# High-Level Architecture (Block Diagram)

* **Frontend (Web):**
  + **Technology:** HTML, CSS, JavaScript (Framework like React or Vue.js would be ideal for efficient UI updates)
  + **Components:**
    - Text Input Area
    - Question Display (one at a time)
    - Multiple Choice Options
    - Submit/Next Button
    - Basic Results/Feedback Area
* **Backend (Server):**
  + **Technologies:** Python (for NLP libraries, GPT/LLAMA model interaction), Node.js (for its asynchronous capabilities would be suitable)
  + **Components:**
    - API endpoint to receive text input.
    - NLP Processing Module (Leverages chosen NLP technique)
    - Question Generation Logic
    - Connection to Database
* **Database:**
  + **Technology:** A simple SQLite database would suffice for PoC purposes. For scaling, consider PostgreSQL or MySQL.
  + **Data Storage:**
    - Input Text
    - Generated Questions
    - User Responses
    - Basic Performance Metrics (Accuracy, etc.)

# Workflow Design (Flowchart)

1. **User Enters Text:** User pastes or uploads text into the designated input area on the frontend.
2. **Text Sent to Backend:** Frontend sends the text to the backend API endpoint.
3. **NLP Preprocessing:** Backend server preprocesses the text (if required for the chosen NLP technique)
4. **Key Concept Identification:** NLP module analyzes the text, identifying important concepts and potential question targets.
5. **Question Generation:** Using the identified concepts, the backend generates multiple-choice questions with plausible distractors.
6. **Data Storage:** Questions and the original text are stored in the database.
7. **Question Display:** Frontend receives the first question and options, rendering them for the user.
8. **User Selects Answer:** User interacts with the UI, selecting their chosen answer option.
9. **Answer Submission:** Frontend sends the selected answer to the backend.
10. **Answer Validation:** Backend compares the answer against the correct one, calculates accuracy metric.
11. **Feedback and Next Question:** Frontend displays basic feedback to the user and loads the next question (repeat 8-11)
12. **Quiz Completion:** After all questions are answered, calculate aggregated metrics.
13. **Result Storage:** Store the final results and metrics in the database.

# Message Sequence Chart (MSC): Question Generation

* **Lifelines:** UI (Frontend), NLP Module (Backend), Question Generator (Backend), Database
* **Message 1:** UI sends text to the NLP Module.
* **Message 2:** NLP Module processes text, returning key concepts.
* **Message 3:** Question Generator receives concepts and generates questions.
* **Message 4:** Question Generator sends questions + correct answer to Database for storage.
* **Message 5:** Database confirms successful storage.
* **Message 6:** Question Generator sends the first question and options back to the UI.

**Optional Deliverables:**

* **UI Wireframe:**
* **Sample Code (Python snippet for Question Generation):**

Python

import spacy # Example: using Spacy library for NLP

# ... (Loading of relevant language model)

def generate\_questions(text):

nlp = spacy.load("en\_core\_web\_sm")

doc = nlp(text)

# Example: Basic question using named entities

for ent in doc.ents:

question = f"What is the significance of {ent.text} in the text?"

options = # Generate plausible options

yield question, options

**Guidance** Let me know if you want specific aspects of any of these diagrams refined further or additional MSCs created. The objective is to provide enough visual detail aligned with the SRS to start the development process!