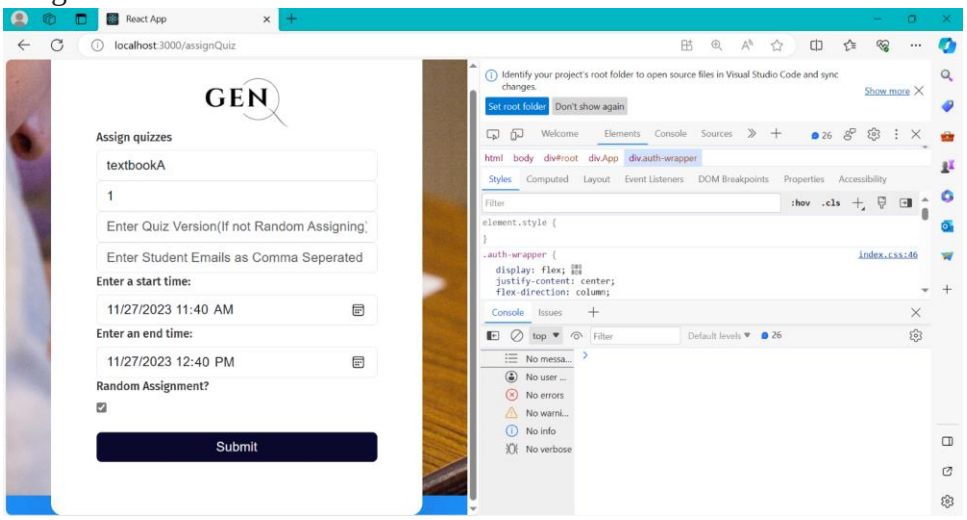
   c) Fill in the blanks questions

   d) Short answer questions

- **General Instructor User Story: Assigning Quizzes**



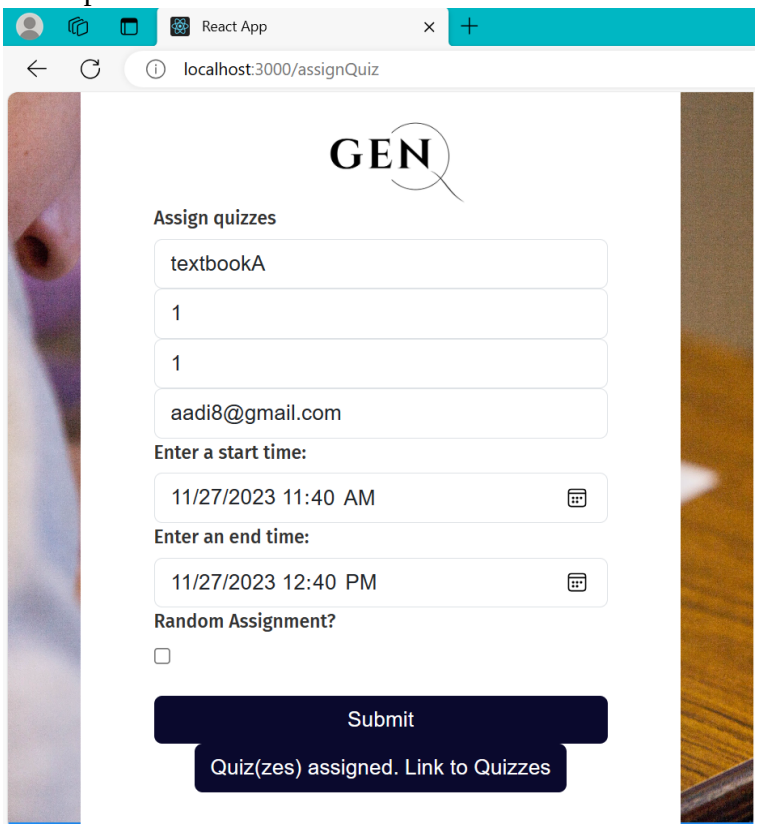
This is the first page that the professor accesses in order to assign quizzes. Here, they should enter the textbook name and chapter by default. Then, if they are randomly assigning quizzes, they just have to specify the start and end time and check the random assignment and hit the submit button.



Here is an example of a professor randomly assigning quizzes for textbookA, chapter 1, starting at 11:40 am and ending at 12:40 am.

```
▼ (10) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
  ▼ 0:
    ▶ canAccess: (10) [1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
      end_time: 202311271240
      id: 1
    ▶ questions: []
      quiz_duration: null
      start_time: 202311271140
      textbook_chapter: 1
      textbook_name: "textbookA"
      version: 1
    ▶ [[Prototype]]: Object
  ▼ 1:
    ▶ canAccess: (10) [2, 12, 22, 32, 42, 52, 62, 72, 82, 92]
      end_time: 202311271240
      id: 2
    ▶ questions: []
      quiz_duration: null
      start_time: 202311271140
      textbook_chapter: 1
      textbook_name: "textbookA"
      version: 2
    ▶ [[Prototype]]: Object
  ▶ 2: {id: 3, textbook_name: 'textbookA', textbook_chapter:
  ▶ 3: {id: 4, textbook_name: 'textbookA', textbook_chapter:
  ▶ 4: {id: 5, textbook_name: 'textbookA', textbook_chapter:
```


Here is a closeup of the value returned from the backend. All the quizzes have students equally assigned to each of the quiz versions. In addition, the start and end time have been specified.



Here, the professor specifically wants to assign the student with email aadi8@gmail.com to quiz version 1.

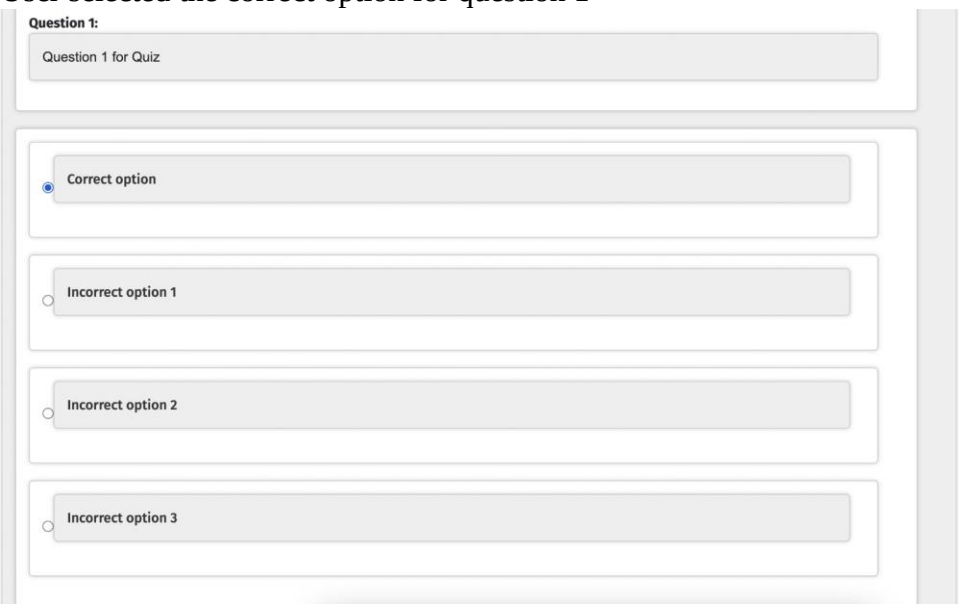
```
                                                    assignQuiz.js:44
▼ {id: 1, textbook_name: 'textbookA', textbook_chapter: 1, ve
  rsion: 1, start_time: 202311271140, …} i
  ▶ canAccess: (11) [1, 9, 11, 21, 31, 41, 51, 61, 71, 81, 91
    end_time: 202311271240
    id: 1
  ▶ questions: []
    quiz_duration: null
    start_time: 202311271140
    textbook_chapter: 1
    textbook_name: "textbookA"
    version: 1
  ▶ [[Prototype]]: Object
'newCreatedQuiz'
```

The backend API then updates just that quiz and inserts the student as having access to that quiz. Note that because python starts numbering at 0, and Django database starts at 1, the user id associated with aadi8@gmail.com is actually id 9.

Student US#3: Student Submits Quiz

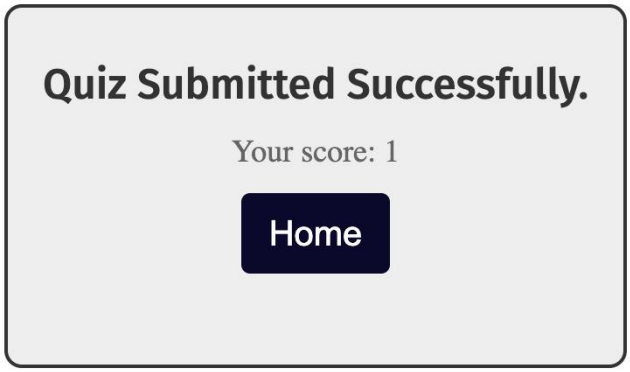
Once the user logs into the system and starts the quiz, user will have an option to select the answers for the quiz by clicking the radio button. Given below are two sample images for test quizzes.

User selected the correct option for question 1

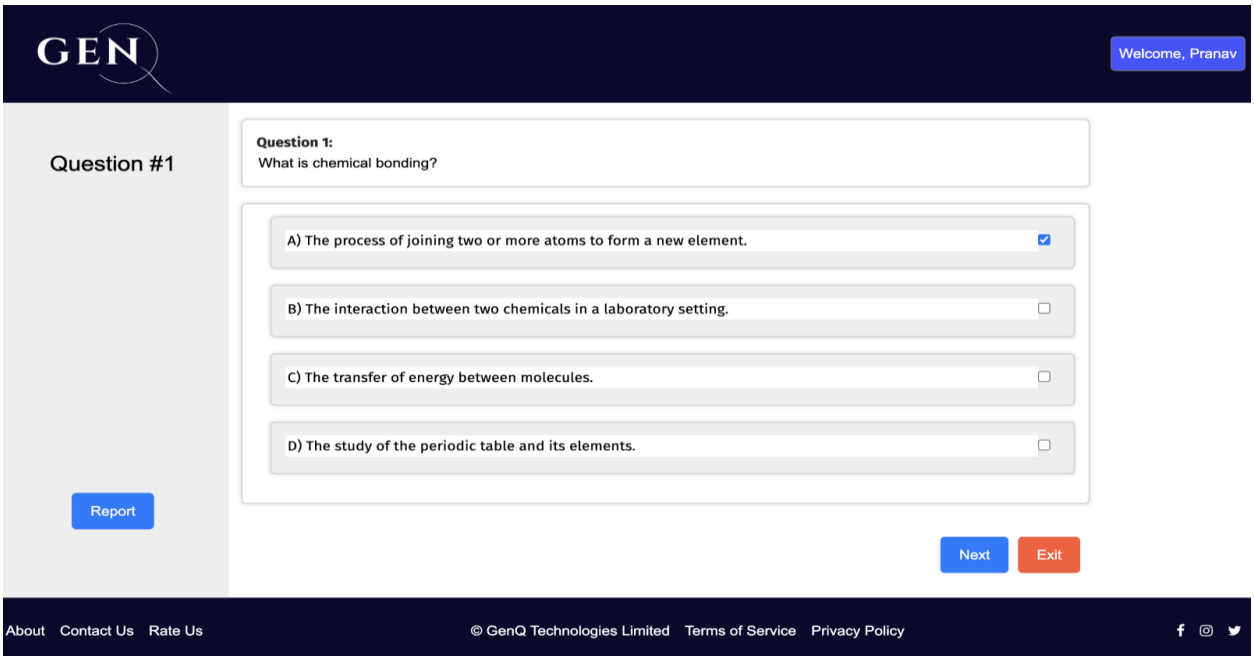


User selected the Incorrect option for Question 2

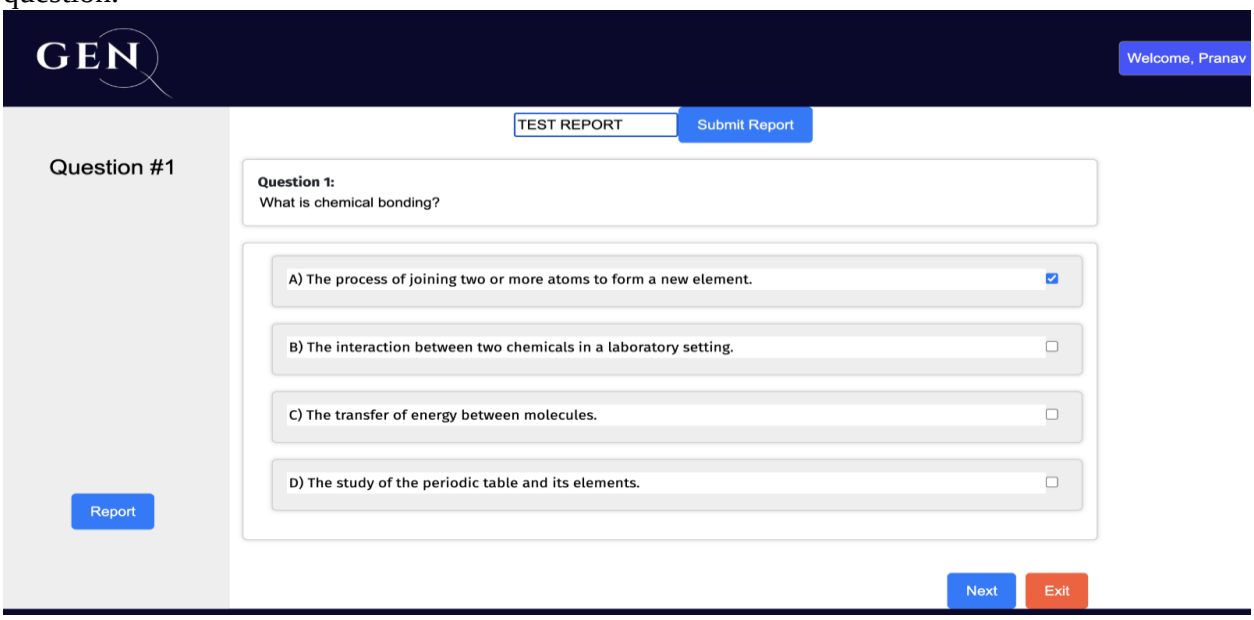
Once the user clicks on submit, he/she will be redirected to a new window in which the score of the quiz will be displayed.



Student US#2 : Student reports Quiz Questions



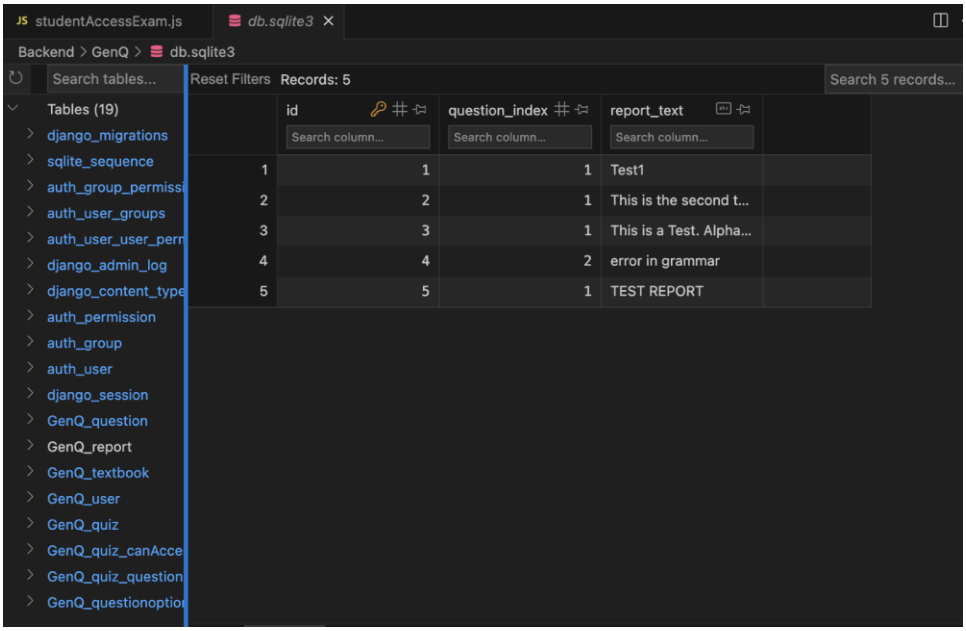
Upon accessing the generated sample quiz, the user will be directed to the sampleAccessQuiz frontend page where they will be presented a question with its respective answer choices. Here, we can see a report button that is available to select if the user wants to report anything about the question.



Upon selecting the report button, a text input box is presented to the user. Here, the text input will take the input from the user and translate it as a string datatype for the backend and it will also take the question index number as well. When the user hits the submit button, both of these data values will be sent to the backend.

```
urllib3/issues/3020
  warnings.warn(
System check identified no issues (0 silenced).
November 27, 2023 - 16:48:07
Django version 4.2.6, using settings 'GenQ.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[27/Nov/2023 16:48:17] "GET /question/ HTTP/1.1" 200 3033
[27/Nov/2023 16:48:17] "GET /question/ HTTP/1.1" 200 3033
[27/Nov/2023 16:48:57] "OPTIONS /saveReport/ HTTP/1.1" 200 0
[27/Nov/2023 16:48:57] "POST /saveReport/ HTTP/1.1" 200 40
```

The image above shows the server log messages in reaction to the submit report button being clicked by the user. We can see that the OPTIONS /saveReport/ accepts the response with a 200 message and then the POST /saveReport/ responds with a successful 200 message as well as a 40 message to indicate the creation of the resource.



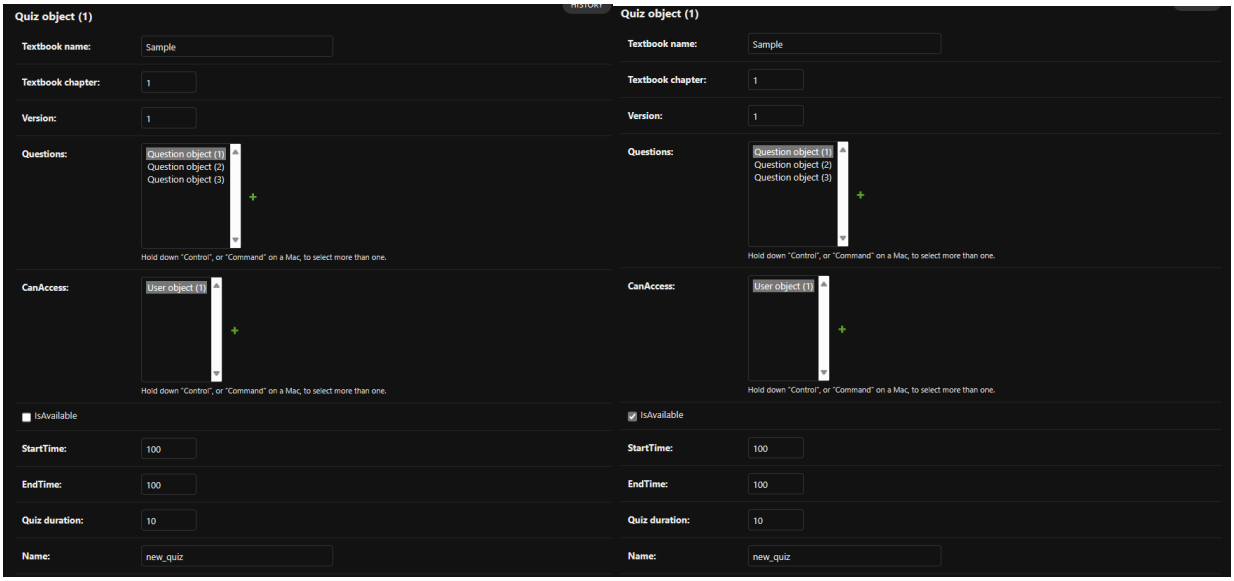
Finally, the image above shows the result of the report being saved in the database with a unique primary key id value as well as the two data values that were passed by the submitReport. The question_index column displays which question number the text belongs to and the report_text is the string content of the report itself.

Instructor User Story #9: Specify the starting time of quizzes

When the instructor creates a quiz, they specify a start time to the API. The start time automatically creates an ‘at’ command on the server to run the Django command which makes the quiz available. Below is the command line output of creating the ‘at’ command:

```
Capstone_Project/genq/Backend/GenQ/GenQ$ job 9 at Tue Nov 28 03:25:00 2023
```

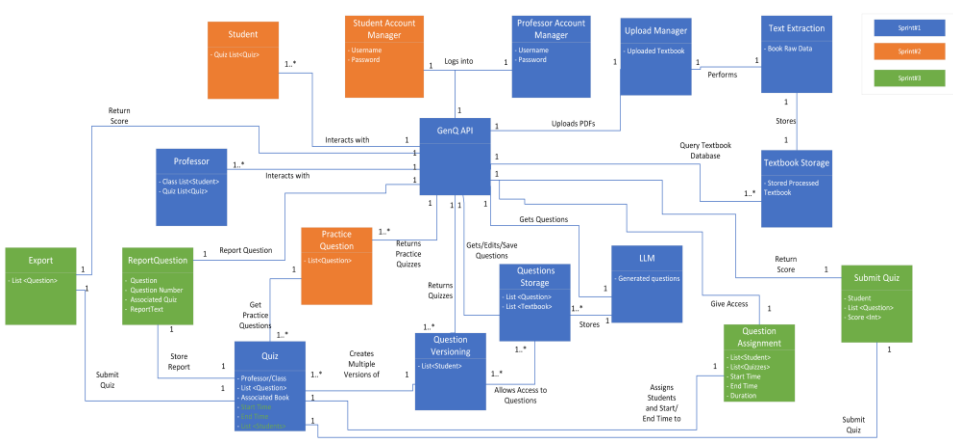
When the server clock reaches that time, the command triggers, and the is_available field on the associated quiz object is set to True. This lets the front end know that the quiz is now available for students to view and take.



Shown on the left is the quiz object before the specified time, on the right, after the specified time. Note that the isAvailable field becomes marked as True.

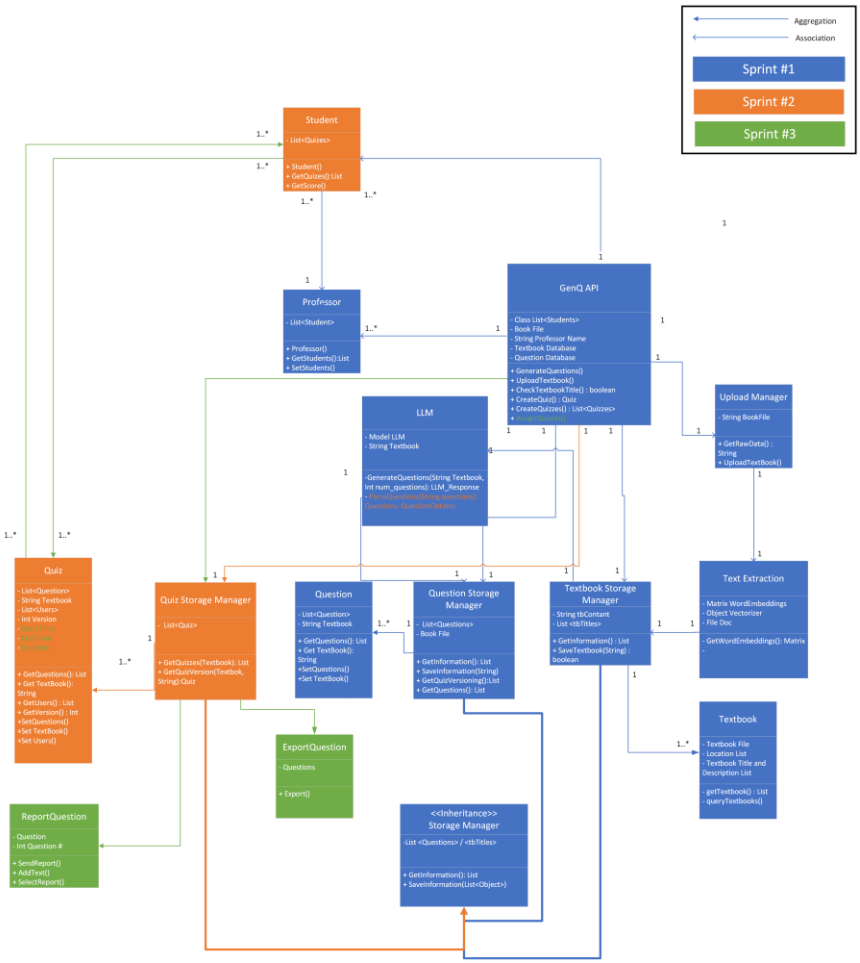
Design

- System Class Diagram:
  - The diagram below showcases our system's class diagram segmented into the three sprints.



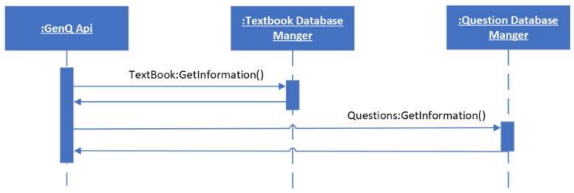
- System Domain Model

- The diagram below shows our domain model segmented into three sprints we had during the semester.



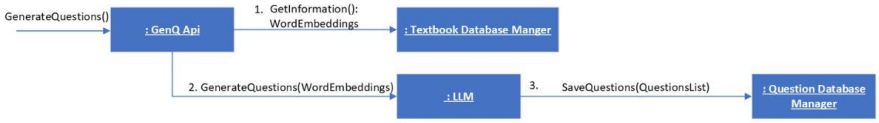
- Interaction Diagrams
  - Each of the following diagram references each of the sprint's interaction diagrams for each stories of that specific sprint.

Instructor User Story #10: Access Generated Questions



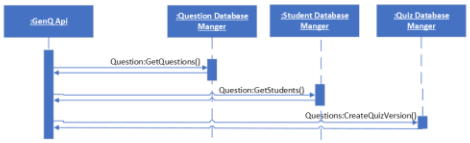
Instructor User Story #1: Upload Textbook

Instructor User Story #2: Generate Questions



Instructor User Story #3: Version Questions

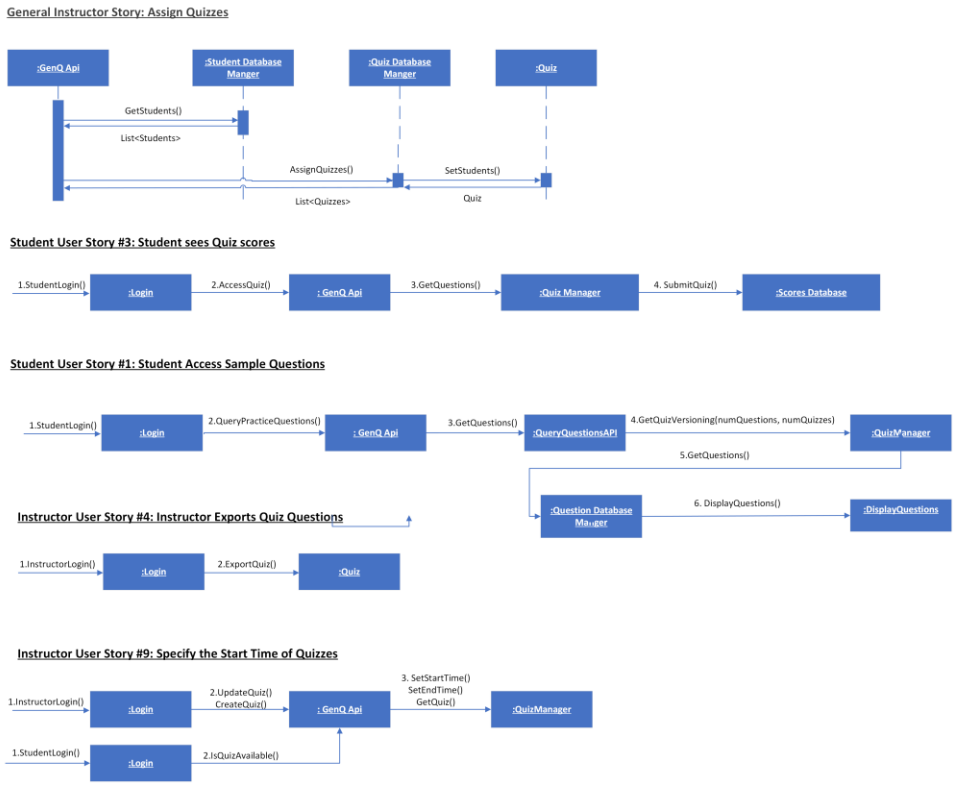
Instructor User Story #3: Version Quizzes



Instructor User Story #6: Instructor Edits Generated Questions

Instructor User Story#5 & Student User Story #1: Student Access Sample Questions

Instructor User Story #5: Instructor Choices Student Sample Questions



- System Tools
  - Frontend libraries
    - React

The frontend for this project is implemented using ReactJS. This open-source JavaScript library is used to build user interfaces for single-page applications. The component-based nature of the library allowed independent development among the team members. Each component is then linked to the backend library through API calls accessing an endpoint.

  - Backend libraries
    - Access API endpoints via Django
    - Django lite database

The backend for our design is implemented using the Django Python library. Django provides a fast, secure, and scalable web framework to create ‘views’ for each required API endpoint, respond to requests, and contains its own lite database to define and store backend objects. The API communicates with both the LLM and the Frontend UI via these API calls which are accessible through URL endpoints. ￼

  - IDE
    - Visual Studio Code

Our development was done in the Visual Studio Code IDE for its flexibility in handling the multiple different languages incorporated in the project.

- LLM library

The questions are generated using the gpt-3.5-turbo model from the python LLM library. This model is used for its stability, and relative speed compared to other available LLMs. Since it is already pre-trained, it can be used out of the box, or additionally fine-tuned on textbook question data for improved performance. LLMs function by tokenizing input sentences into numerical embeddings, then using a stacked encoder/decoder architecture to transform the sentence level embeddings into a new feature space. In our case, the desired output feature space are newly generated questions. LLMs can take in the context of an input by using bi-directional attention, so that each word can attend to each other word in the input.

- Fast API server

The question generator is served on a lightweight Fast API server. Fast API provides an easy-to-use api that allows communication between the question generation code and the backend. After receiving a request, the server creates a question and responds with a post request containing the question contents one of the Django server's endpoints. This application is much smaller and does not require a database like the main server, so FastAPI is more suitable here than Django.

Retrospection

What Went Well

- Well selected code stack (lightweight, straightforward, beginner friendly)
- No heavy database lifting (SQLite integrated with Django)
- Good intra-team communication (frontend)
- Time dedicated to other teammates
- Healthy balance of independent troubleshooting and reaching out for help
- MR process has improved
- Meeting up increased troubleshooting efficiency
- Intra-team planning helped not waste time
- *Team communication went well even when some members were on holiday*
- *Code readability helped for complex functionality during the sprint*

What Went Wrong

- Lack of documentation led to DB confusions
- Initial push/pull to remote repository was inefficient
- Wasted time by not reaching out for help too late
- Documentation on Merge Requests as well
- Starting earlier

- Unfinished stories due to holiday season

Skills Learned from This Project

- Most members of this project haven't had experience working on Web-based LLM Application
- First time using Django in large project
- First time using react in a large project
- Working in a large group
- First time using Gitlab

Hindsight Actions

The following actions should have been taken to avoid the problems encountered while working on the project:

1. Document Everything
    a. Resolution action: Including detailed steps for project environment set up and code testing is crucial for ensuring that people within your project do not waste time by not knowing what to do or missing a step, no matter how small. Team members should document their steps to set up their specific part of the environment (required libraries, outside technologies, hardware, etc.) and their steps to run new features (execution commands, new file configurations, feature overviews, etc.) to other people getting blocked.
2. Start Early
    a. Resolution action: Have team members discuss roadblocks in regular Monday/Tuesday/Thursday meetings. By identifying and resolving roadblocks early, team members can more easily start work at the beginning of sprints.
3. Ask for Help Early
    a. Resolution action: As software engineers, we will not know how to do everything, and we very well may get blocked on the things with which we do have experience. Therefore, it is imperative to ask someone for help (preferably in the domain of the problem) after a self-specified amount of time being blocked (I.e. one hour, two hours).

Product Improvements
- Integration with external systems (Canvas)
- Adding administration role
- Web application routing
- Making the application more cohesive
- Redesign the of the application to make it easier to all users