**AI Quiz Maker –**

**Automatically Generate Fun**

**Trivia Quizzes**

**Submitted By – Yash Raj Anand**

**Project Description:**

AI Quiz Maker is a cutting-edge, interactive quiz platform designed to address the shortcomings of traditional static quizzes. While conventional quizzes are often static and one-size-fits-all, this system takes a personalized approach, making quizzes more engaging and adaptive. The AI Quiz Maker creates a dynamic learning experience that adjusts to each user’s needs, ensuring that users of all levels stay motivated and challenged.

The core of AI Quiz Maker is its ability to generate multiple-choice trivia questions in real-time based on any topic of the user’s choice. Powered by the Mistral model via Ollama, the system leverages advanced natural language processing to generate accurate and relevant questions, removing the need for manual quiz creation. This AI-driven approach ensures that the quiz content is both unique and tailored to the individual’s topic interests, whether they are studying for exams, learning a new subject, or simply testing their general knowledge.

One of the standout features of the platform is its adaptive difficulty system. As users progress through the quiz, the complexity of questions adjusts based on their performance, providing an experience similar to working with a personal tutor. Correct answers increase the difficulty of upcoming questions, while incorrect answers adjust the difficulty downward, ensuring that users are always presented with questions that are appropriately challenging.

The application is built using FastAPI for the backend, providing a fast, secure, and scalable framework for serving quiz data. The frontend is designed with HTML, CSS, and JavaScript, ensuring that the platform is responsive and user-friendly. The design incorporates multiple background themes, allowing users to customize their quiz environment, creating a more engaging and visually pleasant experience. Immediate feedback after each question helps users understand their strengths and areas for improvement, reinforcing learning through instant scoring and answer reviews.

In summary, AI Quiz Maker transforms traditional, static quizzes into dynamic, engaging, and personalized learning experiences that adapt to the learner's needs. The system effectively combines artificial intelligence, real-time performance tracking, and intuitive design, making it a powerful tool for learners of all kinds.

**Scenarios:**

**Scenario 1: Instant Topic-Based Quiz Generation for Learners**

A college student preparing for a competitive exam wants to test their understanding of “Quantum Physics.” Instead of searching for static quizzes online, they open the AI Quiz Maker, select the “Tech” theme for visual comfort, and input the topic. The system instantly generates 10 multiple-choice questions tailored to that subject using the Mistral model. This real-world use case showcases how the platform eliminates the need for manual quiz preparation, making topic-specific revision fast and intelligent.

**Scenario 2: Personalized Difficulty Adjustment in Real-Time**

A user takes a quiz on “World History” but finds the first few questions too easy. As they answer correctly, the backend automatically increases the difficulty of upcoming questions (easy → medium → hard). When they make a mistake, the system lowers the difficulty to match their knowledge level. This scenario demonstrates how AI Quiz Maker mimics a tutor-like experience by adapting to each user’s capability—making the quiz more engaging and appropriately challenging.

**Scenario 3: Real-Time Feedback and Learning Reinforcement**

After completing a 10-question quiz on “Human Anatomy,” a user receives an immediate performance summary. This includes their final score, a breakdown of which questions they got right or wrong, their selected answers, and the correct answers—each visually color-coded. This helps the user identify weak areas and reinforces learning through feedback. This scenario reflects the project’s goal of providing real-time evaluation and a clear path for improvement.

**Pre-requisites:**

1. FastAPI Framework Knowledge: [FastAPI Documentation](https://fastapi.tiangolo.com/)
2. Ollama and Local LLM Setup (Mistral): [Ollama Documentation](https://ollama.com/)
3. HTML, CSS, and JavaScript Skills: [W3Schools HTML/CSS/JavaScript Tutorials](https://www.w3schools.com/)
4. Python Programming Proficiency: [Python Documentation](https://docs.python.org/3/)
5. Fast API Course : https://youtu.be/7t2alSnE2-I?si=n6P2SY3Ig3L26Nim

**Technical Architect**

AI Quiz Maker

Full stack Application

Adaptive Difficulty

(Easy -Medium- Hard)

HTML/CSS/JS

(Frontend)

Receives

Evaluate Answers

And Feedback

User Input

(Topic + Theme)

Final Result Summary

(Score+ Answer + feedback

Sends Request

Triggers

Fast API

(Backend)

Calls

/quiz/generate

Prompt

Ollama

(Mistral Model)

Generate Question

(10 MCQ)

**Project Overview:**

The AI Quiz Maker project is structured around solving the problem of time-consuming manual quiz generation. This solution allows users to input any topic and instantly receive a multiple-choice quiz generated via a locally hosted AI model. The application adjusts its difficulty dynamically based on real-time user performance and supports a themed, responsive user interface for engagement.

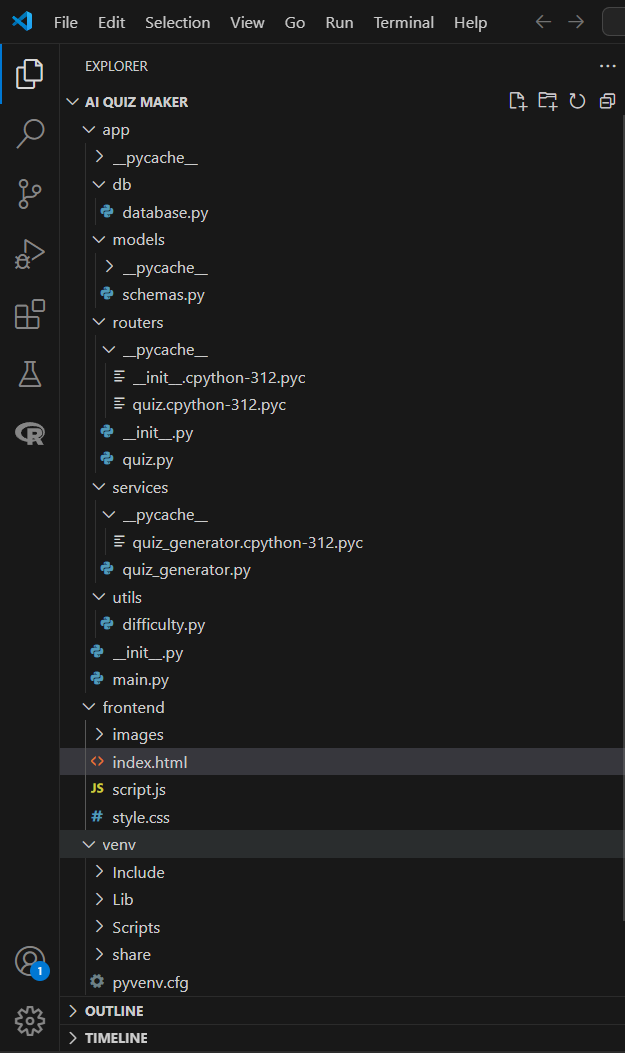
Below is a summary of the key phases and associated activities that shaped this project:

* **Phase 1: Research & Planning**  
  Conducted comparative analysis of LLMs like GPT-3, T5, BERT, and selected Mistral via Ollama for lightweight local deployment. Defined architecture using FastAPI for backend and HTML/CSS/JS for frontend.
* **Phase 2: Backend Development**  
  Implemented RESTful APIs using FastAPI. Integrated Mistral for question generation via structured prompts. Managed user sessions, answer tracking, and adaptive difficulty using Python and UUID.
* **Phase 3: Frontend Implementation**  
  Designed a modern UI with theme selection, quiz screens, and a results page. Built JavaScript logic for real-time answer submission, progress display, and final score summary.
* **Phase 4: Personalization & UX Enhancements**  
  Added multiple theme backgrounds, hover effects, and animated transitions to enhance interactivity. Integrated a visual feedback system showing correct/incorrect answers.
* **Phase 5: Testing & Validation**  
  Used Swagger UI for endpoint testing and ensured model compliance with JSON formatting. Debugged edge cases and verified functional stability through repeated test runs.

**Project Directory Structure:**

To support the modularity and scalability of the project, the directory structure was carefully organized:

* **app/** – Backend logic:
  + db/ – (Future scope) Database setup.
  + models/ – Pydantic schemas for data validation.
  + routers/ – API routes for quiz generation and answer handling.
  + services/ – Integration with Mistral via Ollama.
  + utils/ – Difficulty logic and helper functions.
  + main.py – FastAPI app entry point.
* **frontend/** – UI layer:
  + index.html – Core quiz interface and layout.
  + script.js – Handles API calls, question display, and result logic.
  + style.css – Styling, theme switching, and responsiveness.
* **venv/** – Python virtual environment managing dependencies.



**Milestones:**

**Milestone 1: Model Integration and Architecture Planning**

This milestone laid the technical foundation of the AI Quiz Maker application. It involved selecting an appropriate language model, structuring the full-stack architecture, and installing all essential tools and frameworks to establish smooth and scalable integration.

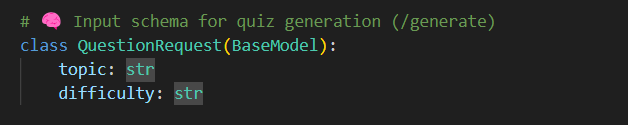
* **Activity 1.1: Model Selection**  
  A range of LLMs such as GPT-3, GPT-4, T5, Bloom, and BERT were explored. Mistral was chosen due to its open-source flexibility, fast inference time, and ability to run efficiently on local machines. Using Ollama, the Mistral model was deployed without internet dependence. This provided high performance and greater control over data privacy, an essential factor for local testing and educational tools. The JSON-compatible response format also made it ideal for consistent backend parsing.
* **Activity 1.2: Architecture Design**  
  The system was organized into three layers:
  + **Frontend Layer:** Developed with HTML, CSS, and JavaScript for user input, theme selection, and quiz display. It communicates with the backend via fetch requests.
  + **Backend Layer:** Built using FastAPI to serve API routes for starting a session, answering questions, and generating new ones. It also tracks user progress using UUID-based sessions.
  + **AI Integration Layer:** The backend interacts with the local Mistral model via HTTP requests to generate quiz questions. It parses AI responses and ensures fallback if malformed data is returned. This architecture ensured separation of concerns, scalability, and ease of debugging during development.
* **Activity 1.3: Environment and Tool Setup**  
  To enable efficient development and model serving, the environment was carefully prepared:
  + Installed Python 3.10+ and created virtual environments for package isolation.
  + Installed FastAPI, Uvicorn (for ASGI server), Requests (for API calls), Pydantic (for schema validation), and UUID (for session management).
  + Installed and launched Ollama, verified the Mistral model was available and responsive via ollama run mistral.
  + Conducted initial endpoint testing using Swagger UI (auto-generated by FastAPI) and Postman to ensure the model returned a properly formatted JSON object in response to the quiz prompts.

This milestone was critical in laying down a reliable infrastructure that combined performance, modularity, and AI capability while remaining lightweight and easy to maintain.

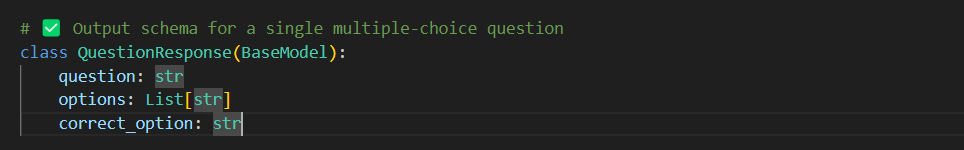
**Milestone 2: Backend Development**

This milestone focused on creating a fully functional backend that could communicate with the AI model, validate inputs and outputs, manage quiz sessions, and adjust question difficulty dynamically based on user performance.

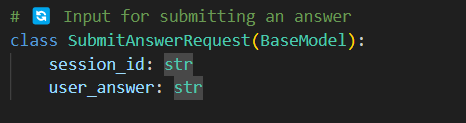
* **Activity 2.1: Schema Creation with Pydantic**  
  Pydantic models were used to define structured schemas for API data validation. Each schema ensured type safety and consistency in request and response formats:
  + QuestionRequest: Accepts a topic and difficulty, ensuring valid quiz generation requests.



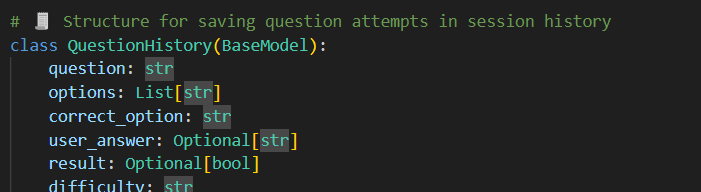
* + QuestionResponse: Contains the AI-generated question, options, and correct\_option to structure quiz content.



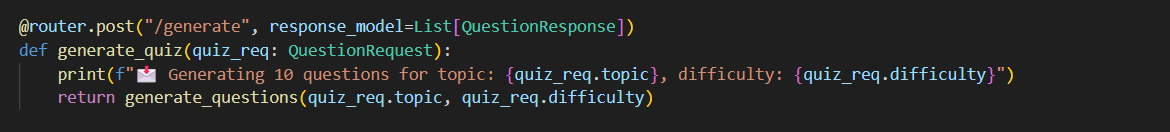
* + SubmitAnswerRequest: Captures session\_id and the selected answer, used to maintain quiz flow.



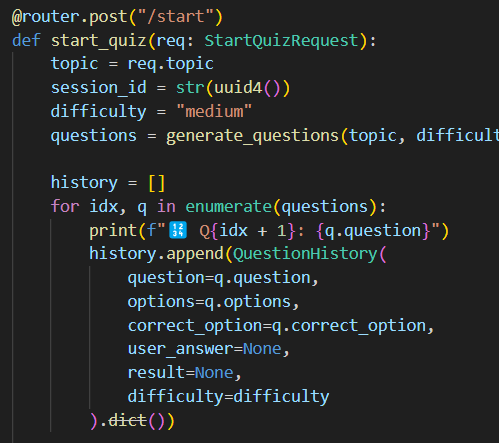
* + QuestionHistory: Stores each user's response, correct answer, result, and difficulty, useful for real-time feedback and session tracking. This modular schema design improved error handling, allowed for Swagger auto-documentation, and enabled reliable testing.



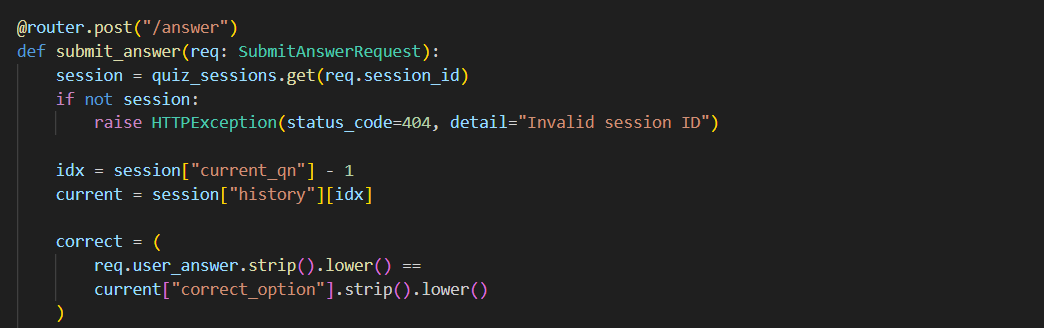
* **Activity 2.2: API Endpoint Implementation**  
  The backend routes in quiz.py provided the functional logic for interacting with the AI model and serving quiz content. These included:
  + POST /quiz/generate: Accepts a topic and difficulty level, invokes the Mistral model, and returns 10 quiz questions.



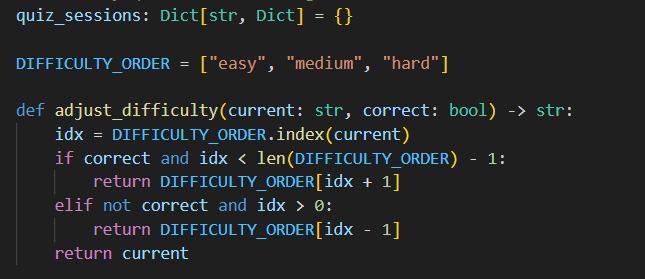
* + POST /quiz/start: Initializes a new quiz session with a unique session ID, sets default difficulty to medium, and stores generated questions.



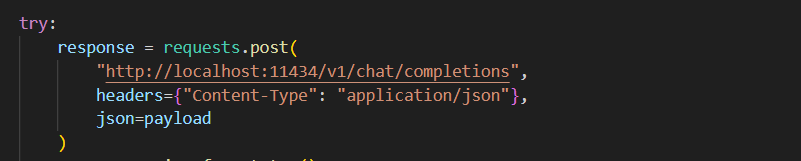
* + POST /quiz/answer: Accepts user answers, updates quiz history, adjusts difficulty using logic, and returns the next question. Middleware for CORS support was added to allow frontend communication.



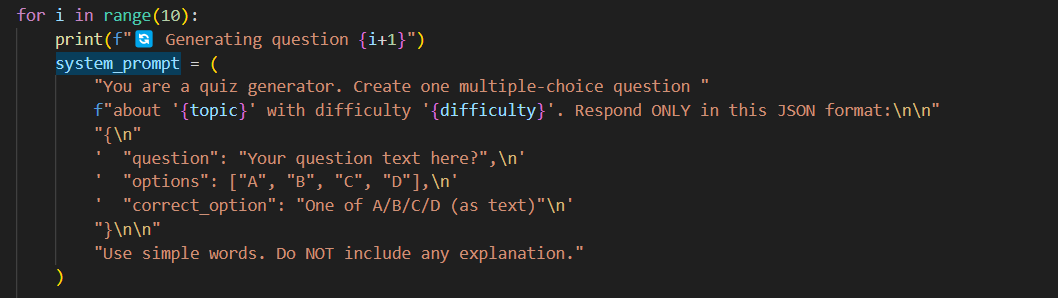
* **Activity 2.3: Adaptive Difficulty Logic**  
  A core feature of the backend, this function increased or decreased difficulty dynamically:
  + The current difficulty was adjusted up or down based on whether the user's previous answer was correct.
  + The order of difficulty ([easy, medium, hard]) was implemented using list indexing.
  + This adaptive logic kept the quiz challenging and personalized for each user.



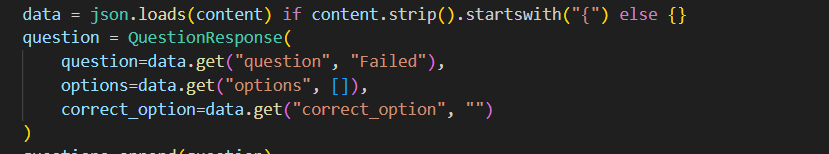
* **Activity 2.4: AI Integration (via quiz\_generator.py)**
  + The backend used requests.post() to call the Mistral model running on localhost via Ollama.



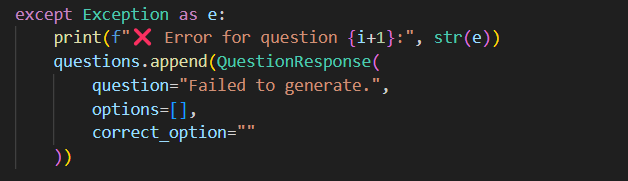
* + Each call included a structured prompt instructing Mistral to respond with one MCQ in JSON.



* + The JSON was parsed and validated before being returned to the user.



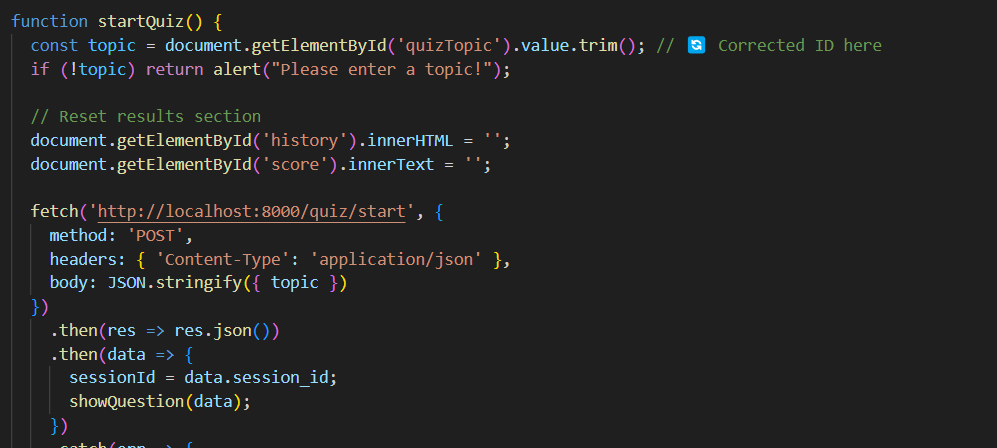
* + If malformed data was detected (e.g., missing keys or incorrect types), the system logged the error and continued with fallback responses.



**Milestone 3: Frontend Development**

This milestone focused on creating a responsive and visually engaging user interface that would allow users to interact seamlessly with the quiz engine. The frontend was built using a combination of HTML, CSS, and JavaScript, forming a dynamic layer that bridges the user experience with backend intelligence.

* **Activity 3.1: UI Layout and Design**  
  A structured and minimalist user interface was developed using HTML and styled with CSS to maintain clarity, responsiveness, and user engagement:
  + A topic input field was created at the landing page for the user to type their quiz subject.
  + A visually intuitive theme selection area was designed using card-style components to represent different themes.
  + The layout used a combination of Flexbox and CSS Grid for responsive design to ensure proper alignment across desktops, tablets, and smartphones.
  + Quiz questions were displayed in a clean container with properly spaced multiple-choice options to enhance readability.
* **Activity 3.2: Quiz Logic with JavaScript**  
  JavaScript was used to manage all interactive elements and handle API communication:
  + The script detected the user's theme and topic input, then made a fetch() request to /quiz/start to initialize a new session.



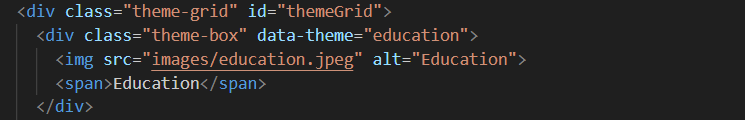
* + It dynamically parsed the question and rendered it into the DOM, ensuring each radio option was clickable and tracked.
  + When an answer was submitted, another API call to /quiz/answer was made, which returned the next question or the quiz result.
  + State management was handled in-memory on the frontend, tracking the question index and storing correct/incorrect flags.
  + Real-time feedback was shown immediately after answering, along with difficulty level indicators that reflected backend adjustments.
* **Activity 3.3: Result Summary Rendering**  
  Once the user finished the quiz, a complete visual breakdown of the session was shown:
  + A final score was calculated and prominently displayed in large font with styled containers.
  + The result panel iterated through each previous question, showing:
    - The original question
    - User’s selected option
    - The correct answer
    - Whether the user got it right or wrong, indicated with colored borders or icons
  + A "Try Again" or "Retake Quiz" button was added to improve reusability and user retention.
  + CSS animations were applied to the result screen for a smooth transition from the quiz panel.



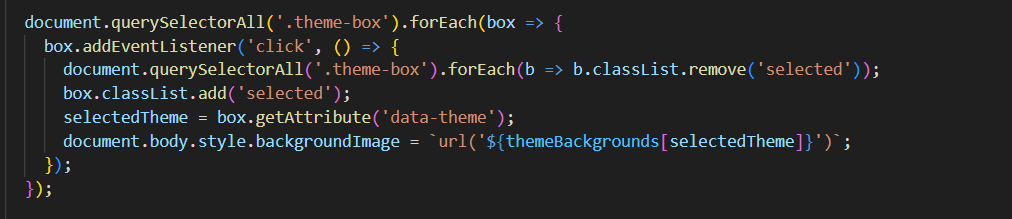
**Milestone 4: Theme Personalization and User Experience Enhancement**

This milestone aimed to enhance the quiz's aesthetic and user engagement level. While the core functionality of the quiz was already in place, these features focused on improving how users felt while using the platform by providing personalization, responsiveness, and visual feedback.

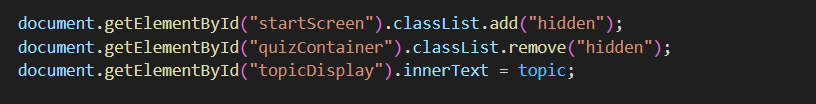
* **Activity 4.1: Theme Selector**  
  Implemented a visually interactive theme selection interface on the start screen. Each theme (such as Tech, Nature, Education, History, and World) was displayed as a styled box with background imagery. The theme selector allowed users to personalize their quiz environment:
  + Each theme was linked to a specific background image.



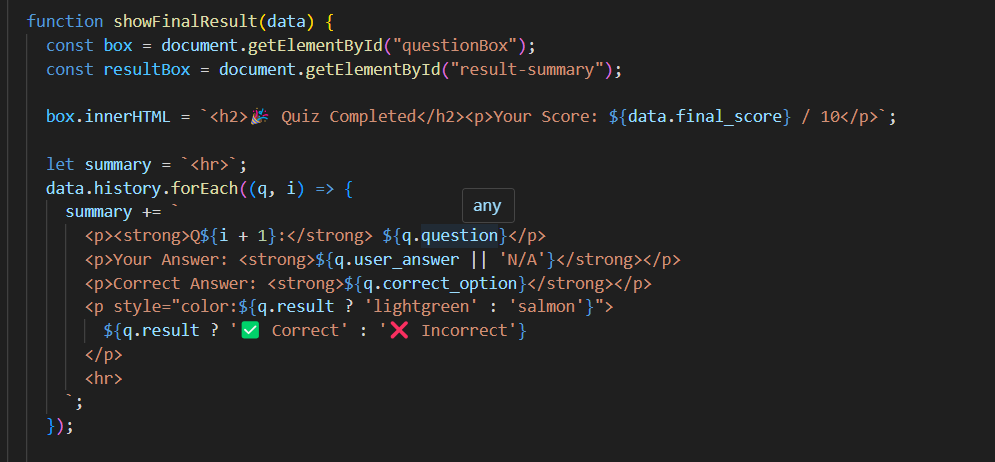
* + JavaScript was used to track which theme was selected and dynamically update the quiz interface background based on user preference.



* + Visual cues like border highlights and hover effects were added to improve interaction clarity.
* **Activity 4.2: Visual Transitions**  
  Introduced smooth transitions between quiz stages and theme changes:
  + CSS animations were applied when transitioning from the home page to the quiz page.
  + Fade-in/out effects ensured that theme changes and screen switches were fluid.
  + Background changes were handled by dynamically altering the CSS background-image property of the page, providing visual continuity.



* **Activity 4.3: Styling and Layout Enhancements**
  + All quiz containers were styled with rounded corners, soft shadows, and padding to enhance legibility.
  + Hover effects were added to options to give immediate visual feedback on hover.
  + The layout was built to be mobile-responsive, with grid-based design and media queries ensuring compatibility across devices.
  + Buttons were animated with subtle motion effects to enhance interactivity.
* **Activity 4.4: Styled Result Interface**  
  The result display was designed to clearly and visually represent user performance:
  + Each question listed in the result screen was color-coded: green for correct, red for incorrect.
  + The user's selected answer and the correct answer were both displayed.
  + A progress bar and score header showed overall performance.
  + The result summary was presented in a scrollable, responsive card format.
  + A retry button was implemented for immediate re-engagement.



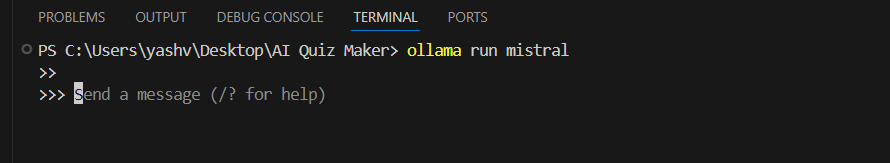
**Milestone 5: Testing and Deployment**

This milestone focused on successfully deploying, running, and testing the full AI Quiz Maker system in a local development environment. It ensured all components were properly integrated and functioning—from the AI model backend to the responsive frontend interface.

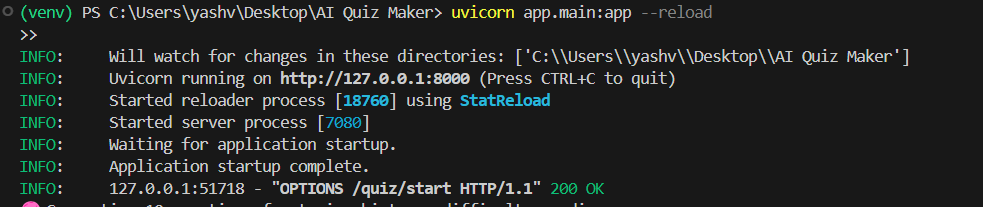
* **Activity 5.1: Local Setup Validation**  
  To begin testing, both the backend server and the AI model were set up locally:
  + The Ollama model server was launched using ollama run mistral, which made the Mistral model available at localhost:11434.
  + The FastAPI backend was started using uvicorn main:app --reload, serving the API endpoints at <http://127.0.0.1:8000>.
  + This step validated that both the AI model and backend API were live and ready for interaction.
* **Activity 5.2: End-to-End Workflow Validation**  
  This step verified that all parts of the application worked seamlessly together — from launching services to completing a quiz session.

**Step-by-step process:**

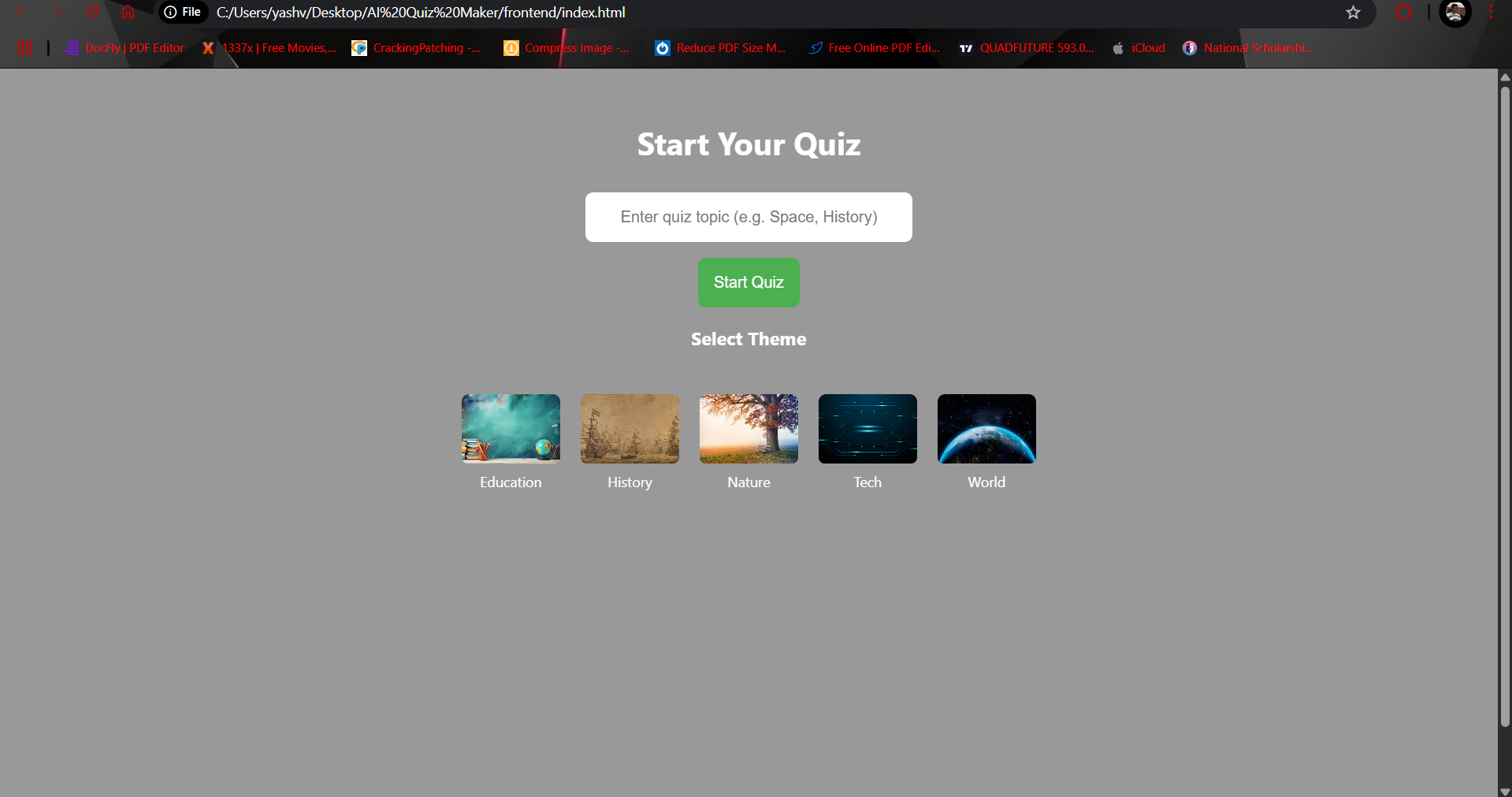
* + **Start the AI model:** Launched the Ollama container for the Mistral model using the terminal command ollama run mistral. This served the model on localhost:11434, making it ready to process quiz generation prompts.



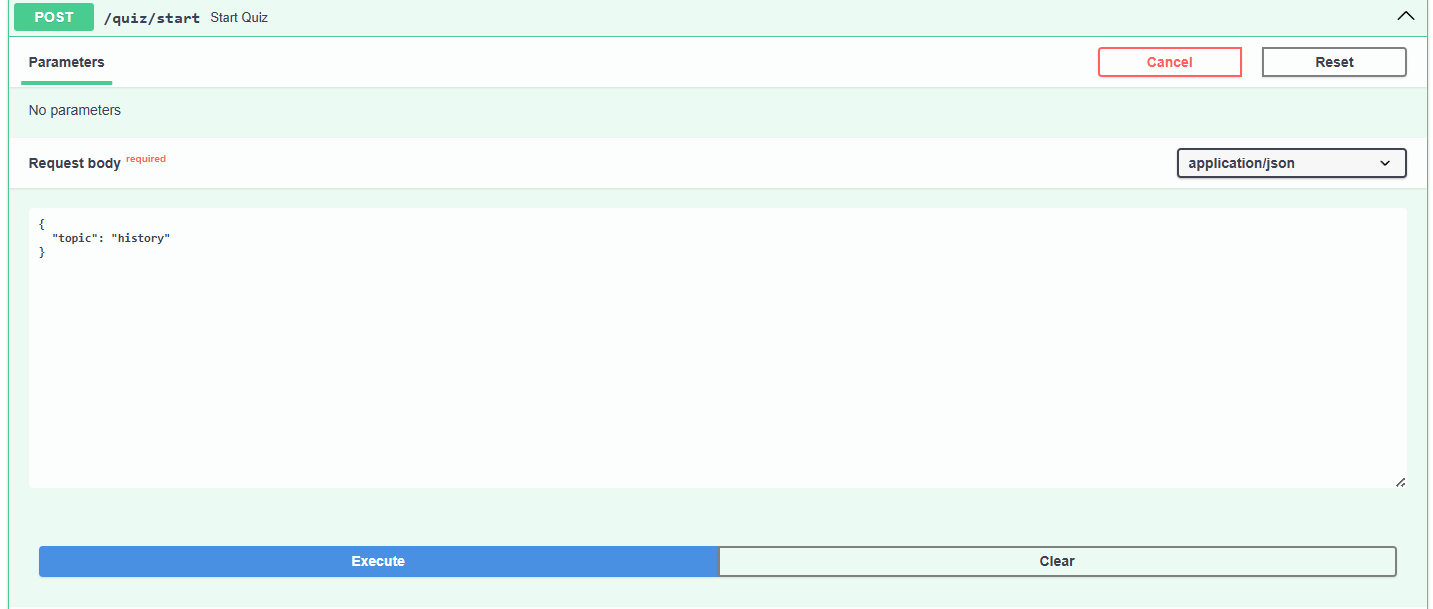
* + **Run the FastAPI backend:** Opened a second terminal and executed uvicorn main:app --reload. This served the backend at http://127.0.0.1:8000, which handled quiz sessions, adaptive logic, and communication with the AI model.



* + **Open the frontend:** Opened the index.html file in a web browser. The homepage appeared with a theme selector and a topic input field.



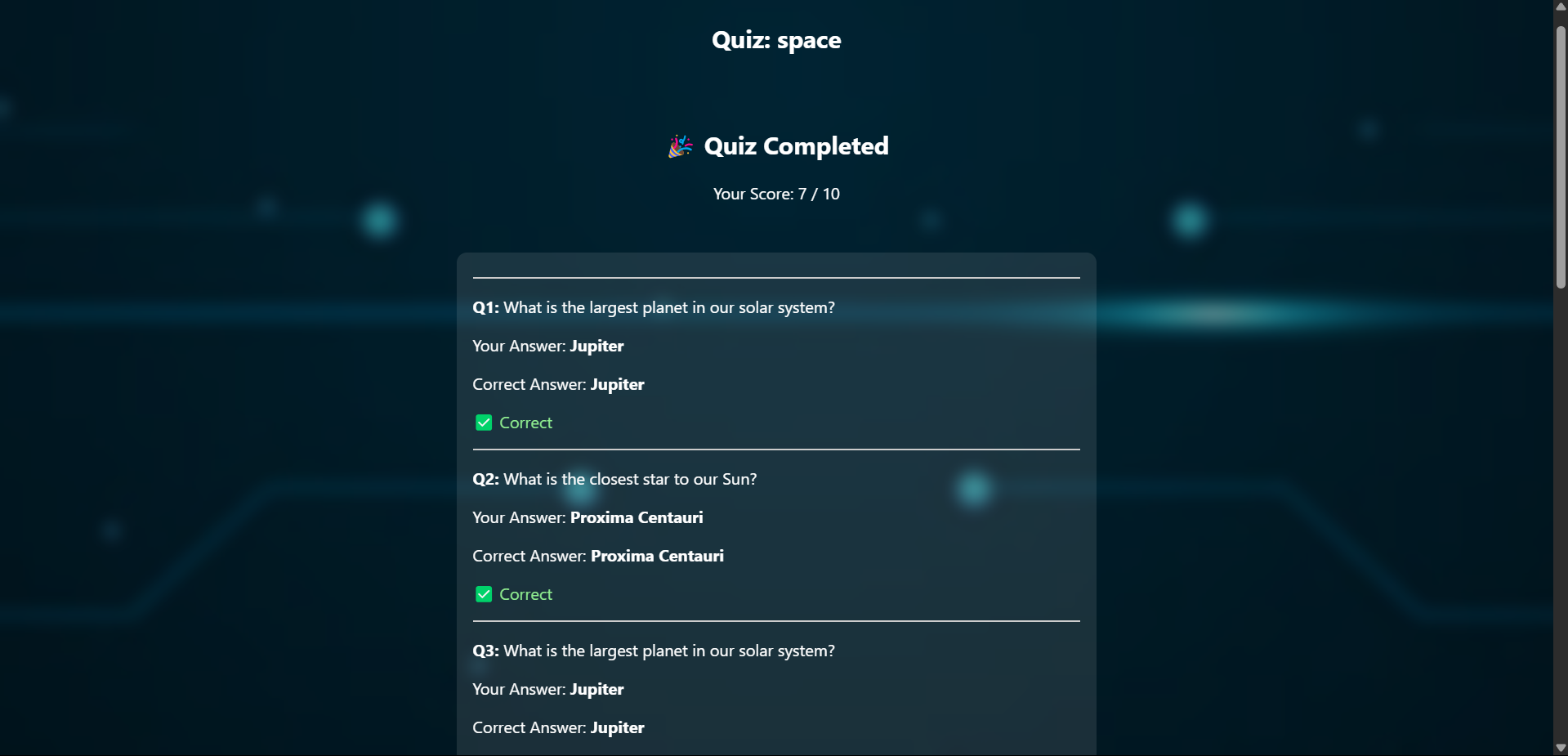
* + **Start a quiz session:** Entered a topic like “History” and selected a theme. Clicking the "Start Quiz" button triggered a JavaScript function that made a POST request to /quiz/start with the topic and difficulty level.



* + **AI generates questions:** The backend received the topic, contacted the Mistral model, and returned 10 JSON-formatted questions. These were dynamically loaded onto the frontend one at a time.



* + **User answers questions:** For each question, the user selected an option. On clicking “Next”, the frontend sent a POST request to /quiz/answer, carrying the selected answer and session ID.
  + **Backend processes and adapts difficulty:** The backend checked the answer, updated the difficulty, and returned the next question until all 10 were completed.
  + **View results:** After the final question, a summary screen was rendered showing the total score, each question, selected answers, correct answers, and color-coded feedback.



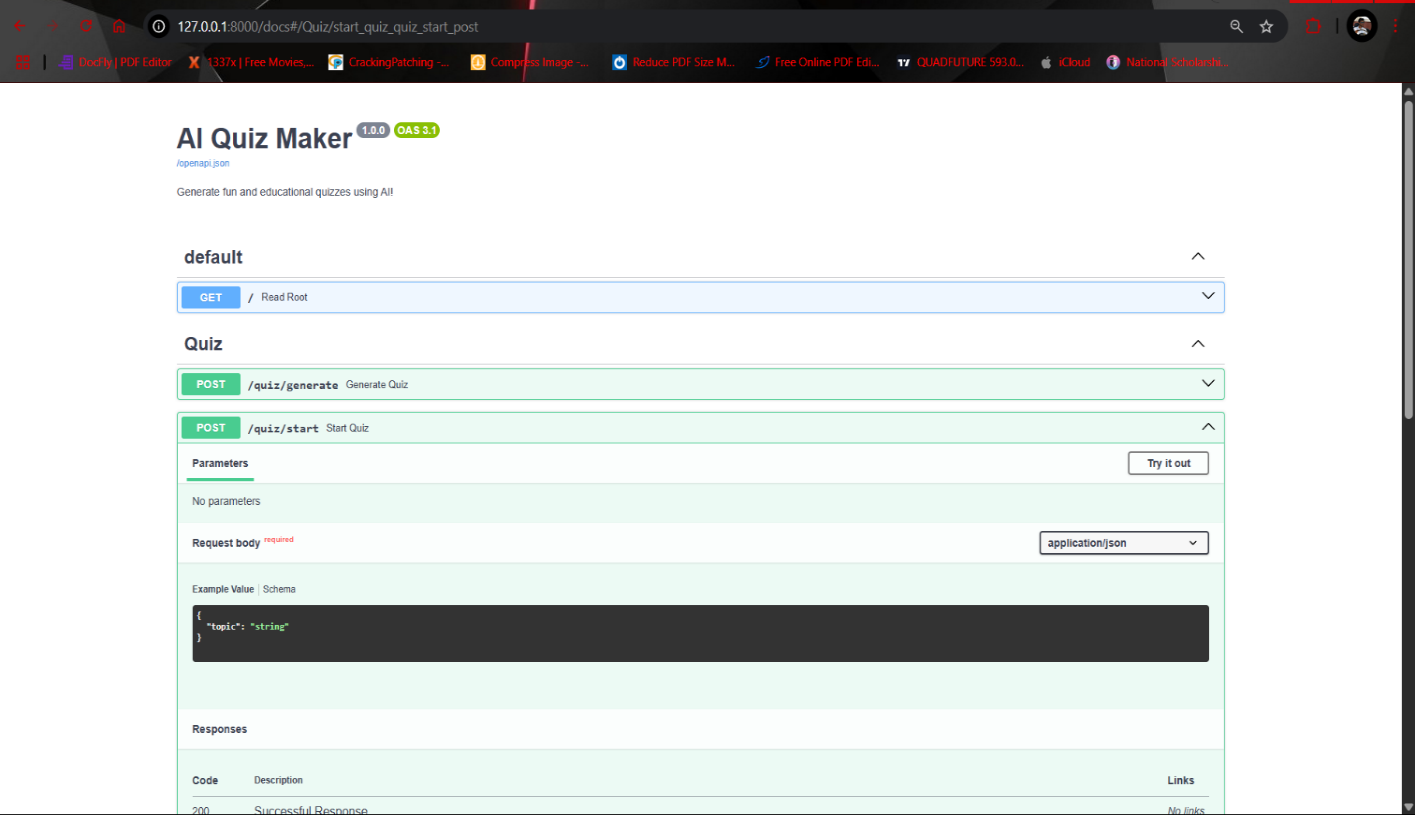
* + **Live validation:** Throughout the quiz, the terminal running FastAPI showed logs of user sessions, AI responses, and question handling, confirming backend activity. Console logs on the browser verified frontend behavior.

This validation loop ensured the user could complete a full quiz seamlessly, with the system accurately processing each step — from model generation to final scoring.

**Activity 5.3: Swagger API Testing and Validation**

Swagger UI, automatically generated by FastAPI, was used to thoroughly test and validate all backend endpoints and their respective schemas:

1. **Accessing Swagger Interface:** Opened the Swagger UI in a browser at http://127.0.0.1:8000/docs. This provided a user-friendly interface listing all available endpoints, their input schemas, and response formats.



1. **Testing /quiz/start Endpoint:** Submitted a POST request with a JSON payload including a topic like "Science". Verified that the response returned 10 questions with expected structure (question text, four options, correct answer).

**c - Testing /quiz/answer Endpoint:** Sent simulated user answers to the backend.

Observed whether the system returned the next question correctly and adjusted the difficulty level in response to right or wrong answers.

1. **Schema Validation:** Ensured each endpoint adhered to the Pydantic models defined in the backend. Swagger automatically validated field types, required fields, and returned helpful error messages for malformed inputs.
2. **Error Handling and Logging:** Purposefully submitted invalid data (e.g., empty topic, missing fields) to test backend validation. Observed correct HTTP error responses and helpful log outputs in the terminal.
3. **Model Response Verification:** Compared AI-generated responses to the expected JSON structure. Verified that the backend correctly parsed and presented the data even when the AI returned slightly varied formats.

This activity ensured not only the functional correctness of the API layer but also the integrity of schema validation, making the backend robust and reliable for real-time communication with the frontend and AI engine.

**Activity 5.4: Manual QA & Screenshots**

Manual Quality Assurance (QA) was conducted to ensure the complete functionality of application from both backend and frontend perspectives. The testing involved simulating real user interactions and validating correct behavior at each stage:

* 1. **Frontend Flow Testing:** Opened the index.html in a browser and conducted multiple quiz sessions.
     + Checked that the theme selector responded to user clicks and visually changed the background.
     + Ensured topic input was properly captured and submitted to the backend.
     + Verified that questions rendered in sequence and updated dynamically as users answered.
     + Examined visual feedback for correct and incorrect answers using color indicators.
  2. **Visual Design Review:** Confirmed that the UI components were styled correctly across devices.
     + Checked spacing, font consistency, and alignment using browser dev tools.
     + Ensured responsiveness by adjusting the window size to test tablet and mobile behavior.
  3. **Dynamic Logic Confirmation:** Traced the adaptive difficulty feature.
     + Manually answered questions right/wrong in sequence and verified difficulty changed (easy → medium → hard and back).
     + Validated that the displayed difficulty level matched backend behavior..
  4. **Data Verification:** Cross-checked browser console logs and API responses.
     + Ensured that Mistral’s JSON was parsed correctly.
     + Verified that incorrect or malformed questions were handled gracefully (fallback text or error notice).

This manual QA process, supported by real-time visual documentation, demonstrated the robustness of the system and provided proof of complete integration. The screenshots will be appended to the report as evidence of working features, validated logic, and user-friendly design.

**Activity 5.5: Terminal Logs and Console Feedback**

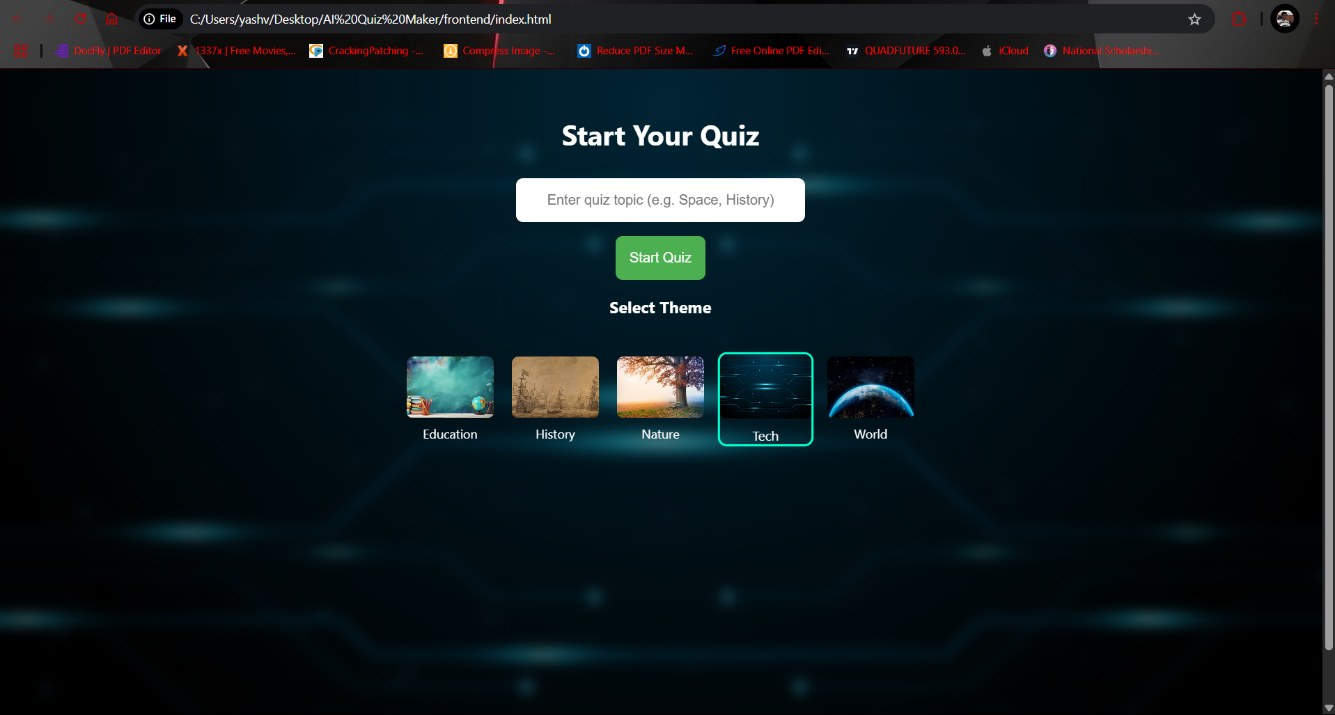
Terminal output and browser console logs were closely monitored throughout development and testing to validate system behavior and troubleshoot any issues in real-time:

1. **FastAPI Logs:**
   * + Each incoming API request was logged with timestamps, endpoint names, and payload.
     + Verified that the /quiz/start and /quiz/answer endpoints were hit with the correct parameters.
     + Debug print statements in the backend confirmed when Mistral was called and returned results.
     + Session IDs, question order, and score updates were also printed for transparency.
2. **Ollama Logs (Mistral Model):**
   * + The Ollama terminal displayed loading indicators for model start-up and memory allocation.
     + Each prompt sent by the backend was printed along with the AI's response.
     + Errors like model timeouts or malformed replies were immediately visible, helping with prompt tuning.
3. **Browser Console Logs:**
   * + Used console.log() in script.js to monitor frontend behavior.
     + Tracked outgoing fetch requests to API endpoints and their responses.
     + Helped diagnose issues like empty topic input, network delays, and data parsing problems.
4. **Cross-verification:**
   * + Verified that each step (model call → response parsing → frontend render) completed successfully by observing all three logs in tandem.
     + This full-stack visibility allowed for rapid debugging and confident confirmation of working logic.

Logging at each level of the application was essential for both development efficiency and post-deployment assurance. It ensured traceability of every quiz session and confirmed that each system component was reliably executing its responsibilities.

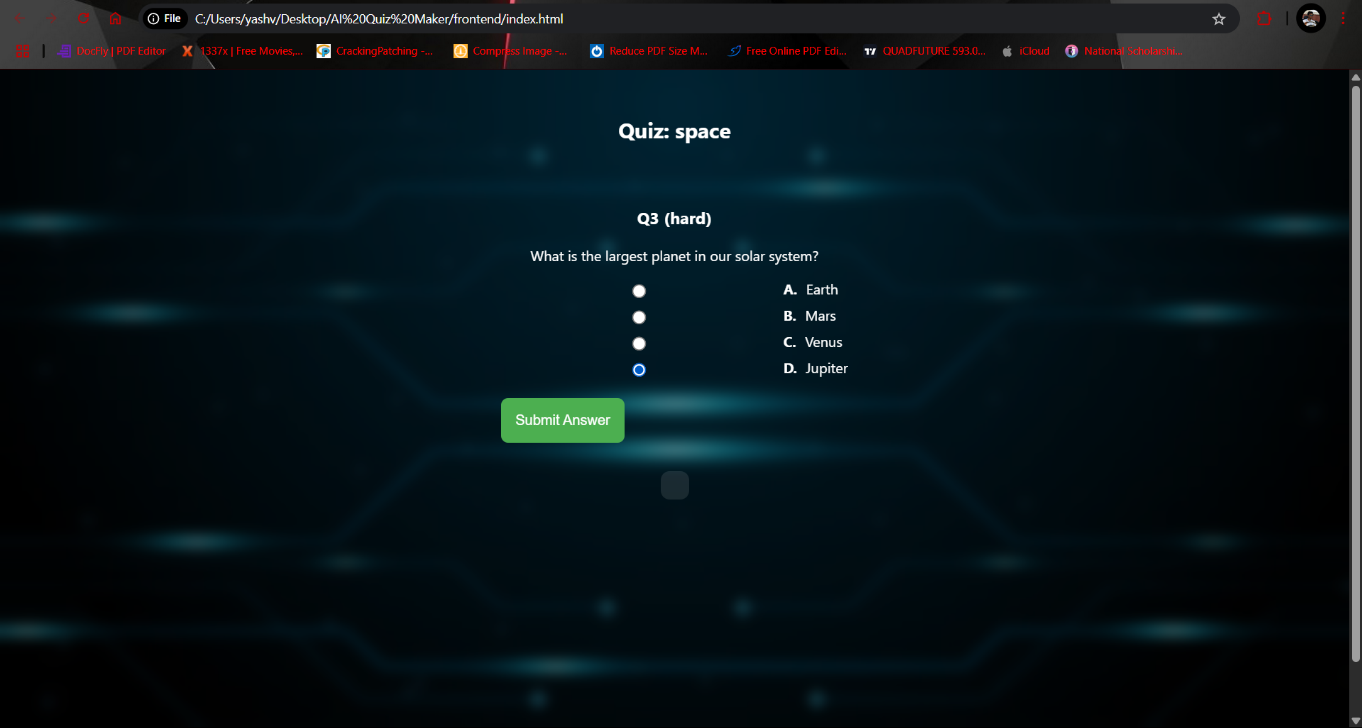
This milestone confirms the project is deployment-ready and fully functional in a local setup. It verifies every stage of the user journey, from initial topic input to final result summary, with AI integration and dynamic UI features performing as intended.

**Project Output –**



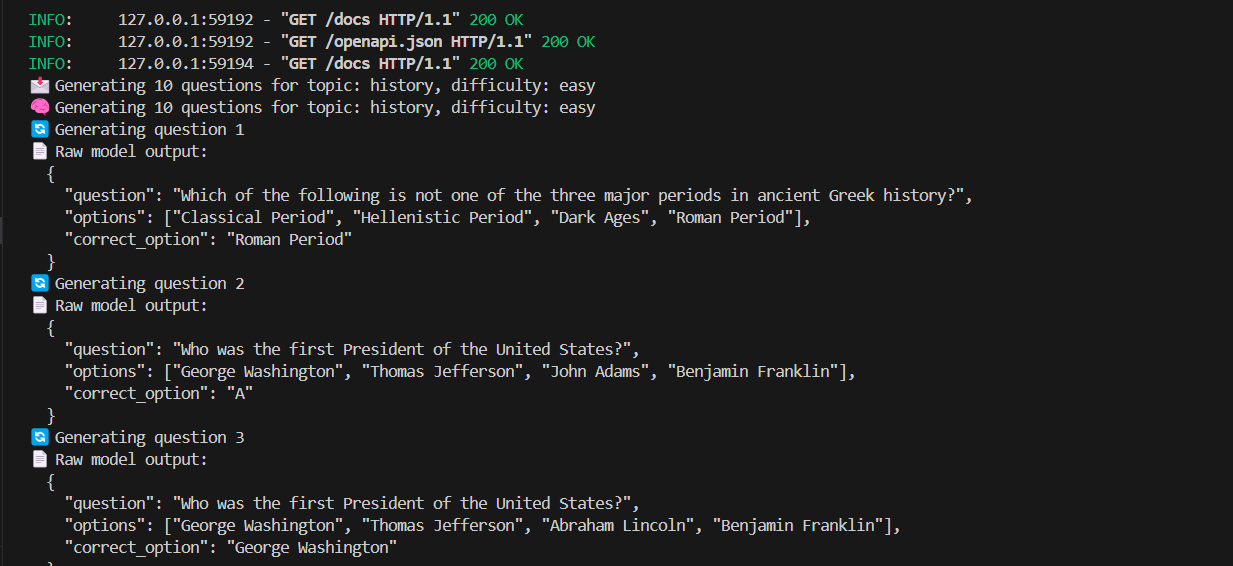
This is the landing page of the AI Quiz Maker interface. It demonstrates:

* A clean, interactive **user input form** where users can enter any quiz topic (e.g., “Space”, “History”).
* A **theme selection panel** with five visual themes: Education, History, Nature, Tech, and World. These affect the background aesthetics of the quiz.
* The **Start Quiz button** triggers the /quiz/start backend call with the selected topic and theme.  
  This screen shows how users can personalize the quiz environment before starting.



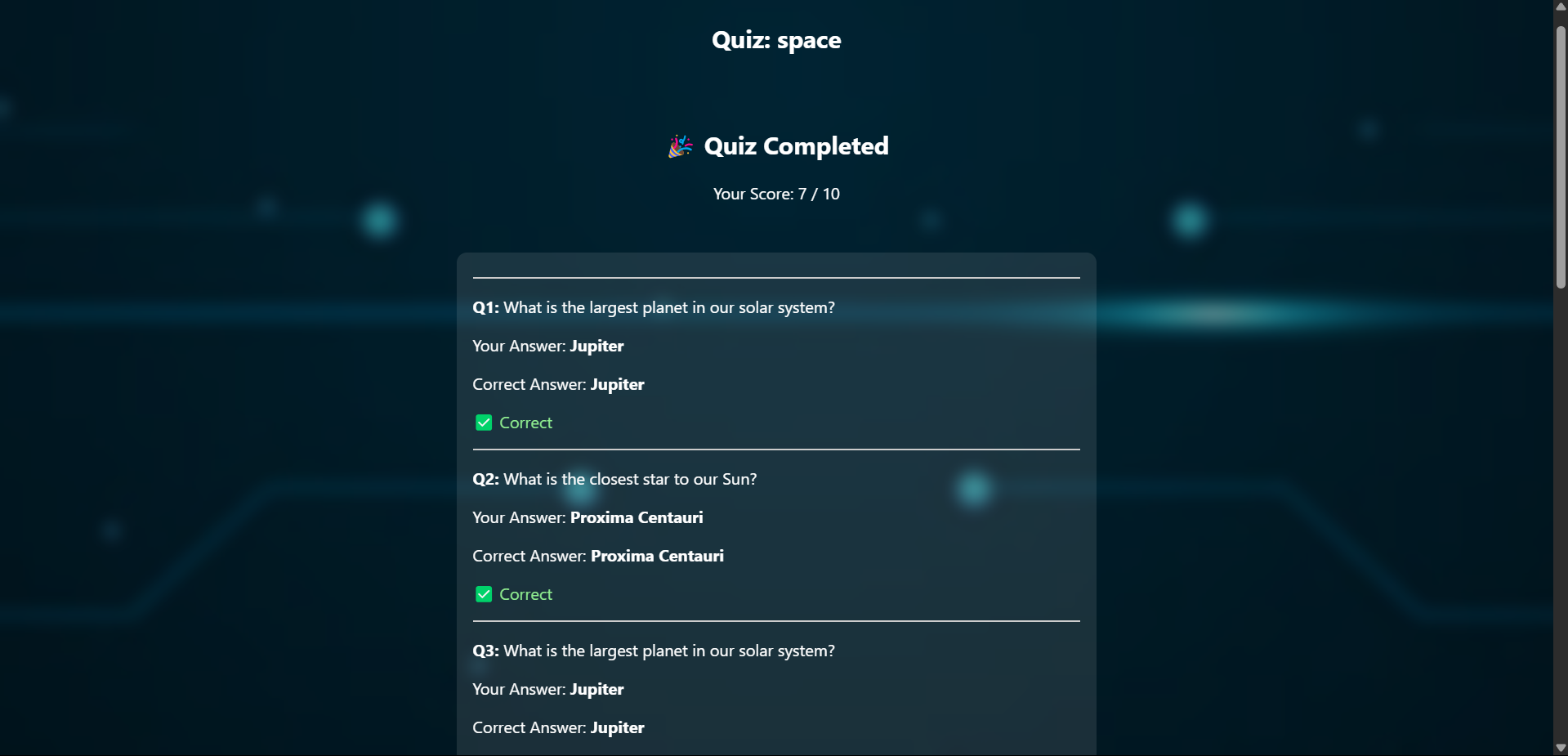
This page displays a live quiz question interface in action. It includes:

* The topic and current difficulty level (Q3 (hard)).
* A multiple-choice question with **radio buttons** for answer selection.
* A **Submit Answer** button that triggers a request to /quiz/answer endpoint and advances to the next question.  
  This interface highlights real-time difficulty tracking and question rendering based on the adaptive logic.



This backend terminal output confirms:

* Successful FastAPI server operations (GET /docs, openapi.json).
* Backend logs of topic selection (Generating 10 questions for topic: history) and the **current difficulty level**.
* **Raw model outputs** from Mistral via Ollama showing:
  + The generated question
  + Options list
  + Correct answer returned in structured JSON  
    This proves that the AI is generating quiz questions dynamically and returning them correctly for parsing and display.



This screen appears after the quiz is completed and displays:

* The **final score** (Score: 7 / 10)
* A **question-by-question breakdown** with:
  + The original question text
  + The user’s selected answer
  + The correct answer
  + A visual feedback indicator ( Correct or Incorrect)  
    This summary demonstrates the **real-time evaluation** system, showing both correctness and reinforcing learning through detailed feedback.

**Conclusion:**

The **AI Quiz Maker** project successfully demonstrates how generative AI can be integrated into modern educational tools to deliver dynamic, adaptive, and personalized learning experiences. Designed as a full-stack application, it combines the robustness of a FastAPI backend with an interactive, visually themed frontend built using HTML, CSS, and JavaScript. At its core, the system uses the **locally hosted Mistral model via Ollama** to generate topic-specific multiple-choice questions in real time, enabling seamless quiz creation on any subject chosen by the user.

**What Was Achieved**

The project achieved all of its key objectives. It enables users to:

* Enter any topic and receive a tailored 10-question quiz
* Experience **adaptive difficulty** where questions adjust in complexity based on user performance
* Select from **multiple visual themes** to personalize the interface
* Receive **real-time scoring and feedback** after each question
* View a detailed final result summary with performance indicators

The backend efficiently handles session management, prompt generation, and answer evaluation, while the frontend provides a smooth and responsive user interface. This integration creates a complete learning tool that is both technically sound and user-friendly.

**Challenges Faced**

Several technical and design challenges were encountered during development:

* **Model Output Consistency:** Ensuring that the Mistral model returned well-structured JSON required prompt tuning and fallback logic in case of malformed responses.
* **Difficulty Adaptation Logic:** Implementing real-time adjustment of difficulty based on user performance had to be carefully balanced and tested.
* **Frontend-Backend Synchronization:** Maintaining consistent state and transitions between questions, especially under changing difficulty levels, required rigorous debugging.
* **Theme Integration:** Ensuring smooth background transitions and responsive design across all devices added extra layers to the frontend complexity.

Despite these hurdles, iterative testing and modular coding allowed each issue to be resolved efficiently.

**Learning Outcomes**

This project provided a rich, hands-on learning experience in the following areas:

* **AI Integration:** Practical understanding of working with large language models (LLMs) locally using Ollama and Mistral.
* **API Design:** Building robust, testable RESTful APIs using FastAPI and Pydantic schemas.
* **Frontend Development:** Creating responsive UIs, handling API calls with JavaScript, and implementing real-time interactivity.
* **System Architecture:** Designing scalable, modular structures with a clear separation of concerns between UI, logic, and AI layers.
* **Problem Solving:** Overcoming unexpected edge cases, malformed outputs, and ensuring fault tolerance in real-time applications.

**Future Scope**

The current version lays a strong foundation for multiple future enhancements, including:

* **User Authentication:** Allowing users to create profiles and track their quiz history.
* **Leaderboard and Ranking:** Enabling competitive features for gamification.
* **PDF Export:** Providing downloadable quiz reports for review and sharing.
* **Multilingual Support:** Extending the AI prompt to support quizzes in different languages.
* **Mobile App Integration:** Building a cross-platform mobile version using frameworks like Flutter or React Native.

In summary, AI Quiz Maker is not just a demonstration of technical skills, but also a scalable, real-world application that showcases how AI and web technologies can be combined to transform education into a more interactive, accessible, and adaptive experience.