Requirements Document - Prompt Generation

Note: This requirements document is intended only for the prompt generation component of the project, so the expectation is that all of Geela's requirements will be covered across all 3 teams' requirement documents.

Functional requirements

- Convert Input parameters (question type, context, regeneration flag) to prompt
- Ensure the regenerated question is different to the original question. (Changed prompt to question)
- Ensure the code given is syntactically correct and contains the context and topic. (Added)
- Send the generated and checked code snippet to the team that is responsible for the Parson's problem interface. (Changed the phrasing of the sentence)
- Ensuring that the client has the ability to view the data analytics (e.g. time spent in the session)

Requirements	Priority	Solutions	Status	Acceptance Criteria
Converting input parameters to prompt.	High	We will be taking in 2 strings of the topic and context and generate a prompt string.	Completed	The backend receives topic, context with 100% success. API calls are documented.
Ensuring the regenerated question is different to the original question.	High	We will be storing all the code that was generated for that session and ensure that the user does not receive the same question twice during their session.	Completed	Implemented topic-specific prompt with instructions. The chat history feature is also put in place.
Ensuring the code given by Gemini is syntactically correct and contains the context and topic.	High	We will be implementing a Python interpreter to ensure that the code is correct the moment we receive the code given by Gemini. Once the code is correct, we plan to send the code back to Gemini to ask if the code contains the given contest and topic.	Completed (context and topic weren't issues)	Implemented a Python interpreter to check for code errors successfully. Topic and context inclusion is also ensured since Gemini is quite good at doing this.
Sending the code to the team that is	High	We will be converting the	Completed	Team 4's Problem Page receives

responsible for the Parson's problem interface.		code, description and expected output into JSON format and will be sending the other team a JSON object.		generated problem with 100% success. API calls are documented.
Ensuring client has the ability to view data analytics.	High	We will get the user IP address and store the time spent on the session, number of questions attempted and number of questions that the user got right.	Completed	Database is implemented alongside all features needed to access the necessary data analytics. Team 2 confirms success of data retrieval.

Non-functional requirements

Requirement type	Requirement	Priority	Solution	Status	Acceptance Criteria
Supportability	System should be open to extension to more topics and context in the future.	Medium	Ensure the code is well documented for the ease of adding new things. Use good coding practices (eg GRASP principles) to keep code open to extension	Completed	Prompt is built in a way that doesn't require any specific contexts. Instructions for each topic are in separate js files with a string being returned to ensure it can be added easily to prompt.
Reliability	System should return code that is at least syntactically correct and contains no run time errors.	High	Filtering the code from chatgpt to ensure that the code that is given to be correct before showing it to the students. Use LLM to check. Run the code with in built	Completed	Implemented a Python interpreter to check for code errors successfully.

			compiler / interpreter to see if it results in an error.		
Reliability	System should check that code actually contains context and is the correct problem type	High	Return the generated code back to the LLM in a new chat to eliminate any bias to check if the given context and topic is in the code.	Not necessary	Gemini is very effective at ensuring this already.
Reliability	System should return a backup problem based on the context and topic if LLM fails (5 tries).	Low	Store a set backup problems for combinations of topics and contexts.	Completed	Backup problem feature is implemented successfully. A timeout of 20 seconds is added when generating new problem (instead of 5 tries)
Performance	System should not take over 30 seconds from generation to receiving the problem	Medium	Run tests throughout development to ensure it does not take over 5 seconds.	Completed	A timeout of 20 seconds is added when generating new problem (instead of 30 seconds)
Performance	System shouldn't generate repetitive problems	Low	Either store past prompts. Or use same chat log to ensure different prompts are generated to previous ones	Completed	Implemented topic-specific prompt with instructions. The chat history feature is also put in place.
Performance	System should take less than 30 attempts to generate a problem that passes checks	Medium	Tune the LLM to improve its prompt generating abilities (May not be necessary if LLM is already good at	Backup problem implementation removed the need for this requirement	Backup problem feature is implemented successfully. A timeout of 20 seconds is added when generating new

			generating problems)		problem (instead of 5 tries)