Product Design

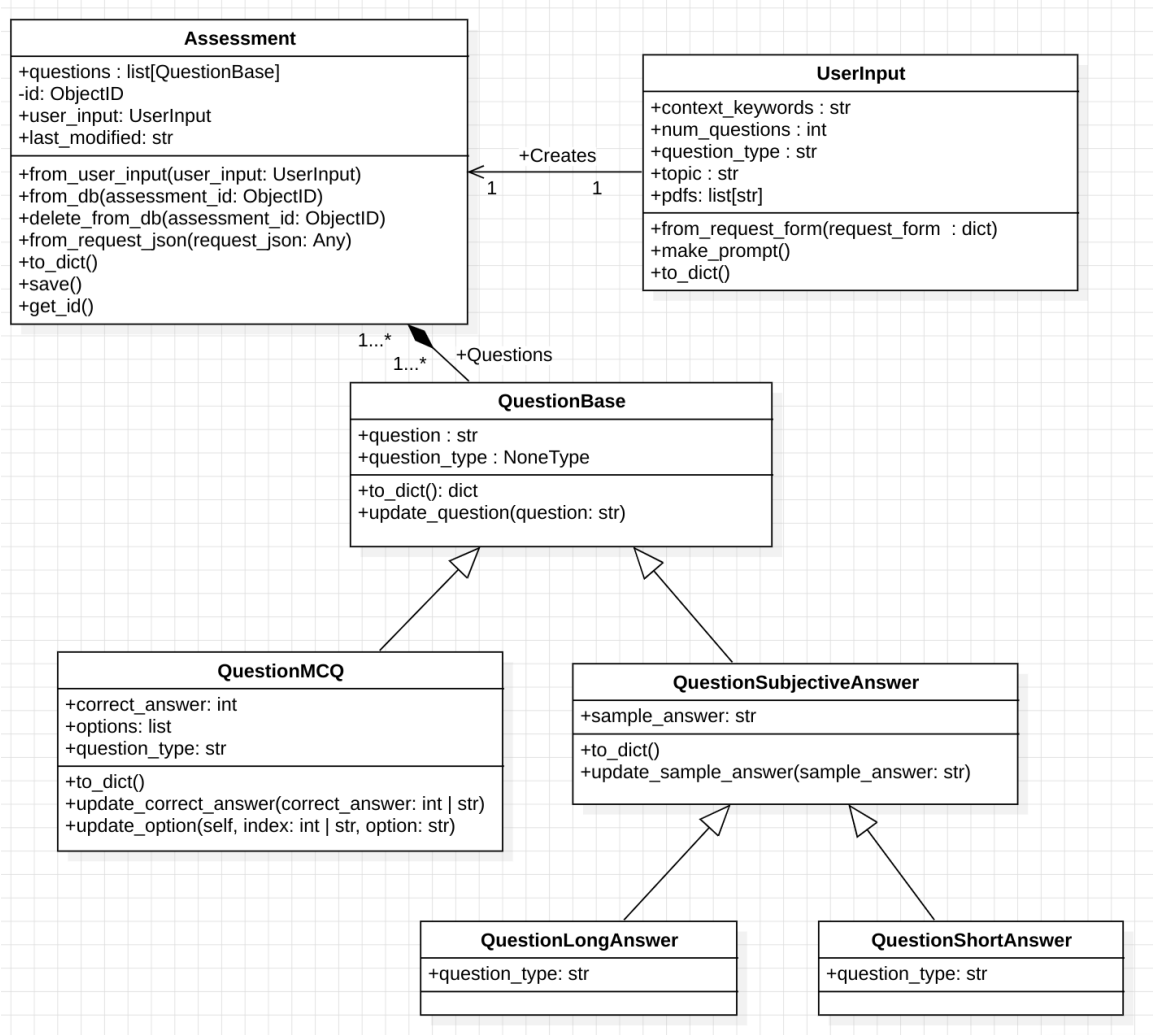
Team 20: Archisha Panda, Gargi Shroff, Ankith Pai, Prakhar Jain

Design Model

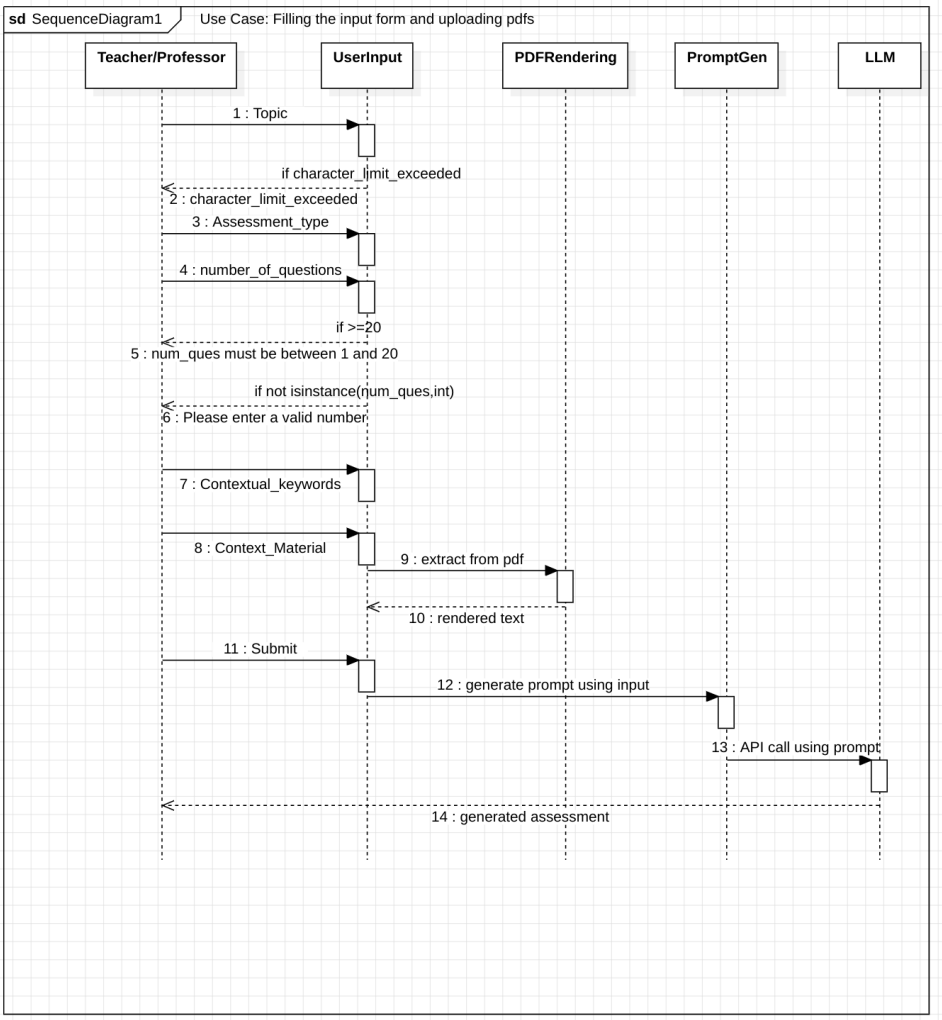
`UserInput`	This is a class that represents the input entered by the user sent from the frontend for page 1. Class state • `topic:` Topic of the assessment to be generated. • `question_type:` Type of the questions in the assessment. • `num_questions:` Number of questions to make. • `context_keywords:` Any optional contextual keywords needed for the LLM • `pdfs: List of PDF contexts uploaded.` Class behavior • `from_request_form:` Takes the input dictionary and instantiates the class from it. • `make_prompt:` Constructs the prompt string to be forwarded to the LLM. • `to_dict:` Method to get dict representation of current instance.
`QuestionBase`	This is a base class used to represent a question. Class state • `question:` Stores the question string. • `question_type:` Type of the question. Class behavior • `update_question:` This method updates the stored question • `to_dict:` Returns a dict representation of the question
`QuestionSubjectiveAnswer`	This is a subclass of `QuestionBase` for representing a subjective question, inherits all state and behavior, and additionally implements the following. *Class state* • `sample_answer:` Stores a sample answer. Class behavior • `update_sample_answer:` This method updates the stored question.
`QuestionShortAnswer`	This is a subclass of `QuestionSubjectiveAnswer` for representing a subjective short answer type question. It

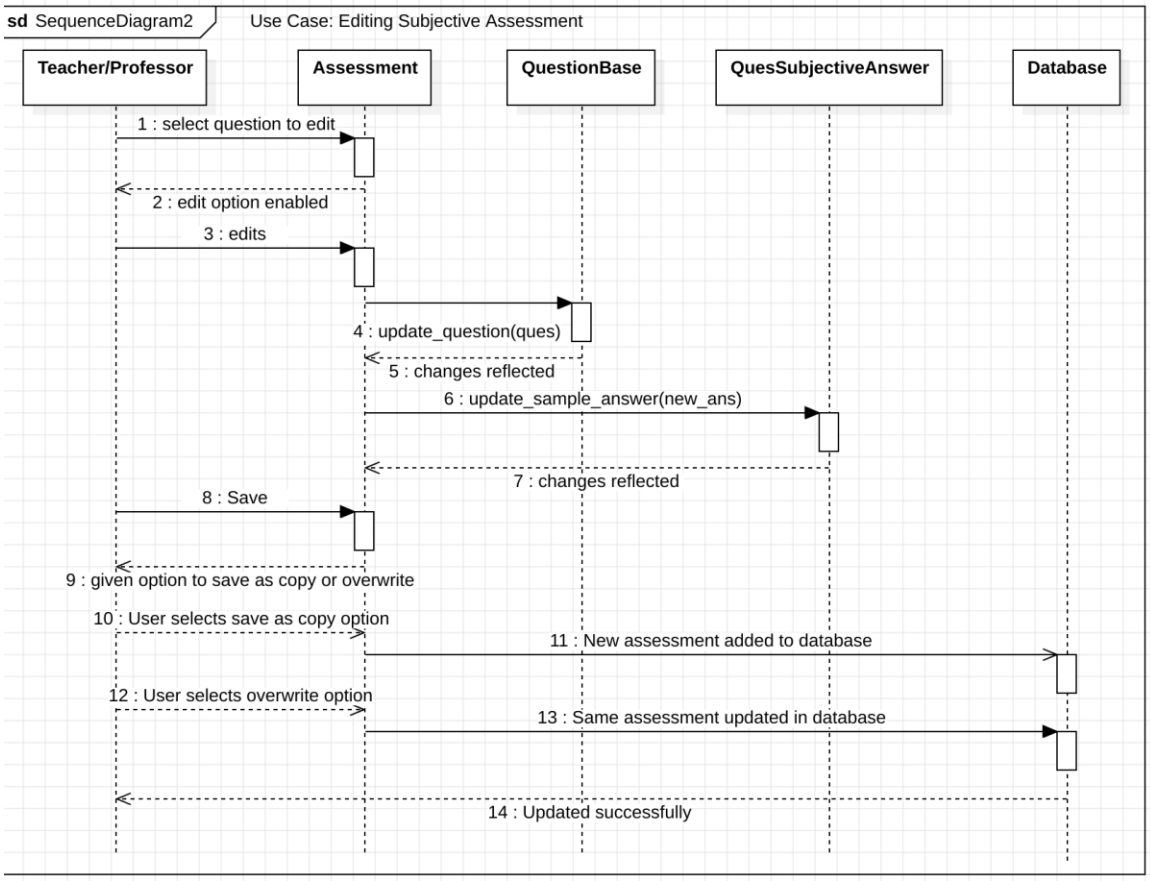
	inherits all state and behaviour, and does not implement anything more.
`QuestionLongAnswer`	This is a subclass of `QuestionSubjectiveAnswer` for representing a subjective long answer type question. It inherits all state and behaviour, and does not implement anything more.
`QuestionMCQ`	This is a subclass of `QuestionBase` for representing an MCQ question, inherits all state and behavior, and additionally implements the following. *Class state* • `options:` Stores a list of all options that are pickable. • `correct_answer:` Stores the index of the correct option. Class behavior • `update_option:` This method updates a particular option at the given index. • `update_correct_answer:` This method updates the correct answer.

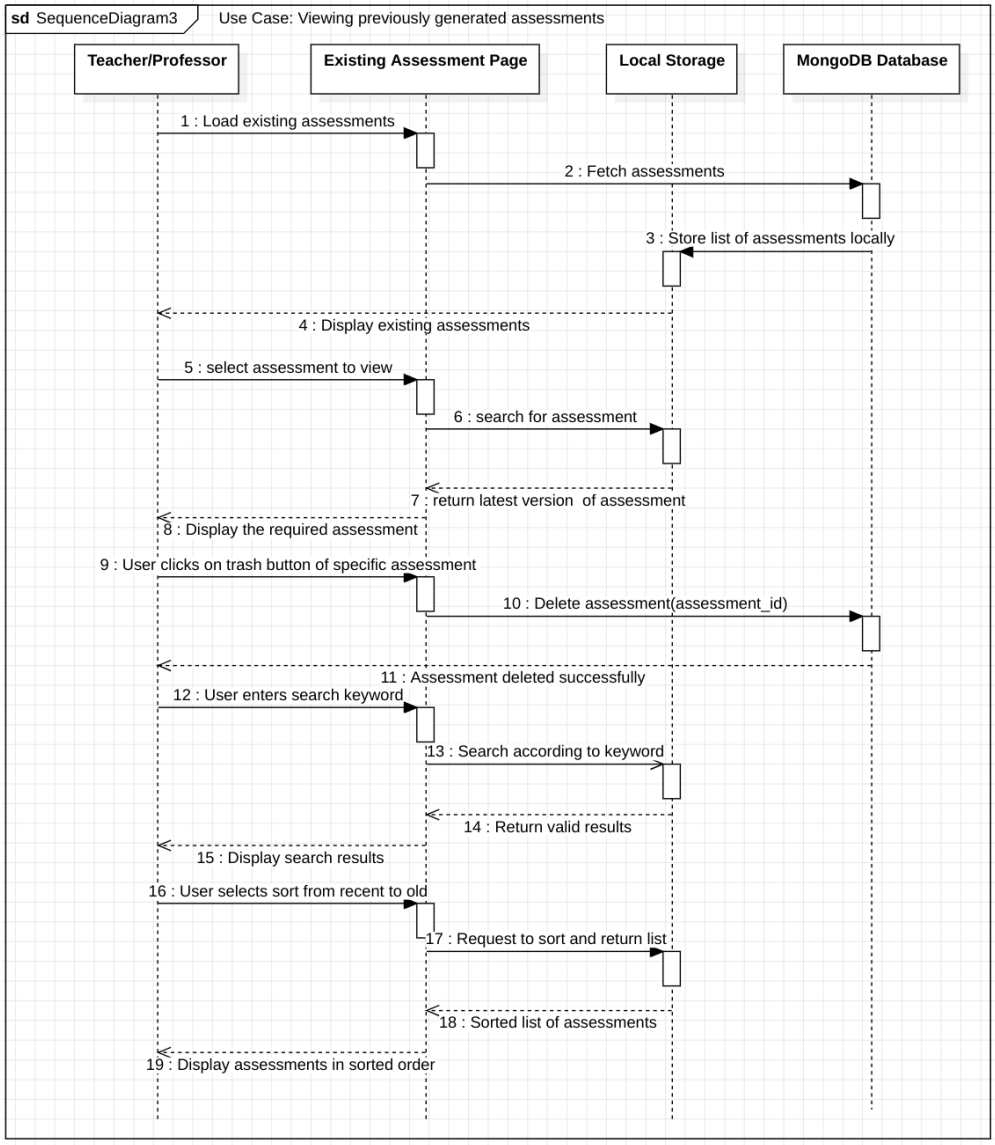
`Assessment`	This is a class that represents a generated assessment. *Class state* • `questions:` Stores the list of questions • `_id:` Stores a unique ID for the assessment • `user_input:` Stores the associated UserInput instance • `last_modified:` Stores the date/time the instance was created/modified Class behavior • `from_user_input:` Constructor to construct assessment from an UserInput instance • `from_db:` Constructor to get assessment from database, given assessment id. • `delete_from_db:` Method to delete an instance of assessment from the database. • `from_request_json:` Constructor to construct assessment from a request object (sent by save endpoint) • `to_dict:` Returns dict representation of the assessment • `save:` Saves the assessment instance in the database. • `get_id:` Helper function to get the ID of the instance

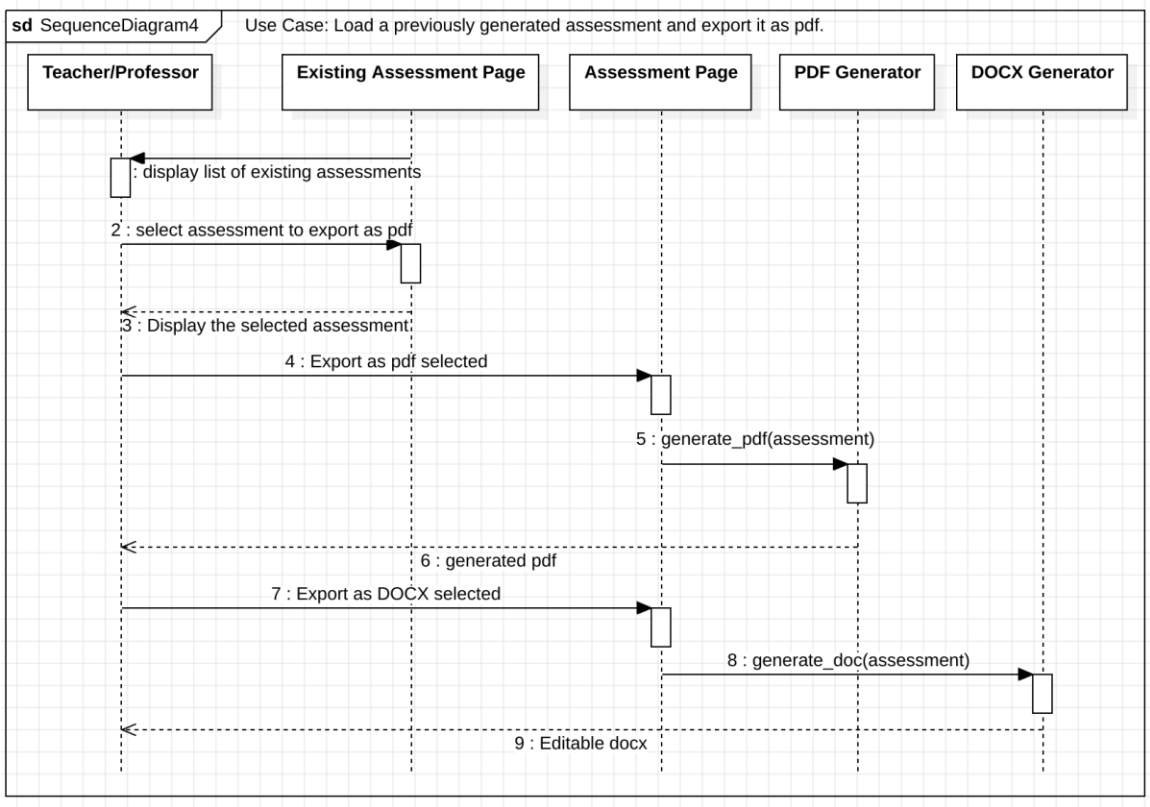


Sequence Diagram(s)


sd SequenceDiagram1
Use Case: Filling the input form and uploading pdfs
Teacher/Professor
UserInput
PDFRendering
PromptGen
LLM
1 : Topic
if character_limit_exceeded
2 : character_limit_exceeded
3 : Assessment_type
4 : number_of_questions
if >=20
5 : num_ques must be between 1 and 20
if not isinstance(num_ques,int)
6 : Please enter a valid number
7 : Contextual_keywords
8 : Context_Material
9 : extract from pdf
10 : rendered text
11 : Submit
12 : generate prompt using input
13 : API call using prompt
14 : generated assessment


sd SequenceDiagram2
Use Case: Editing Subjective Assessment
Teacher/Professor
Assessment
QuestionBase
QuesSubjectiveAnswer
Database
1 : select question to edit
2 : edit option enabled
3 : edits
4 : update_question(ques)
5 : changes reflected
6 : update_sample_answer(new_ans)
7 : changes reflected
8 : Save
9 : given option to save as copy or overwrite
10 : User selects save as copy option
11 : New assessment added to database
12 : User selects overwrite option
13 : Same assessment updated in database
14 : Updated successfully


sd SequenceDiagram3
Use Case: Viewing previously generated assessments
Teacher/Professor
Existing Assessment Page
Local Storage
MongoDB Database
1 : Load existing assessments
2 : Fetch assessments
3 : Store list of assessments locally
4 : Display existing assessments
5 : select assessment to view
6 : search for assessment
7 : return latest version of assessment
8 : Display the required assessment
9 : User clicks on trash button of specific assessment
10 : Delete assessment(assessment_id)
11 : Assessment deleted successfully
12 : User enters search keyword
13 : Search according to keyword
14 : Return valid results
15 : Display search results
16 : User selects sort from recent to old
17 : Request to sort and return list
18 : Sorted list of assessments
19 : Display assessments in sorted order



Design Rationale

- Removed the `from_dict` classmethod from all question classes and `from_list` and `from_str` methods of `Assessment`. The functionality of these methods is now executed by the class constructor, reducing redundancy in the original design and increasing cohesion.
- Added a `QuestionSubjectiveAnswer` class that `QuestionLongAnswer` and `QuestionShortAnswer` inherit from. This removes code duplication in these two classes, by moving their common functionality to the super class.
- Added `update_*` methods that update various attributes of question classes. This is done so that input validation can be enforced.
- The PDF export functionality had to be refactored to fix some technical and visual issues in the rendered PDF (one such issue was that some page-breaks were incorrectly being inserted in the PDF). Previously, the PDF export method was just scraping the text from the rendered page, but we then updated it so that it can directly use the data available in a dictionary format.
- Added methods `from_db`, `delete_from_db`, `save` and `get_id` for implementing and abstracting away database interfacing. Due to these methods, the higher-level caller code now does not need to directly interface with database API.
- Added `from_user_input` and `from_request_json` convenience constructors that can be called directly from higher-level code.
- Added `to_dict` method to `UserInput` and `Assessment` because this functionality was needed in a couple of places, so these methods reduce duplication.