# **Wikipedia Articles Summarization and Conversion to Audio**

# **Project Participants:**

* **Jagadeesh Nadimpalli** - Project Lead, Responsible for system architecture, backend development, and integration of services.
* **Deepak Kandikattu** - Co-developer, assisted with frontend development and testing of the system.

# **Project Goals:**

The primary objective of the Wikipedia Articles Summarization and Conversion to Audio project was to create a service that improves the accessibility and convenience of information consumption from Wikipedia articles. This was accomplished by establishing a system that concisely condenses extensive Wikipedia content and subsequently transforms these summaries into audio format. Below are the key accomplishments of this project:

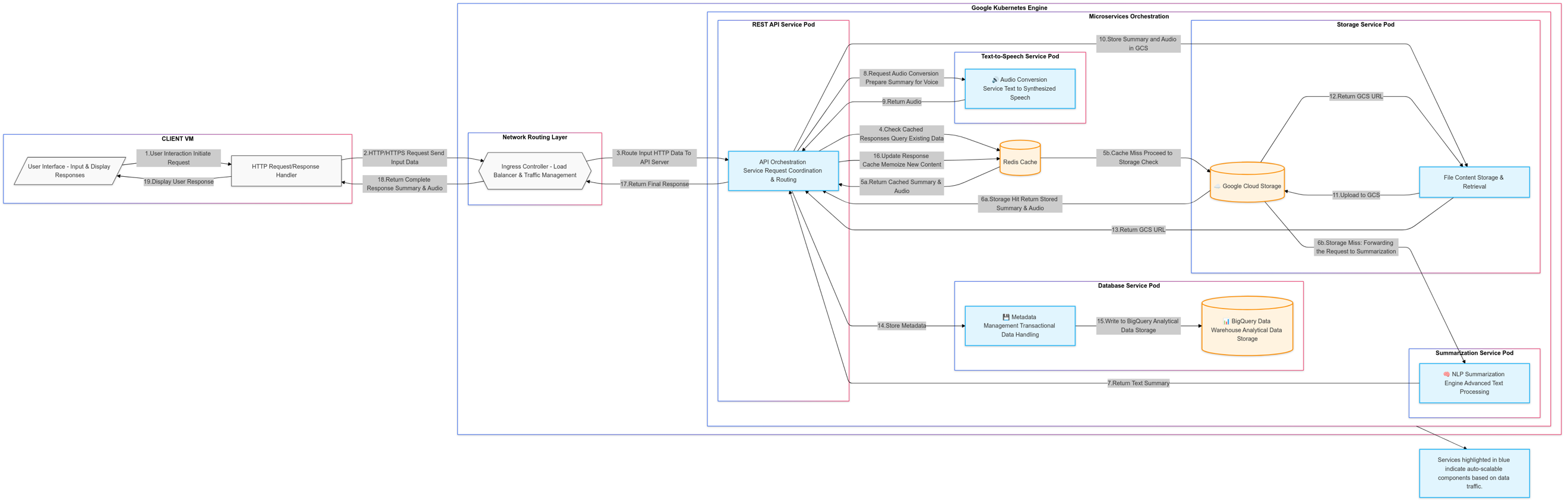
1. **Time-Efficient Content Consumption**:
   * Created a service that automatically extracts and condenses Wikipedia articles, enabling users to comprehend critical information without engaging with the complete text. This is especially beneficial for individuals pursuing quick insights or studying various subjects.
2. **Enhanced Accessibility**:
   * Implemented a text-to-speech system that transforms summaries into audio format, facilitating access to information for both auditory learners and individuals with visual disabilities. This feature helps make things more inclusive by meeting the needs and goals of all users.
3. **Multi-Functional System**:
   * Implemented a comprehensive system that allows for a variety of user interactions, such as the input of articles, summary personalization according to word count, and audio playback management. This functionality provides a user-friendly experience that can be adapted to a wide variety of learning environments and circumstances.
4. **Scalable and Reliable Architecture**:
   * A robust Kubernetes architecture was used to build the project, which ensured that the system is scalable and capable of effectively managing increasing loads. This configuration offers a reliable platform that is capable of supporting real-time data processing and scalability of seamless services.
5. **Real-World Application and Learning**:
   * The project utilizes advanced technologies in NLP and cloud services while providing practical applications in educational and professional environments. It acts as an invaluable resource for students, researchers, and casual learners, enhancing their educational experiences and productivity.

These objectives were accomplished through careful planning, development, and iterative testing, which resulted in the creation of a system that not only satisfies the requirements of a wide range of users but also incorporates cutting-edge technology in order to deliver a service that is seamless.

# **Software and Hardware Components:**

1. **Flask(Rest API)**:
   * **Description**: Flask is a lightweight WSGI web application framework implemented in Python, utilized for the development of web applications. This framework offers a range of tools, libraries, and technologies that facilitate the development of a robust web service.
   * **Usage**: The Flask framework is employed in various backend services, encompassing the REST API, Summarization, Text-to-Speech (TTS), and Database Services. Its primary functions include managing HTTP requests, directing incoming traffic, and provide responses.
2. **Transformers and Beautiful Soup**:
   * **Description**: The Transformers library by Hugging Face facilitates the utilization of pre-trained models for natural language processing (NLP). Beautiful Soup is a library that facilitates web scraping and the parsing of HTML and XML documents.
   * **Usage**: The Summarization Service uses Transformers to process and summaries textual content, while Beautiful Soup is used to scrape Wikipedia articles for the extraction of raw text.
3. **gTTS (Google Text-to-Speech)**:
   * **Description**: gtts is a Python library and CLI tool to interface with Google Translates text-to-speech API.
   * **Usage**: It converts the summarized text into spoken audio, providing an auditory representation of the content for the TTS Service.
4. **Redis**:
   * **Description**: Redis is an open-source, in-memory data structure store, used as a database, cache, and message broker.
   * **Usage**: Redis is utilized to cache the results of operations in order to reduce the amount of time it takes to respond to repeated requests, thereby improving the overall performance of the system.
5. **Google Cloud Storage (GCS)**:
   * **Description**: A cloud storage service that offers object storage for archived or live data within Google Cloud.
   * **Usage**: Utilized for the purpose of persistently storing and retrieving large amounts of data, such as audio files and summaries, which is essential for the Storage Service.
6. **Google Cloud BigQuery**:
   * **Description**: A fully-managed enterprise data warehouse on Google Cloud Platform with SQL-like capabilities.
   * **Usage**: Utilized by the Database Service to store, query, and manage the metadata produced by the application.
7. **Google Cloud Ingress**:
   * **Description**: A managed service within Google Cloud that provides load balancing and traffic management for HTTP(S) applications.
   * **Usage**: Configured to regulate access to microservices within the Kubernetes cluster, effectively directing traffic from the external frontend to the corresponding backend services.
8. **Google Kubernetes Engine (GKE)**:
   * **Description**: A managed environment in Google Cloud Platform for the deployment, management, and scaling of containerized applications utilizing Google infrastructure.
   * **Usage**: Manages the Kubernetes clusters for deploying our microservices, facilitating automated management, scaling, and recovery of our containerized applications.
9. **Google Compute Engine/Virtual Machine (VM)**:
   * **Description**: Google compute engine Virtual machines are simulations of physical computers that provide the functionality of a physical computer. They run applications and operating systems in isolated environments.
   * **Usage**: The frontend service is hosted on a separate virtual machine instance(VM) that is located outside of the Kubernetes cluster. It communicates with the backend services by means of the Ingress controller that is configured in GKE.
10. **Docker**:
    * **Description**: Docker is a set of platform-as-a-service products that use OS-level virtualization to deliver software in packages called containers.
    * **Usage**: The tool is employed to containerize and encapsulate the environment and dependencies associated with each service, thereby ensuring consistency across development, testing, and production environments.

# **Architecture Diagram:**



# **Description of Software and Hardware Component Interactions:**

1. **User Interface to REST API Service:**

* The User Interface (UI) on a Virtual Machine initiates the process upon the submission of a Wikipedia article URL or search term by the user. This data is transmitted through HTTP requests handled by the HTTP Request/Response Handler, which directs these requests to the Google Cloud Ingress Controller. In the following step, the Ingress Controller will route these requests to the appropriate service within the Kubernetes cluster, beginning with the REST API Service. This service functions as the primary coordinator, analyzing incoming data and determining the next steps within the data processing pipeline.

1. **REST API Service to Redis Cache:**

* Upon receiving a request, the REST API Service initially queries the Redis Cache to check whether a response for the identical request has been previously cached. If a cached response is available, it is immediately returned to the user, significantly reducing processing time and load on other services.

1. **REST API Service to Storage and Summarization Services:**

* If the response is not found in the cache, the REST API Service queries Google Cloud Storage (GCS) to check whether the summarized data and audio files from prior operations are available. If located, it retrieves these files and transmits them back to the client. If not, the service directs the request to the Summarization Service for the processing of the Wikipedia article's text.
* The Summarization Service uses the Transformers library for natural language processing tasks and Beautiful Soup for content scraping when necessary. Upon summarization of the article, the summary is transmitted back to the REST API Service.

1. **REST API Service to Text-to-Speech (TTS) Service:**

* Upon obtaining the text summary from the Summarization Service, the REST API Service transmits this text to the Text-to-Speech (TTS) Service. The gTTS library is used by the TTS Service to convert the summary text into audio format, which is subsequently returned to the REST API Service.

1. **REST API Service to Storage Service:**

* The REST API receives the summary and audio files from Summarization service and the TTS Service respectively, which then it coordinates with the Storage Service to store these files in Google Cloud Storage for future retrievals. This helps in reducing redundant processing by saving results of frequently requested articles.

1. **REST API Service to Database Service:**

* Simultaneously, the REST API Service collects metadata regarding the request, including the article title, summary length, and user interactions, and transmits it to the Database Service. This service employs Google BigQuery for the efficient management of analytical data, facilitating comprehensive data analysis and reporting functionalities.

1. **Updates to Redis Cache:**

* Upon preparation of the response, prior to sending it to the user, the REST API Service updates the Redis Cache with the updated summary and audio file, thereby facilitating faster service for subsequent requests pertaining to the same article without necessitating reprocessing.

1. **Response to User Interface:**

* The final response, including a summary and audio file, is transmitted via Google Cloud Ingress to the User Interface (UI) for user accessibility. Users can review the summarized content and access the audio directly.
* The user interface includes a specific rating field for users to assess the quality of summarization and audio conversion. Ratings are determined by three criteria: summarization accuracy, audio quality, and overall experience. Submitted ratings are transmitted as a POST request to the /submit\_rating endpoint of the REST API, which subsequently updates metadata in BigQuery through the /update\_rating endpoint.
* This feedback is essential for evaluating the effectiveness of NLP models and TTS services, enabling ongoing analysis and improvements. User ratings help identify areas for enhancement, such as increasing NLP accuracy and improving audio naturalness, fostering continuous service improvement.

**Load Balancing and Traffic Management:**

Throughout this process, the Google Cloud Ingress is essential in managing traffic distribution among service instances, preventing any single service from becoming overloaded during peak demand periods. This is essential for preserving system responsiveness and availability.

**Auto-scaling Capabilities:**

The Kubernetes cluster employs Horizontal Pod Autoscalers (HPA) for key services like the Summarization and Text-to-Speech Services, enabling dynamic scaling based on CPU and memory usage. Autoscaling decisions are triggered by predefined metrics, such as CPU utilization, allowing the system to automatically increase or decrease the number of active pods as demand fluctuates. This capability prevents service degradation during peak loads, ensuring optimal performance and accessibility. Configuration and management of autoscaling rules are handled through Kubernetes YAML files and the API, offering flexibility to adapt scaling policies according to real-world usage and performance analytics.

# **Debugging and Testing Methodologies:**

Throughout the development and deployment phases of the Wikipedia Articles Summarization and Conversion to Audio project, a comprehensive and structured testing and debugging strategy was essential to ensure the system’s functionality and robustness.

1. **Debugging Methodologies:**
   * **Development Environment Setup:**
     + Each service was developed in a local environment utilizing Docker Desktop to replicate the production settings. With this approach, dependencies could be isolated, and consistent behavior could be ensured across a variety of development environments.
   * **Logging:**
     + Systematic logging was established within each service utilizing Python's integrated logging library. The logs contained comprehensive details regarding data flow, error notifications, and system alerts. This was essential for real-time monitoring and retrospective analysis in both development and production environments.
   * **Error Handling:** 
     + Exception handling was meticulously applied throughout all layers of the application. Each service incorporated try-catch blocks to handle exceptions, specifically concerning network requests, data parsing, and external API calls, facilitating the rapid identification and isolation of failures.
2. **Development and Local Testing:**

* **Initial Setup and Local Testing:** The initial phases of development and testing for the project were carried out utilizing Docker Desktop. Initially, each service, including the storage service and the metadata service, was developed and tested in a local environment. For instance, The storage service was tested through the Minio object store to assess its file storage functionalities, whereas the metadata service initially utilized CSV files for the storage of metadata.
* **Debugging Tools:** To track the behavior and output of services during local testing, logging and error-handling mechanisms were extensively utilized. This facilitated the early detection and resolution of issues.

1. **Migration to Google Kubernetes Engine (GKE):**
   * **Service-by-Service Migration:** Services were migrated one at a time from Docker Desktop to GKE. This step-by-step approach allowed for careful management of dependencies and configurations specific to the cloud environment.
   * **Adaptation of Service Logic:** For example, the connection of the storage service was shifted from Minio to Google Cloud Storage, requiring adjustments in the service's logic to handle different storage mechanisms.
   * **Logging and Error Handling**: Comprehensive logging was implemented to facilitate real-time monitoring and debugging. Error handling mechanisms were added to each service to manage and log exceptions effectively.
   * **Testing with kubectl and curl:** Each service, once deployed on GKE, was tested individually using kubectl port-forward and curl commands to simulate client requests and confirm the functionality of the service within the cloud environment.
2. **Integration Testing:** 
   * **Comprehensive Service Integration:** A comprehensive integration test was conducted following the deployment of all services on GKE. The REST API service, which orchestrates interactions among all services, was evaluated for its ability to manage requests accurately, direct them to the appropriate services, and consolidate responses.
   * **Use of Pod Logs:** Kubernetes pod logs were essential for system monitoring during these tests, offering immediate insights into errors and operational metrics.
3. **Ingress Configuration and Testing:**
   * **Implementation of Google Ingress Controller:** The ingress controller was configured to optimize traffic management to the services, eliminating the necessity for port-forwarding.
   * **Ingress Testing:** The functionality of the ingress was assessed by sending requests via its external IP and confirming that they were accurately directed to the REST API service.
4. **Frontend Development and Testing:**
   * **Separate Virtual Machine Setup:** The frontend was developed on a virtual machine external to the Kubernetes cluster. It was developed utilizing HTML, JavaScript, and CSS, and evaluated across various browsers to guarantee uniform rendering and functionality.
   * **End-to-End System Testing:** The entire system, including both frontend and backend, was evaluated to monitor data flow and interactions via the ingress controller and all backend services.
5. **Load Testing and Auto-Scaling Evaluation:**
   * **Simultaneous Requests:** Multiple concurrent requests were transmitted through the frontend to evaluate the system's responsiveness and scalability.
   * **Monitoring and Scaling Behavior:** Observations were conducted regarding the efficacy of ingress in managing load distribution, the replication of pods under load, and the performance of each service under stress conditions. The auto-scaling functionality of the summarization service was meticulously examined to confirm it achieved anticipated performance standards.
6. **Performance Monitoring:**
   * **Google Cloud Monitoring:** This tool was utilized to monitor the CPU and memory utilization of each service during testing, yielding critical insights into the system's performance under diverse conditions.

**Conclusion:**

This methodical and phased approach to testing covered various dimensions of testing including unit-like testing for individual services, integration testing for service interaction, and performance testing under load. The extensive utilization of Kubernetes-native tools for monitoring and debugging enabled a comprehensive understanding of the system's behavior in production-like environments.

**Explain the working system, what kind of workload can it handle, where are potential bottlenecks?**

# **Overview of the Working System:**

The Wikipedia Articles Summarization and Audio Conversion service functions via a resilient microservices architecture implemented on Google Kubernetes Engine (GKE). This configuration enables each system component—spanning content retrieval, summarization, audio conversion, and storage—to operate independently while maintaining cohesion. The system is engineered to process requests for summarizing Wikipedia articles and provide those summaries in both textual and audio formats. Users interact with the system via a frontend hosted on a separate virtual machine, which communicates with the backend services through a Google Cloud Ingress controller.

# **System Capabilities and Limitations:**

**Operational Capacity:**

The "Wikipedia Articles Summarization and Conversion to Audio" system aims to facilitate the accessibility and consumption of Wikipedia content by summarizing articles and transforming them into audio format. The architecture constructed on Google Kubernetes Engine (GKE) facilitates a scalable and robust infrastructure that handles varying demand levels.

* **Workload Handling:**
  + **Scalability**: Kubernetes enables each service component to scale independently according to the workload. For instance, during high-demand periods, the Summarization Service and TTS Service can scale out to additional pods to handle increased traffic.
  + **Load Balancing**: The Ingress controller distributes incoming requests uniformly among the available service instances, preventing any single pod from becoming a bottleneck.
  + **Resource Allocation**: Resources such as CPU and memory are allocated according to the demand shown by each service, with constraints established to prevent any service from monopolizing system resources.

## **Potential Bottlenecks and Mitigation Strategies:**

1. **Summarization Service:** The summarization process involves complex NLP tasks, which may lead to this service becoming a bottleneck during periods of high demand, particularly when extensive text processing and AI inference are necessary. Auto-scaling eliminates this issue; however, there is an intrinsic delay in provisioning new instances that may impact responsiveness during sudden demand spikes.
2. **Text-to-Speech (TTS) Service**:
   * **Bottleneck**: Like the Summarization Service, the TTS service may encounter bottlenecks when converting large amounts of text to speech concurrently, primarily due to constraints in processing power.
   * **Mitigation**: Autoscaling is enabled according to CPU and memory metrics to facilitate the management of large workloads. Moreover, sophisticated caching techniques may be utilized to reutilize previously generated audio files when possible.
3. **Database Service**:
   * The write operations to Big Query may become a bottleneck due to exceedingly high concurrent data logging requests.
   * **Mitigation**: Employ batch processing for write operations and ensure the database schema is optimized for rapid writes and queries.
4. **Network Latency Between Components:** Due to the distributed architecture of services, particularly with the frontend hosted independently from the Kubernetes cluster, network latency can affect the response time experienced by users. Enhancing network configurations and potentially utilizing a content delivery network (CDN) could solve this problem.

# **System Scope:**

* **Current Scope:**

The system is engineered to effectively process English Wikipedia articles, providing summarization and audio conversion functionalities via a user-friendly interface. The system utilizes a modular and scalable backend architecture, leveraging microservices on Google Kubernetes Engine (GKE) to facilitate the management of diverse functions such as summarization, audio conversion, and data storage. This framework improves maintenance capabilities and facilitates the scaling of components in response to user demand. The frontend is deployed on a virtual machine, offering a user-friendly interface that enables users to conveniently submit article URLs or search terms, as well as access summarization results along with playable audio files. This design caters to a diverse audience, including students, researchers, and casual users seeking quick information, ensuring accessibility and ease of use for all.

# **Future Scope:**

1. **Multilingual Support:**
   * **Language Expansion:** Future enhancements may incorporate support for multiple languages to ensure universal accessibility of the system. This could involve the incorporation of multilingual NLP models capable of processing diverse languages and dialects, thereby significantly broadening the user base.
2. **Content Adaptability:**
   * **Beyond Wikipedia:** Enhancing the system's functionalities to incorporate diverse document types and sources, including news articles, scholarly papers, and books, could significantly augment its utility. Modifying the summarization algorithms to handle various formats and writing styles is essential.
3. **AI and Model Enhancements:**
   * **Continuous Learning:** Implementing a continuous learning system for the AI models to adapt and improve based on user feedback and new data could enhance accuracy and relevancy of the summarizations over time.
   * **Personalization Features:** Implementing features that enable users to customize the depth of summarization and the voice and speaking style of the audio output could improve personalization and user satisfaction.
4. **Advanced User Interaction:**
   * **Interactive Summaries:** Integrating interactive components into summaries, such as hyperlinks to comprehensive sections or pertinent articles, may enhance user engagement and information delivery.
   * **Voice Interaction:** Implementing voice command functionality could facilitate hands-free operation, enhancing the system's accessibility and convenience, particularly for users with disabilities or those on the go.
5. **Global Deployment:** Extending deployment across various geographical regions within Google Cloud can diminish latency for globally dispersed users and offer redundancy in the event of regional cloud service failures.
6. **Advanced NLP Features:** The current summarization lacks the capability to tailor summaries according to user preferences or varying contexts. Future improvements may encompass machine learning models that adjust according to user feedback or context, delivering more tailored summaries.

# RESULTS:

**Accessing with VM Instance External Ip address:**

A screenshot of a computer

Description automatically generated

**Request Processing at backend**

A screenshot of a computer

Description automatically generated

**Response message displayed on UI:A screenshot of a computer

Description automatically generated**

**Feedback message when submit rating:**

A screenshot of a computer

Description automatically generated

**Reset button, when user wants to retry:A screenshot of a computer

Description automatically generated**