MAP INNOVATIVE

Quiz Management System

22BCE510	22BCE530	22BCE545
AARSHIT JOLAPARA	PRIYANSH PATEL	JATIN VARYANI

API's used in this project :

- **Auth Microservice**
- **Quiz Microservice**
- **AI Microservice**

Auth Microservice

1. Overview

This microservice handles user authentication and token based authorization. It enables secure management of user sessions and protects access to sensitive operations. All endpoints are designed to work with JSON requests and responses.

Base URL: "/api/auth"

2. Endpoints

2.1 Register User

This endpoint allows the creation of a new user account. It accepts a username and password, validates the input, and ensures that the username is unique within the system. Upon successful registration, a confirmation response is sent to the client.

URL: "/api/auth/register"
Method: POST
Description: Registers a new user in the system.

Request Body

```
{
  "username": "string",
  "password": "string"
}
```

Responses
201 Created: User successfully registered.
400 Bad Requests: Invalid input or user already exists.

2.2 Login User

This endpoint authenticates an existing user by verifying their email and password. Upon successful authentication, it generates and returns a pair of tokens: an access token for immediate use and a refresh token for extended sessions.

URL: “/api/auth/login”
Method: POST
Description: Authenticates a user and provides a JWT token.

Request Body
```
{
    "email": "string",
    "password": "string"
}
```

Responses
200 OK: Returns tokens.
```
{
    "accessToken": "string",
    "refreshToken": "string"
}
```

401 Unauthorized: Invalid credentials.
400 Bad Requests: Missing or incorrect input data.

2.3 Logout User

This endpoint terminates a user session by invalidating their tokens. It ensures the access token can no longer be used to access protected resources, securing the session.

URL: "/api/auth/logout"
Method: POST

Description: Logs out a user by invalidating their token.

Responses
- 200 OK: User successfully logged out.
- 400 Bad Request: Invalid or missing token.

2.4 Refresh Access Token

This endpoint is used to obtain a new access token by providing a valid refresh token. This helps users maintain their session without needing to reauthenticate.

URL: "/api/auth/refreshtoken"
Method: POST
Description: Refreshes the access token using a valid refresh token.

Responses
200 OK: Returns a new access token.

```
{
    "accessToken": "string"
}
```

403 Forbidden: Invalid or expired refresh token.

2.5 Verify Token

This endpoint checks the validity of a provided access token. It ensures the token is still active and has not been tampered with or expired.

URL: "/api/auth/verify"
Method: POST
Description: Verifies the validity of the provided access token.

Request Body

```
{
  "accessToken": "string"
}
```

Responses
200 OK: Token is valid.
401 Unauthorized: Token is invalid or expired.

2.6 Get User

This endpoint retrieves the details of the authenticated user. The request must include a valid access token for authorization.

URL: "/api/auth/getUser"
Method: POST
Description: Retrieves details of the authenticated user using an access token.

Request Body

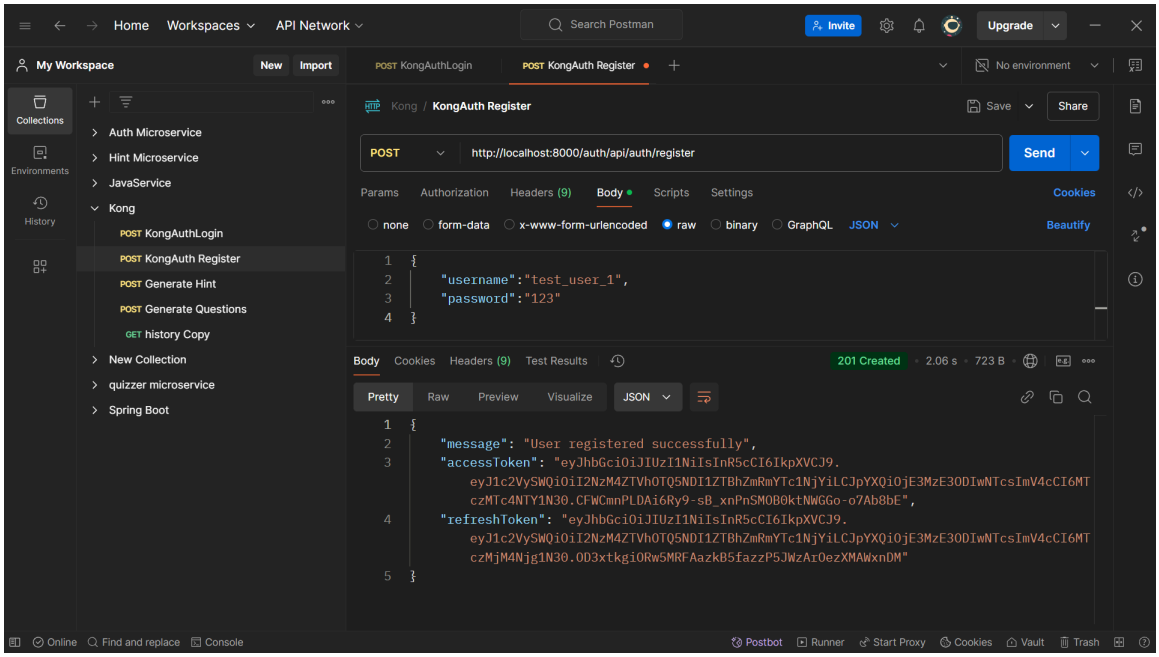
```
{
    "username": "string"
}
```

Responses
200 OK: Returns user details.

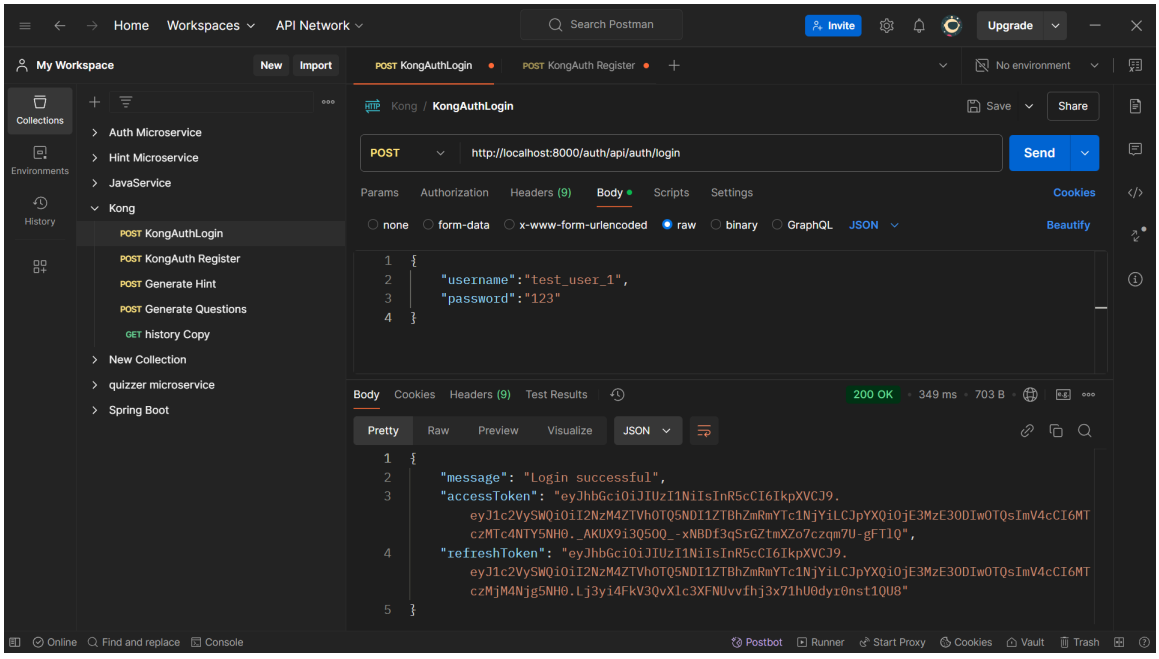
```
{
    "id": "string",
    "username": "string"
}
```

401 Unauthorized: Invalid or missing access token.

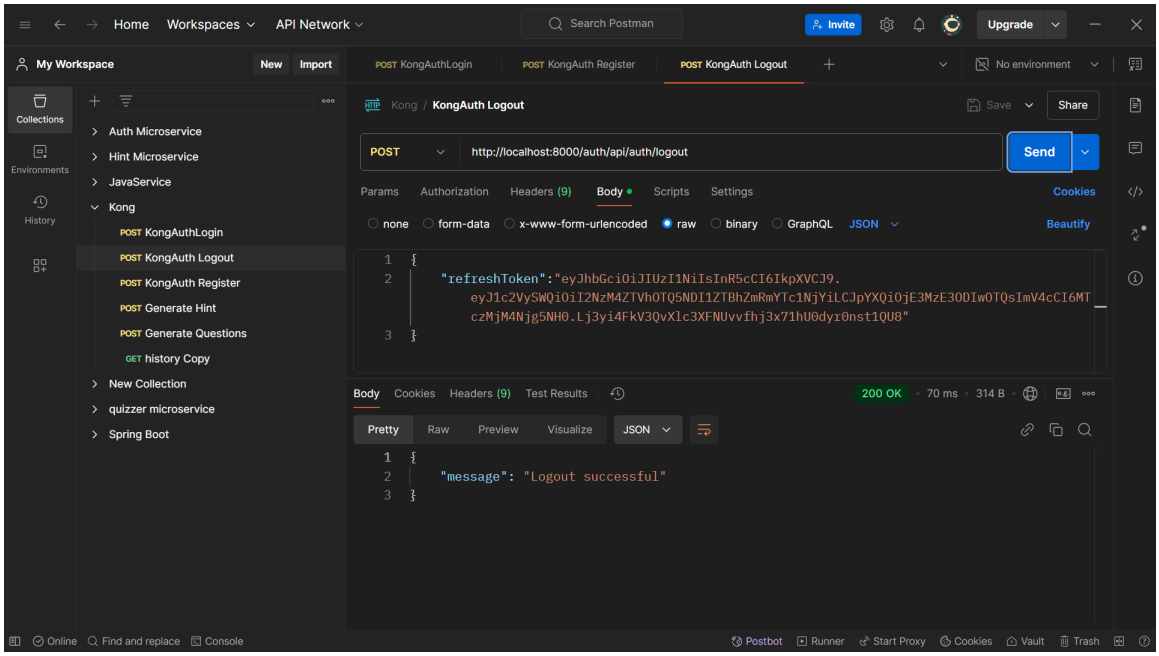
3. Screen Shots:



User Registration



User Login


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
POST KongAuthLogin
POST KongAuth Register
POST KongAuth Logout
No environment
Collections
Environments
History
Auth Microservice
Hint Microservice
JavaService
Kong
POST KongAuthLogin
POST KongAuth Logout
POST KongAuth Register
POST Generate Hint
POST Generate Questions
GET history Copy
New Collection
quizzer microservice
Spring Boot
Kong / KongAuth Logout
Save
Share
POST
http://localhost:8000/auth/api/auth/logout
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"refreshToken":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ1c2VySWQiOiI2NzM4ZTVhOTQ5NDI1ZTBhZmRmYTc1NjYiLCJpYXQiOjE3MzE3ODIwOTQsImV4cCI6MT
czMjM4Njg5NH0.Lj3yi4FkV3QvXlc3XFNUvvfhj3x71hU0dyr0nst1QU8"
}
Body
Cookies
Headers (9)
Test Results
200 OK
70 ms
314 B
Pretty
Raw
Preview
Visualize
JSON
{
"message": "Logout successful"
}
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

User Logout

Quiz Microservice

1. Overview

The Quiz Microservice API facilitates the management of quizzes, enabling functionalities such as creating, submitting, retrieving quiz histories, and retrying quizzes. It ensures security by requiring valid JWT tokens for all operations.

Base URL: "/api/quiz"
Purpose: Provide secure quiz management for authenticated users.

2. Endpoints

2.1 Create a Quiz

This endpoint enables the creation of a new quiz. It accepts detailed quiz information such as title, grade, subject, total questions, maximum score, difficulty level, and the list of questions. Each question includes a unique identifier and the correct answer. The quiz is associated with the authenticated user.

URL: "/api/quiz/create"
Method: POST

Request Body

```
{
  "title": "string",
  "grade": "integer",
  "subject": "string",
  "totalQuestions": "integer",
  "maxScore": "integer",
  "difficulty": "string",
  "questions": [
    {
      "questionId": "string",
      "correctAnswer": "string"
    }
  ]
}
```

"title": The name of the quiz.
"grade": The grade level targeted by the quiz (e.g., 5, 10).
"subject": The subject of the quiz (e.g., Mathematics, Science).

"totalQuestions": Total number of questions in the quiz.
"maxScore": Maximum achievable score for the quiz.
"difficulty": The difficulty level of the quiz (e.g., Easy, Medium, Hard).
"questions": An array of question objects containing:
"questionId": A unique identifier for each question.
"correctAnswer": The correct answer for the question.

Responses
201 Created: The quiz is successfully created and stored.
500 Internal Server Error: Quiz creation failed due to server issues.

2.2 Submit Quiz

This endpoint allows users to submit their answers for a quiz. The responses are evaluated, and the user's score and performance status are calculated.

URL: "/api/quiz/submit"
Method: POST

Request Body

```
{
  "quizId": "string",
  "responses": [
    {
      "questionId": "string",
      "userResponse": "string"
    }
  ]
}
```

"quizId": The unique identifier of the quiz being submitted.
"responses": An array of response objects containing:
"questionId": The unique identifier of the question.
"userResponse": The answer provided by the user.

Responses
200 OK: Quiz submission is successful, returning the user's score and performance evaluation.
404 Not Found: The specified quiz does not exist.
500 Internal Server Error: Quiz submission failed due to server issues.

2.3 Get Quiz History

This endpoint retrieves the history of quizzes taken by the authenticated user. Optional query parameters can be used to filter results based on specific criteria such as grade, subject, marks, or a date range.

URL: "/api/quiz/history"
Method: GET

Query Parameters
"user": (string) Username to filter quizzes. Defaults to the current authenticated user.
"grade": (number) Grade level to filter quizzes.
"subject": (string) Subject name to filter quizzes.
"marks": (number) Filter quizzes by marks obtained.
"from": (date) Start date for filtering.
"to": (date) End date for filtering.

Responses:
200 OK: Returns a list of quizzes matching the specified criteria, including details like quiz title, marks scored, and date taken.
404 Not Found: No quiz history matches the given criteria.
500 Internal Server Error: History retrieval failed due to server issues.

2.4 Retry Quiz

Fetches a quiz for retrying based on the provided quiz identifier. The endpoint also provides details of the user's previous submissions, if available.

URL: "/api/quiz/retry/:quizId"
Method: GET

Responses
200 OK: The quiz data and submission history are returned, enabling the user to retry.
404 Not Found: The quiz specified by the "quizId" does not exist.
500 Internal Server Error: Quiz retrieval for retrying failed due to server issues.

3. Usage Recommendations

- **3.1 Security**
  - Ensure all requests include a valid JWT token for authentication.
  - Use HTTPS for secure communication between the client and the API.
  - Validate input data on both the client and server sides to avoid security vulnerabilities.

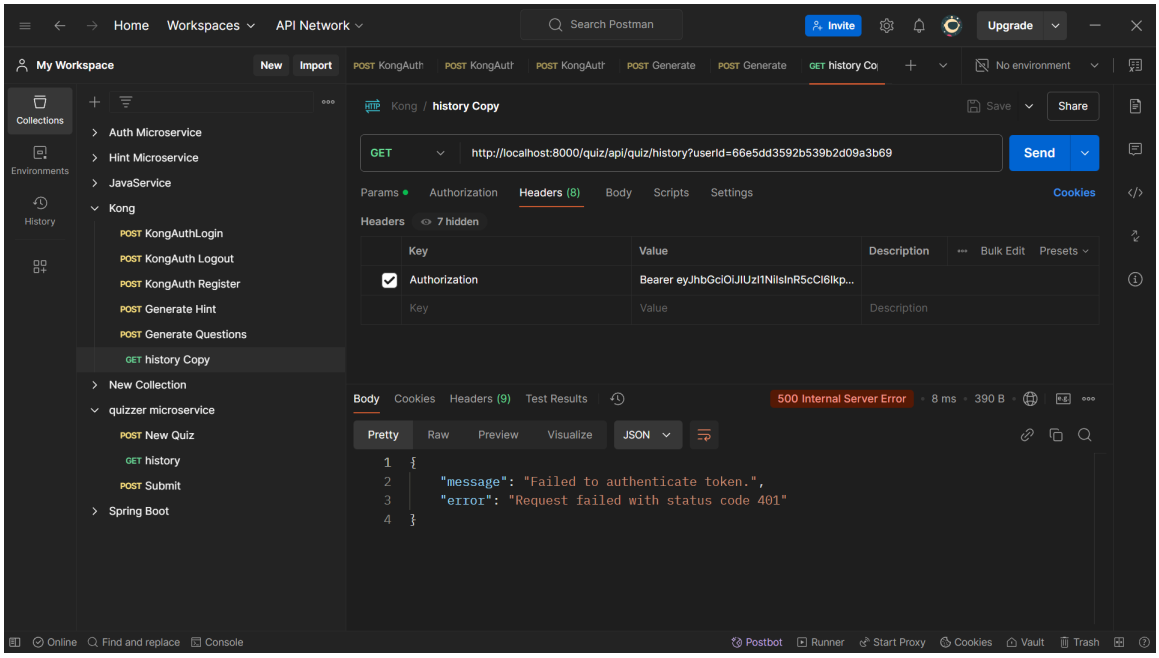
- **3.2 Token Management**
  -
  - Keep tokens secure by storing them in HTTPonly cookies or secure storage mechanisms.
  - Refresh tokens periodically to maintain session continuity.

- **3.3 Error Handling**
  -
  - Implement client side logic to gracefully handle error responses:
    - Retry actions for "500 Internal Server Error" responses.
    - Display appropriate error messages for "404 Not Found" or other status codes.

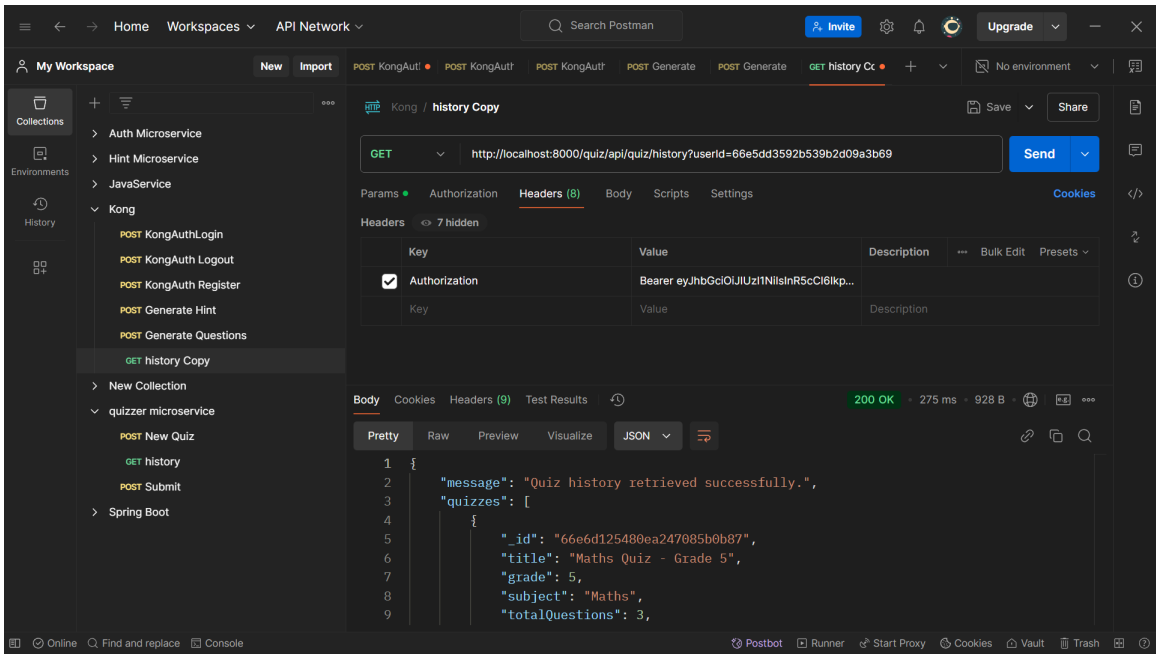
- **3.4 Performance and Scalability**
  - Optimize quiz creation by limiting the number of questions and avoiding excessively large payloads.
  - Use pagination for retrieving quiz history when dealing with large datasets.

- **3.5 Testing and Monitoring**
  - Test all endpoints using tools like Postman or automated test suites.
  - Monitor API performance and errors using analytics tools to detect potential issues.

4. Screen Shots:



Quiz History (Error for unauthorized user access)



Quiz History Fetched Successfully

AI Microservice

1. Overview

The AI Microservice API enables the dynamic generation of AI for specific questions. The AI is crafted based on the question's complexity level and its associated answer by leveraging the Groq API.

Base URL: /generatehint/
Purpose: Provide dynamically generated AI to enhance quiz experiences or educational platforms.

2. Endpoints

2.1 Generate Hint

Generates a hint for the provided question by considering the complexity level and answer. The service dynamically interacts with the Groq API to produce contextual AI.

URL: /generatehint/
Method: POST

Request Body

```
{
  "question": "string",
  "complexity": "string",
  "answer": "string"
}
```

Parameters:

question: A string containing the question text for which a hint is required.

complexity: Indicates the difficulty level of the question. Accepted values are:

- Easy
- Medium
- Hard

answer: The correct answer to the question, which helps refine the generated hint.

Responses

200 OK: Hint generation is successful. The response includes the generated hint:

```
{
    "hint": "string"
}
```

hint: The dynamically generated hint for the given question.

400 Bad Request: Indicates issues with the request, such as missing or invalid parameters.

500 Internal Server Error: The hint could not be generated due to server or API interaction issues.

Sure! Here's the detailed API specification for the "2.2 Generate Question" service, following the format you provided:

2.2 Generate Question

Generates a set of questions based on the provided input, considering the topic, grade level, subject, and difficulty.

URL: "/generatequestion/"
Method: "POST"

Request Body:

```
{
    "title": "string",          // Title of the topic for which questions are to be generated
   "grade": "string",        // Grade level for the questions (e.g., "5")
   "subject": "string",      // Subject of the questions (e.g., "Science")
   "totalQuestions": "int",  // Number of questions to generate (e.g., 5)
       "difficulty": "string"       // Difficulty level of the questions: "Easy", "Medium", or "Hard"
}
```

Parameters:
"title": A string representing the topic for which the questions should be created.
"grade": The grade level for the target audience (e.g., "5").
"subject": The subject under which the questions fall (e.g., "Science").
"totalQuestions": The total number of questions to generate.

"difficulty": The difficulty level of the questions. Acceptable values are:
"Easy"
"Medium"
"Hard"

Response:

200 OK: If the question generation is successful, the response includes the generated questions in a structured format.

{
"questions": "[\n {\n \"questionText\": \"What is the main function of the roots of a plant?\",\n \"options\": [\"to absorb sunlight\", \"to release oxygen\", \"to absorb water and minerals\", \"to produce fruits\"],\n \"correctAnswer\": \"c\"\n },\n {\n \"questionText\": \"Which part of the plant catches the sunlight needed for photosynthesis?\",\n \"options\": [\"leaves\", \"roots\", \"stems\", \"flowers\"],\n \"correctAnswer\": \"a\"\n },\n {\n \"questionText\": \"What do leaves typically do with the water and nutrients they absorb?\",\n \"options\": [\"store them for later use\", \"use them immediately\", \"release them into the air\", \"spread them to other plants\"],\n \"correctAnswer\": \"b\"\n },\n {\n \"questionText\": \"Why do plants need sunlight, water, and air for photosynthesis?\",\n \"options\": [\"to grow taller\", \"to produce color\", \"to produce food\", \"to make their stems stronger\"],\n \"correctAnswer\": \"c\"\n },\n {\n \"questionText\": \"What is the process by which plants make their own food?\",\n \"options\": [\"respiration\", \"photosynthesis\", \"transpiration\", \"circulation\"],\n \"correctAnswer\": \"b\"\n }\n]"
}

400 Bad Request: Indicates issues with the request, such as missing or invalid parameters.

500 Internal Server Error: If there was an issue generating the questions due to server or API interaction problems.

This format allows the generation of topic based questions tailored for a specific grade and subject, with the ability to specify the number and difficulty of the questions.

3. Best Practices and Recommendations

- **3.1 Security**
  - All requests require token based authentication. Ensure that tokens are included in the headers of each request to protect the endpoint from unauthorized access.
  - Use HTTPS for secure communication and prevent data interception.

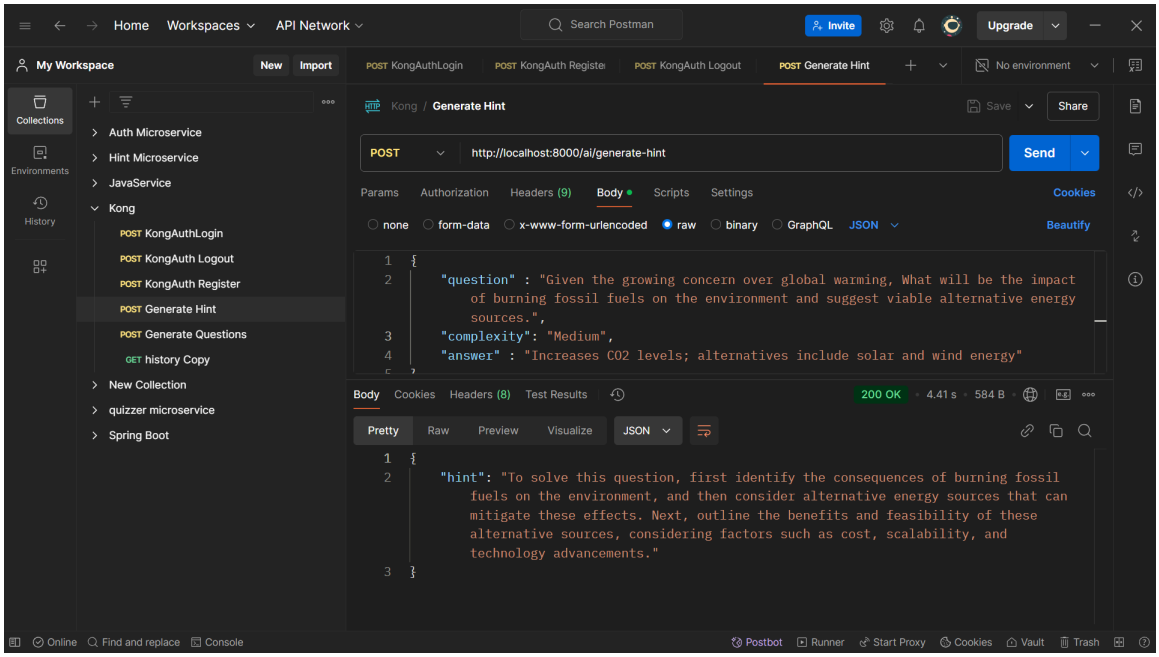
- **3.2 Request Guidelines**
  - Validate input fields on both the client and server sides to ensure proper data formatting and avoid invalid requests.
  - Provide meaningful and complete values for all parameters (question, complexity, and answer) to enhance the quality of generated AI.

- **3.3 Error Handling**
  - Implement robust client side logic to manage errors:
    - For 400 Bad Requests, display user friendly messages highlighting input errors.
    - Retry or log failures for 500 Internal Server Error responses while monitoring server behavior.

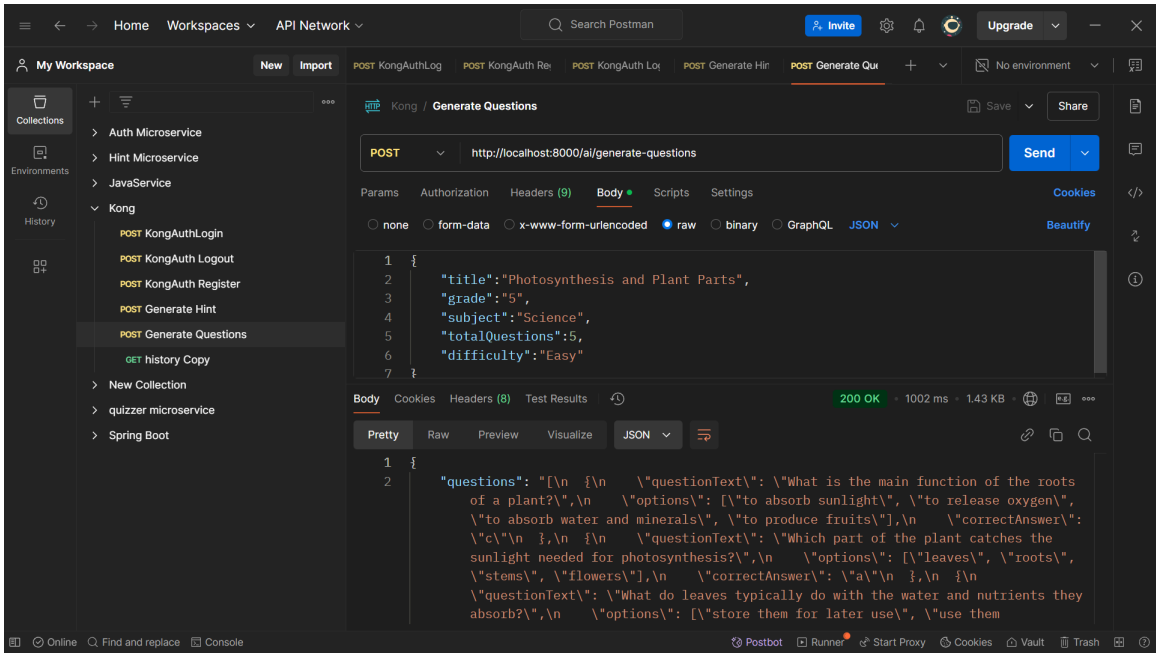
- **3.4 API Scalability**
  - Optimize hint generation by limiting the complexity of questions and ensuring efficient interactions with the Groq API.
  - Consider implementing rate limiting to prevent misuse or overloading of the service.

- **3.5 Testing and Monitoring**
  - Use tools like Postman or automated test suites to rigorously test the endpoint with diverse scenarios (valid, invalid, and edge cases).
  - Monitor API performance and error rates using logging systems to proactively address issues.

4. Screen Shots:



Generate Hint



Generate Question

Docker: A Comprehensive Overview

1. Introduction to Docker:

Docker is an open source platform designed to automate the deployment, scaling, and management of applications within lightweight, portable containers. By encapsulating applications and their dependencies into containers, Docker enables developers to create, test, and deploy applications quickly and consistently across different environments. Docker simplifies the deployment process by ensuring that applications run the same way in development, testing, and production.

2. Core Functions of Docker:

Docker provides several key functions that make it a popular choice for containerization and DevOps practices:

- **2.1. Containerization**
  - At its core, Docker packages applications and their dependencies into containers, ensuring that the application works consistently regardless of where it's deployed. Containers are lightweight, fast, and isolated, which makes them ideal for microservices architectures.

- **2.2. Image Management**
  - Docker uses images to build containers. An image is a lightweight, standalone, executable package that includes everything needed to run a piece of software, including the code, libraries, environment variables, and configuration files.

- **2.3. Container Orchestration**
  - Docker supports container orchestration tools like Docker Swarm and Kubernetes to manage the deployment, scaling, and operation of containerized applications in clusters, enabling automation and resilience.

- **2.4. Version Control and Repository**
  - Docker Hub and other registries (e.g., Docker Registry) allow developers to share and version control container images. This centralized repository enables easy distribution and collaboration on containerized applications.

- **2.5. Networking and Communication**
  - Docker enables containers to communicate with each other and with external systems using virtual networks. Docker's networking features

allow you to create isolated networks for containers and define rules for communication between them.

3. Key Features of Docker

- **3.1. Portability**
  - Docker containers can run on any system that supports Docker, whether it's a developer's laptop, a test server, or a production cloud environment. This portability eliminates the "works on my machine" problem, ensuring consistency across environments.

- **3.2. Isolation and Security**
  - Each Docker container operates in its isolated environment, which prevents conflicts between applications running on the same host system. Docker uses OSlevel virtualization to provide security and resource isolation.

- **3.3. Resource Efficiency**
  - Docker containers are lightweight compared to virtual machines, which allows for higher density on the same hardware. Containers share the host OS kernel but run in their own isolated processes, making them more efficient in terms of CPU and memory usage.

- **3.4. Docker Compose**
  - Docker Compose is a tool for defining and running multi container applications. Using a simple YAML file, developers can specify all the services, networks, and volumes required by their application, and run them with a single command (`**docker compose up**`).

- **3.5. Ecosystem Integration**
  - Docker integrates seamlessly with continuous integration/continuous deployment (CI/CD) pipelines, cloud platforms, and container orchestration systems like Kubernetes, making it an integral part of modern DevOps workflows.

4. Benefits of Using Docker

- **4.1. Simplified Application Deployment**
  - Docker eliminates the complexities of setting up and configuring environments. With Docker, developers can package applications along

with all their dependencies, and deploy them consistently across different systems.

- **4.2. Increased Developer Productivity**
    - Docker enables a faster and more efficient development cycle by allowing developers to focus on code rather than environment configuration. Docker also simplifies collaboration, as developers can share containerized environments with colleagues.

- **4.3. Enhanced Scalability**
    - Docker containers can be easily scaled to meet demand. Containers can be added or removed dynamically, making Docker ideal for microservices architectures and cloud native applications.

- **4.4. Cost Efficiency**
    - By using containers, organizations can maximize resource utilization and reduce infrastructure costs. Containers are lightweight and can run multiple instances on a single host, offering significant cost savings compared to virtual machines.

- **4.5. Consistent Testing and CI/CD**
    - Docker ensures that applications behave the same way during development, testing, and production. This consistency reduces bugs and errors caused by environmental differences and ensures smoother continuous integration and delivery workflows.

5. Deployment and Integration

- **5.1. Deployment Models**
    - Docker can be deployed in several environments, including:

        - **Local Development:** Developers can use Docker on their local machines to build and test containerized applications.

        - **OnPremises:** Docker can be deployed in traditional on premises data centers to support legacy systems and enterprise infrastructure.

        - **Cloud:** Docker integrates well with cloud platforms like AWS, Azure, Google Cloud, and more, enabling organizations to run containers in the cloud for improved scalability and flexibility.

- **5.2. Integration with CI/CD Pipelines**
    - Docker plays a crucial role in modern DevOps workflows by integrating seamlessly with CI/CD tools like Jenkins, GitLab CI, CircleCI, and others. Docker enables automated testing, building, and deployment of containerized applications, ensuring continuous delivery.

6. RealWorld Applications

- **6.1. Microservices Architecture**
    - Docker is a popular choice for deploying microservices. By containerizing each microservice, organizations can easily scale individual components, manage dependencies, and deploy services independently, improving agility and fault isolation.

- **6.2. CloudNative Applications**
    - Docker is widely used in cloud native applications to take full advantage of cloud computing environments. Its portability and lightweight nature make it perfect for elastic scaling and managing cloud resources efficiently.

- **6.3. MultiCloud and Hybrid Environments**
    - Docker's portability and open standards make it an excellent choice for multi cloud strategies, where applications need to be deployed across multiple cloud providers. Docker ensures consistent behavior across different platforms, whether on public or private clouds.

- **6.4. Legacy Application Modernization**
    - For organizations looking to modernize legacy applications, Docker provides a way to containerize older applications, making them easier to deploy and manage in modern cloud environments. This approach can help bridge the gap between legacy systems and modern cloud architectures.

7. Conclusion

Docker has revolutionized application development and deployment by providing a consistent, efficient, and scalable containerization platform. Its ability to simplify application packaging, improve collaboration, and integrate with modern DevOps tools makes it an essential technology for organizations adopting microservices, cloud native architectures, and continuous delivery. With Docker, developers can innovate faster, and organizations can achieve greater operational efficiency and scalability.

Kong API Gateway: A Comprehensive Overview

1. Introduction to Kong API Gateway:

Kong API Gateway is a modern, opensource, and scalable API management solution designed to streamline the interaction between clients and services. Built on top of NGINX, it serves as a lightweight yet powerful platform for managing APIs with high availability and performance. Kong enables developers to secure, observe, and manage APIs effectively while supporting both legacy and modern microservices architectures.

2. Core Functions of Kong:

Kong provides several essential functionalities that make it a popular choice for API management:

- **2.1. API Gateway**
  - At its core, Kong acts as a gateway to manage API requests. It routes client requests to the appropriate backend services, acting as a reverse proxy to abstract and streamline service interactions.

- **2.2. Authentication and Authorization**
  - Kong supports various authentication mechanisms such as API keys, OAuth 2.0, JWT, LDAP, and more. These features ensure secure access to APIs and help organizations enforce access control policies.

- **2.3. Load Balancing**
  - Kong distributes incoming API traffic across multiple backend services, providing effective load balancing to enhance scalability and ensure consistent service availability.

- **2.4. Rate Limiting and Throttling**
  - To prevent abuse and maintain fair usage of APIs, Kong allows rate limiting and throttling. This ensures APIs can handle high traffic without being overwhelmed.

- **2.5. Monitoring and Analytics**
  - Kong offers extensive logging and monitoring capabilities, integrating with tools like Grafana, Prometheus, and Datadog to provide insights into API usage, performance, and error rates.

3. Key Features of Kong

- **3.1. Plugin Architecture**
    - Kong supports a modular plugin system that allows users to extend its functionality. Plugins can be used to implement security, traffic control, transformation, and more. Some plugins come preinstalled, while others can be custom built in Lua or other programming languages.

- **3.2. Scalability and High Availability**
    - Kong is designed for horizontal scalability, allowing it to handle millions of API requests per second. It can be deployed in distributed setups, ensuring high availability across multiple data centers.

- **3.3. Hybrid Deployment**
    - Kong supports hybrid deployment models where the control plane can reside in one environment (e.g., cloud) while the data plane operates in another (e.g., on premises), providing flexibility for enterprise needs.

- **3.4. Service Mesh Integration**
    - Kong can operate as a service mesh for managing microservices communication. It provides features like service discovery, mutual TLS, and traffic routing within a microservices environment.

4. Benefits of Using Kong

- **4.1. Enhanced Security**
    - Kong enables robust API security through authentication, rate limiting, and encryption, protecting APIs from unauthorized access and abuse.

- **4.2. Simplified API Management**
    - Its intuitive interface and plugin system simplify managing complex API infrastructures. Organizations can enforce policies, monitor traffic, and configure services with ease.

- **4.3. CostEffective and Open Source**
    - Being open source, Kong eliminates the need for expensive proprietary solutions while offering an enterprise grade feature set. Paid versions like Kong Enterprise add advanced features for businesses.

- **4.4. DeveloperFriendly**
    - Kong offers an easy to use admin API and a CLI tool, allowing developers to configure and deploy services programmatically. This reduces the overhead of manual setup and maintenance.

5. Deployment and Integration

- **5.1. Deployment Models**
    - Kong can be deployed in various environments, including:
    - Docker and Kubernetes for containerized environments.
    - On Premises setups for enterprise use.
    - Cloud environments like AWS, Azure, and Google Cloud.

- **5.2. Integration with Existing Tools**
    - Kong integrates seamlessly with CI/CD pipelines, monitoring tools, and logging systems, ensuring compatibility with existing DevOps workflows.

6. RealWorld Applications

- **6.1. Microservices Architecture**
    - In microservices, Kong serves as a central gateway to manage API requests, facilitating service discovery, routing, and scaling.

- **6.2. MultiCloud Strategies**
    - Kong's hybrid deployment model supports organizations running services across multiple cloud providers, ensuring consistent API management.

- **6.3. IoT and Mobile Applications**
    - For IoT devices and mobile apps, Kong ensures efficient API communication, handling high volumes of traffic securely and reliably.

7. Conclusion

Kong API Gateway is a versatile tool that simplifies API management for organizations of all sizes. Its robust features, scalability, and extensibility make it a top choice for modern API ecosystems. By centralizing API control, Kong enables businesses to secure, monitor, and optimize their API interactions while empowering developers to innovate faster.