Real-Time Chatbot: Microservices Implementation in Distributed System Architecture

Wisnu Uriawan * , Reza Fahlevi Herdiyanto , Rd Imam Saepul Millah , Sami Irhamnillah , Siti Nurhayati Gunawan

Posted Date: 1 July 2024

doi: 10.20944/preprints202407.0028.v1

Keywords: chatbot, microservices.

Article

Real-Time Chatbot: Microservices Implementation in Distributed System Architecture

Wisnu Uriawan *, Reza Fahlevi Herdiyanto, Rd Imam Saepul Millah, Sami Irhamnillah and Siti Nurhayati Gunawan

Informatics Department, UIN Sunan Gunung Djati Bandung, Jawa Barat, Indonesia; wisnu.uriawan@uinsgd.ac.id; rfahlevih@gmail.com; rdsaepulmillah@gmail.com; samiirhamnillah03@gmail.com; gsitinurhayati017@gmail.com

* Correspondence: wisnu.uriawan@uinsgd.ac.id

Abstract: The development of a real-time chatbot using microservices architecture has been proposed as a solution to overcome the main challenges faced by single-service chatbots, namely limited scalability and performance. In this research, Agile Methodology is used as a development approach that allows the team to flexibly build and update each microservices service iteratively, according to the changing needs that occur. The results show that the implementation of microservices on a chatbot can result in faster responses to user requests, significantly improving system efficiency. The microservices architecture provides advantages in terms of modularity, where system components such as session management, business logic, and user interface can be organized and developed separately. This makes it possible to upgrade and improve certain parts without disrupting the entire system, as well as providing flexibility in customizing features and scaling services as needs evolve. Future research is expected to further explore the broader integration potential of this microservices system in chatbot applications, as well as evaluate the security and reliability aspects in a broader context of use.

Keywords: chatbot; microservices

1. Introduction

Chatbots have evolved into an essential component of digital interaction as technology advances. It has the ability to interact with its users quickly and efficiently which makes it one of the most useful tools to help its users. In an era where user engagement is very important, the existence of chatbots not only fulfils the need for a quick response, but can create an interactive experience for its users. Chatbot usage has reached 1.7 billion users by 2023 in Chat-GPT [1].

Although today's chatbots have become an important part of the human experience, the problems faced by previous chatbots are still common. Problems often faced by previous chatbots are about satisfactory user experience such as the increasing complexity of user requests and the need for integration with various systems. Monolith systems used in conventional methods often cannot achieve the required scale, which can cause problems in scalability and performance. Users may experience long response times and system failures that can create a poor user experience.

An innovative solution to overcome the limitations of traditional chatbots is a distributed system that uses a form of microservices. Because by dividing the chatbot system into small, independent components, the chatbot can become more responsive and scalable. Each microservices, can be managed separately which allows for more flexible and efficient development, testing and maintenance.

The opportunity to enhance a chatbot's ability to integrate data from multiple sources arises with the use of microservices architecture. Chatbots can easily connect with internal systems, databases, and third-party services with small components that can work independently. This allows the chatbot to provide more detailed and relevant answers to its users which can improve the interactive experience for its users. [2]

In addition, implementing a chatbot with microservices enables the development of new features more quickly and effectively. There is no need to pay attention to the impact on the entire system if you want to develop and improve the enhancement of certain functions within each microservices. In this way, it enables faster response to user needs, which results in faster and more relevant innovations.

 doi:10.20944/preprints202407.0028.v1

Microservices architecture can make a chatbot more flexible and adaptable to changes in technology and the needs of its users. This allows the chatbot to adjust its compatibility with its environment quickly and dynamically. Thus, the chatbot can continue to function well to meet the needs and expectations of users. This ensures a smooth interaction between chatbots and users and allows them to stay abreast of the latest trends and developments in the industry, so that they can continue to provide high-quality services. [3]

although traditional Chatbots often fail to provide a satisfactory user experience, despite being widely used. These challenges include handling increasingly complex user requests and ensuring that various systems work well together. Conventional monolithic systems often face scalability and performance issues, leading to system failures and slow response times, which can reduce user experience.

To overcome these limitations, a creative solution using microservices architecture has been proposed. The chatbot system can be divided into small independent parts by microservices, which makes it more scalable and responsive. This method allows each microservice to be managed separately, which enables more flexible and efficient development, testing, and maintenance. Chatbots can help integrate data from multiple sources, provide more detailed and relevant responses, and enrich the user experience.

2. Related Work

In their research, Setyadhi Putra Deriyanto and Heru Agus Santoso explore how to use the Life Cycle Approach to Network Design Method to create a Bot for Microservices Server Monitoring. With an emphasis on the utilization of microservices architecture and cutting-edge technologies like Docker, Prometheus, Nodeexporter, Alertmanager, and Telegram Messenger, the research focuses on building a monitoring system for the Tanggapin emergency response system in Central Java. The writers want to improve network service performance, scalability, and reliability by integrating these technologies, especially for real-time monitoring and notification systems.[4]

The development of chatbot technology has reached an advanced stage, with various methods and architectures designed to meet the specific needs of users in diverse scenarios. One example is the microservices-based chatbot architecture designed to support chronic patients with a focus on the stability of standard data models and conversational modeling. This architecture allows for the addition of new services and functionalities to meet the evolving needs of users in healthcare contexts.[5]

Another method employed is PPDIOO (Prepare Plan Design Implement Operate Optimize), which is applied in a structured and sequential manner from the preparation and planning stages to the operation and optimization of the system as a whole. This method is used in the development of a microservice monitoring system with a Telegram bot to provide real-time notifications related to server downtime or overload. The result is an effective system for providing real-time notifications regarding server issues.[6]

The healthcare industry has also seen recent advancements in chatbot technology. For instance, the significance of stability in conventional data models and conversational modeling has been brought to light by the creation of a microservices-based chatbot architecture intended to support chronic patients. The chatbot's architecture facilitates the smooth integration of additional features and services, guaranteeing its ability to adjust to the changing requirements of users across diverse healthcare environments. The flexibility and scalability of microservices architecture are highlighted by this method, which can be advantageous when combined with real-time monitoring systems such as the one suggested for Central Java's Tanggapin emergency response system.[6]

In addition, current system architectures are using containerization technologies like Docker more and more frequently. The performance and dependability of microservices depend on the continuous deployment of applications across many environments, which Docker makes possible. When coupled with other monitoring tools such as Prometheus and Nodeexporter, Docker offers a stable platform for

 doi:10.20944/preprints202407.0028.v1

tracking performance in real time and identifying problems. In emergency response systems, when prompt messages and system stability are critical, this connection is very helpful.[6]

Moreover, combining messaging apps like Telegram Messenger with alert management systems like Alertmanager provides an efficient channel of communication for instant alerts and notifications. This integration makes it easier to respond quickly to important events, which raises the monitoring system's overall effectiveness. The research seeks to develop a comprehensive monitoring solution that guarantees high dependability and responsiveness in emergency scenarios, while also improving network service performance and scalability by utilizing these state-of-the-art technologies.[6]

Microservices design has shown to be very successful in enhancing system scalability and maintainability in the field of distributed systems and cloud computing. Microservices can be utilized to create scalable chatbot systems for healthcare applications that need to be resilient and highly available, as demonstrated by a study by Roca et al. Because microservices are modular, various components may be developed and deployed independently, lowering the possibility of system-wide failures and making it easier to update or add new functions.[5]

The assistance that microservices architecture offers for continuous integration and continuous deployment (CI/CD) techniques is yet another important benefit. Because it enables automated testing and update distribution, this functionality is essential for preserving the high availability and reliability of services. This approach has proven very helpful in settings that need for quick iteration and deployment, such the creation of healthcare chatbots that offer chronic patients the most recent information and support.[5]

In numerous studies, the usage of Prometheus for microservices monitoring has been thoroughly documented. Prometheus provides thorough insights into system performance parameters through its robust data collection and querying capabilities, allowing for proactive control of system health. It may gather and display data from various system components when combined with Nodeexporter, giving an all-encompassing picture of the system's operational state. This configuration works especially well in complicated microservices systems, because maintaining overall system stability requires an awareness of the interdependencies between services.[5]

A strong monitoring system also requires the integration of Alertmanager with Prometheus. Alertmanager manages alert routing and notifications, including configurable alerting channels and rules. This adaptability minimizes downtime and boosts system resilience by guaranteeing that important concerns are immediately handled by the right staff. For high-stakes settings like emergency response systems or healthcare, these real-time alerting systems are essential to sustaining dependability and ongoing service availability.[6]

Last but not least, using Telegram Messenger as an alert system notification channel has a lot to offer in terms of responsiveness and accessibility. Telegram is a great medium to integrate with monitoring systems because of its extensive use and real-time notification capabilities. This integration guarantees the timely delivery of notifications to relevant parties, hence promoting rapid action and augmenting the system's overall responsiveness. The goal of the suggested monitoring solution is to achieve a high degree of efficiency and dependability by combining various technologies, which is necessary for crucial applications like the Tanggapin emergency response system.[6]

Additionally, the waterfall model with blackbox testing is applied to a system that implements CRUD (Create Read Update Delete) processes for a chat application. This application is integrated with Firebase data and allows users to share stories anywhere and anytime.[7]

Natural Language Processing (NLP) is also a key component in chatbot development, enabling direct chat through the use of word stemming, regular expression matching, and an NLP approach for Q and A sessions. A chatbot application developed with this method meets expectations by providing outputs and validation that align with 100 persen of the test scenarios.[8]

Finally, the waterfall method is used in text processing with TF-IDF, VSM, and cosine similarity techniques to develop a chatbot application that trains itself with information regarding academics. For instance, this chatbot can provide information about university addresses, registration room locations,

 doi:10.20944/preprints202407.0028.v1

the names of deans, vice deans, and program heads at Panca Marga University Probolinggo, thus functioning as a highly useful virtual assistant in an academic context.[9]

This paper proposes a credit scoring method based on FICO oscore and FIVE Cs, we analyze the research conduct of credit scoring model that have been implemented [10].

3. Methodology

There are two approaches used in this research, namely agile for system development and microservices for software architecture.

3.1. Agile Methods

Agile is a software development methodology that emphasizes flexibility and rapid responsiveness to change. It allows teams to continuously develop, test, and modify products through short iterations. Each iteration produces a product component that can be evaluated, allowing for adjustments based on feedback provided throughout the process.

This method emphasizes the importance of individual interaction and processes and tools, as well as software that has complete documentation. To ensure that the product being developed meets the needs and expectations of customers, it is essential to work closely with them. The ability to adapt to change, even in the late stages of development, is an important aspect of agile.

Agile prioritizes transparency and continuous communication among team members. Planning, designing, developing, testing, implementing, and reviewing are the steps in this study that use the agile approach, as shown in figure 1. By considering the steps mentioned, developers can effectively change project strategies and tactics according to the changes that occur. This method encourages innovation and more efficient development during the software development cycle and increases responsiveness to change.

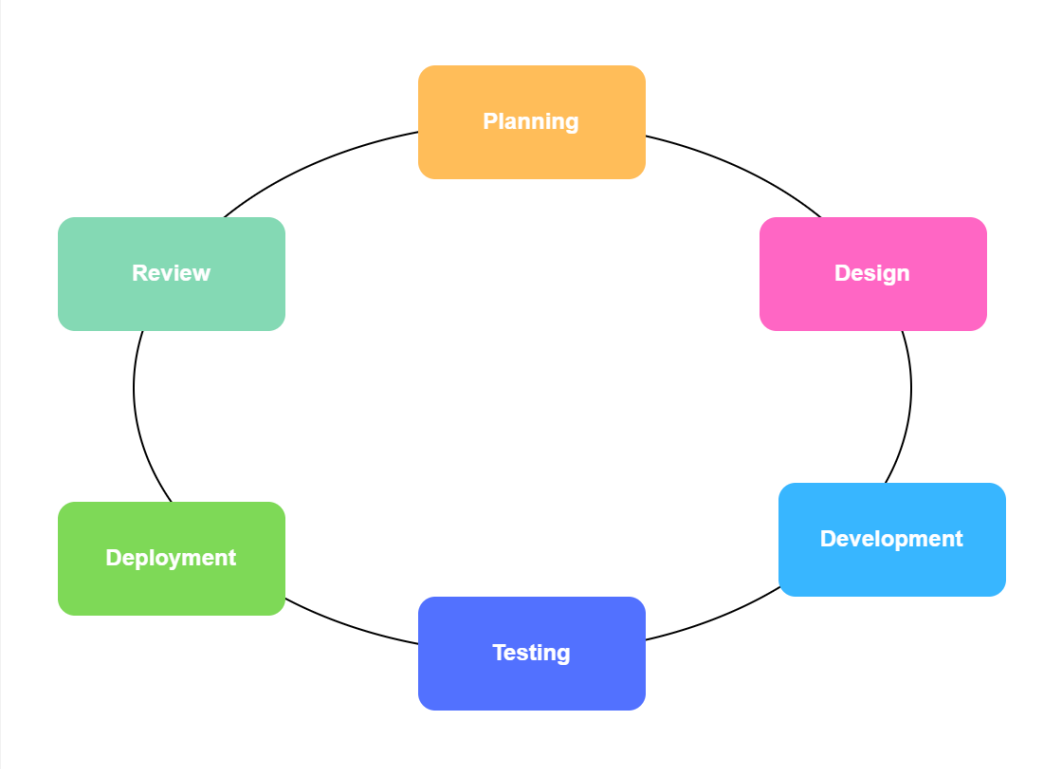

Figure 1. Agile Sofware Development Life Cycle [11]

 doi:10.20944/preprints202407.0028.v1

3.2. Microservices Methods

Microservices is a software architecture method where the application is divided into a collection of small, independent services. Each of these services is responsible for a specific function and can be developed, managed, and operated independently. These services communicate with each other through HTTP/REST protocols or queues. Compared to the traditional monolith architecture that combines all system functions into one large unit, this method allows for more flexible development and scaling.

Scalability and resilience are the main advantages of microservices. The development team can choose a suitable technology for each service without worrying about compatibility with the rest of the system because each microservice can operate independently. Moreover, it allows certain services to be scaled independently as needed, without the need to upgrade the entire application. Moreover, errors or disruptions that occur in one service will not have a direct impact on other services, thus increasing the overall durability of the system. Since services can be updated or replaced without changing the overall system, this method allows for faster and iterative development.

Based on the above two approaches, we can describe this research path as follows:

1. Planning

   This planning step includes identifying and analyzing the needs that must be met during the iteration. Planning is done to ensure what the real-time chatbot should do to achieve the goals and priorities in the next sprint. Such as determining the main features to be developed in iterations.
2. Design

   After the planning is complete, the solution to be developed is designed according to the planned needs. Design is done iteratively, with a focus on flexibility and the ability to adjust to changes that may occur during iterations. Includes the design of microservices architecture that supports the operation of real-time chatbots in distributed systems.
3. Development

   In this stage, coding and implementing the designed features or services is done in sprints. Conformance and quality are ensured through constant feedback throughout the development process which is done incrementally and iteratively. Each microservice is developed on its own, but remains connected to other services.
4. Testing

   Each iteration involves thorough testing to ensure that the developed components are working properly. These tests are conducted regularly and include performance and functionality testing to ensure that all features run according to the specified specifications.
5. Deployment

   After testing, the features or services that have been developed will be deployed to the production environment. This deployment will be done in stages for each iteration. This ensures that changes can be integrated quickly and efficiently. Microservices are deployed to the production or staging environment separately and then integrated.
6. Review

   This stage includes evaluation of the results of each iteration. Meetings are held to review the performance and results achieved, identify improvements and lessons learned that can be applied to the next iteration, and improve the overall process and product quality. For the next iteration, adjustments and improvements are made based on user feedback and testing.

 doi:10.20944/preprints202407.0028.v1

4. Result and Discussion

4.1. Result

From the beginning to the end of this research, Agile methodology was used to develop a real-time chatbot with distributed system architecture. The research process is as follows :

1. Planning

   To build and deploy a chatbot well, planning is necessary. This planning includes identifying objectives, defining a clear functional scope, scheduling development iterations using Agile methodologies, and planning for supporting infrastructure, such as using microservices in a distributed system architecture. To ensure that the resulting solution not only meets expectations but also meets the desired quality standards, planning also involves analyzing needs and risks to identify issues that may arise during the implementation process. Below is a needs analysis that has been compiled based on the results of the previous analysis :

   (a) Functional Requirements

   The first functional requirement is real-time communication. To fulfil this need, we chose the HTTP protocol. This protocol can keep the chatbot continuously connected with the user, which allows instant message exchange without the need to load a page or wait for a response. A quick response will improve the user's interaction experience and help them feel well served.

   We use a secure authentication mechanism for user authentication. One of our options is JSON Web Token (JWT) to create secure authentication which includes the use of authentication tokens to verify users. By using JWT, we ensure that the authentication process is fast and secure, allowing secure access to the system. This is an important step in keeping our users' data and information safe.

   In addition, integration with Groq's Programming Application Interface (API) allows this system to function better and can connect it to various services and resources. An interactive and responsive experience is provided to users by relying on fast responses like a regular chatbot.

   Data management is very important in the development of this system. By combining MongoDB for unstructured data and SQL for structured data, this system can efficiently store and manage various types of data. SQL ensures data integrity and optimal system performance, while MongoDB allows flexibility in handling unstructured data.

   (b) Non Function Requirements

   The first non-functional requirement is performance. The system should have low latency to enable fast response to users. This involves efficient use of resources and process optimisation to minimise the system's response time to user requests. In addition, there needs to be an optimisation mechanism to ensure that its operation remains efficient even during high workloads.

   In addition, scalability is important for the system to be scalable so that it can handle an increasing number of users and requests without compromising performance. This involves the use of technologies that support distribution, as well as careful capacity planning to ensure the infrastructure can handle the anticipated growth without the need for replacement of major components.

2. Design

   In system design, there is a system architecture and Unified Modelling Language (UML) created based on the previous requirements analyst. It starts by describing the overall structure of the system, the main components involved, and how they relate to each other. To give a better idea of how the system works, a Unified Modelling Language (UML) diagram will be presented.

 doi:10.20944/preprints202407.0028.v1

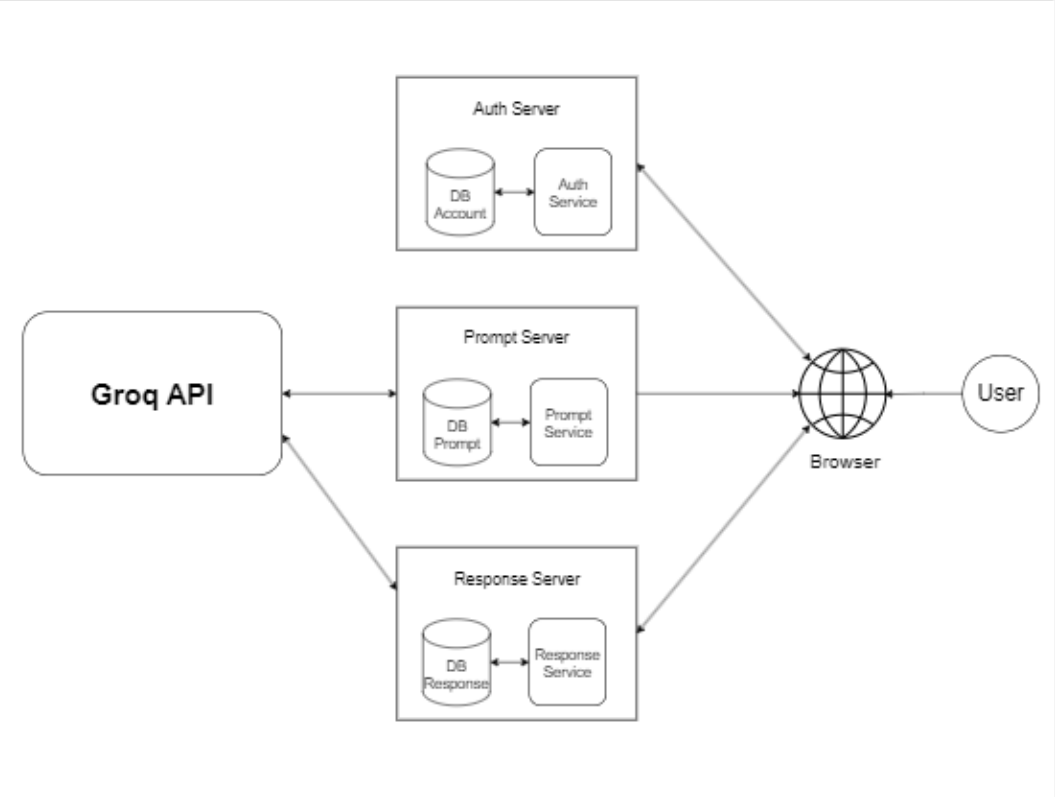

Figure 2. Architecture System Microservices

The microservices-based chatbot architecture diagram in Figure 2 shows several main components, such as the authentication server, prompt server, response server, Groq API, and users who can access through a browser. To provide the chatbot service, this diagram shows how each component interacts with each other.

Auth server has a role in user authentication. Auth server consists of two components, namely DB Account and Auth Service. DB Account is responsible for storing user account information, while Auth service is responsible for handling the authentication process. When a user wants to use a service, the authentication server is responsible for ensuring that the user's credentials are valid before granting permission to use other services. This ensures that only authorised users can use the system.

The Prompt Server is responsible for handling and storing questions or prompts asked by users. Prompt Server consists of two main components, namely DB Prompt and Prompt Service. The Prompt DB stores the prompt data and the Prompt Service processes the prompt. When a user sends a prompt through a browser, Prompt Server sends this data to Groq API for further processing. Therefore, the Prompt Server serves as the starting point for handling user requests.

Response Server has the task of storing response data to user prompts. Its main components are DB Response and Response Service. DB Response stores the response data generated, while Response Service processes the response. Upon receiving a prompt that has been processed by Groq API, Groq API will provide a response and will be retrieved by Response Server and stored in DB Response. This will return the appropriate response to the user via the browser, ensuring that each user request is responded to with the right information.

Groq API is the main link between Prompt Server and Response Server. The role of Groq API is crucial to ensure efficient and smooth communication between various server components. It receives prompts from the Prompt Server and processes them immediately, then sends the

 doi:10.20944/preprints202407.0028.v1

response results to the Response Server. With this API, prompt processing and response delivery can be done quickly.

Through the browser, users interact with this system. The browser serves as the user's interface with the system. The user sends authentication to the Auth Server, and once authenticated, the user will send a prompt to the Prompt server. Then the Response Server will send a response back to the browser to be displayed to the user. Therefore, the browser facilitates the entire interaction process between the system and the user.

This chatbot architecture starts with the authentication process. When a user attempts to access the chatbot, an authentication request is sent to the Auth Server. The Auth Server then verifies the user's credentials by accessing the DB Account to ensure the user's identity. Once verification is complete, the Auth Server returns the authentication status to the user's browser, allowing the user to proceed to the next step.After successful authentication, the user can send a prompt through the browser. This prompt is then forwarded to the Prompt Server. The Prompt Server will store the prompt data in the Prompt DB and process it with the help of Prompt Service. This process ensures that every user prompt is saved and ready for further processing. The Prompt Server will then forward the prompt to the Groq API for further processing. Groq API processes the prompt and generates the data required for the response. This processed data is then sent to the Response Server. The Response Server will store the response in the Response DB, and process it using the Response Service, before sending the final result back to the user's browser.The entire user interaction occurs through the browser. Therefore, the interaction between the user and several server services and this architecture is connected by the browser. This microservices-based architecture allows the chatbot to operate efficiently as each component of the system can work independently yet remain integrated.

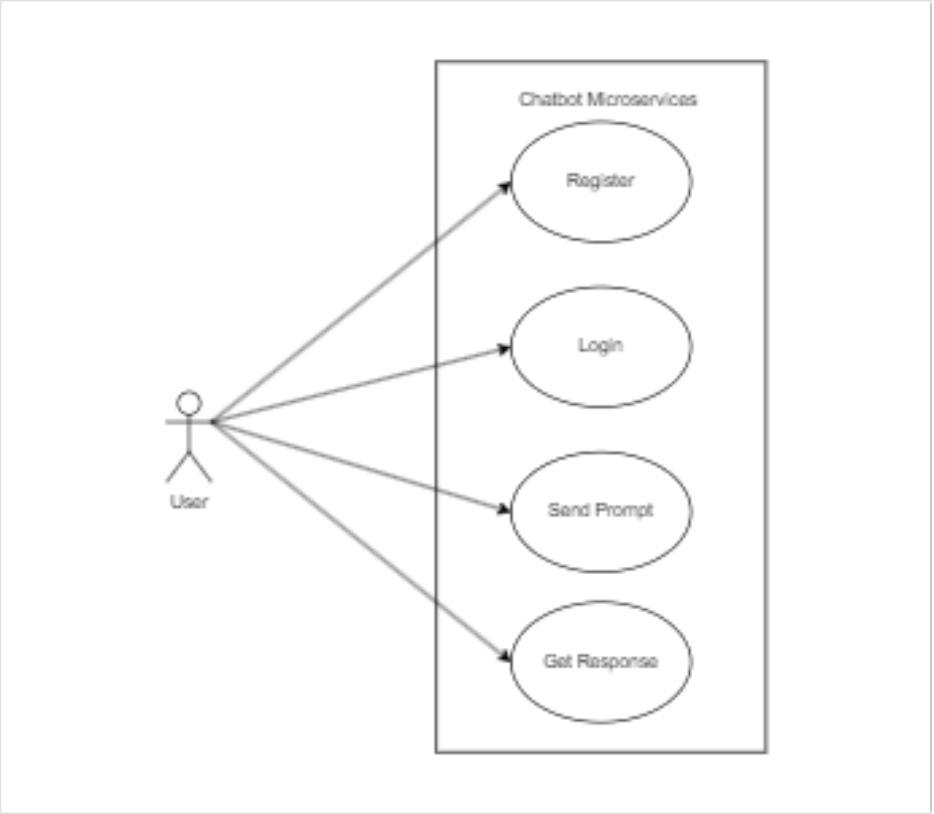

Figure 3. Use Case Diagram

In the use case diagram figure 3, there are four main functions that can be accessed by users, namely register, login, send prompt, and get response. To show the interaction that occurs

 doi:10.20944/preprints202407.0028.v1

between the user and the system, each function is depicted with an elliptical circle that is directly connected to the user. In addition, the diagram explains how each function is directly related to user activity.

Any new user who wants to use the microservices chatbot can start with the Register function, where they can register their account by providing their email and password. This registration process ensures that any user who has a valid account can access the microservices chatbot. With that, the system can recognise and authenticate the user's account.

After registration, users need to perform the login process. This login process requires validation of user credentials, such as checking whether the email and password entered by the user are registered or not. In addition, this system uses Jason Web Token (JWT) to verify the token during the login process. The token given after a successful login will be matched with the token in the DB Account, so that only users who have a valid token can access the microservices chatbot.

Users who access the service with a valid account, they can access two main functions on this microservices chatbot, namely send prompt and get response. Through the Send Prompt function, users can send questions to the chatbot. The system will then process this input by sending it to the API service. The chatbot can be used directly by users through the Send Prompt feature, which allows users to provide various questions. This interaction between the user and the system can improve the user experience because the communication between the user and the system occurs in real-time.

The Get Response function is used to access the results after the user input is processed by the system. The chatbot will generate an answer to the user's question according to the question. This Send Prompt function with Get Response works together to make this chatbot able to communicate smoothly with users. Each component in the microservices framework is very important to ensure good interaction, meet user needs, and can provide the desired service. This use case diagram as a whole provides an overview of how the system interacts with the user.

In the development of real-time chatbot with microservices architecture, there is an activity diagram that describes how the process flow of the system will be made. In this activity diagram, there are several activities that show the main steps of the system, as can be seen in the pictures below :

(a) Activity Diagram Register

Figure 4 shows the activity diagram of the user registration process for a chatbot application that uses microservices. There are two main columns, "User" and "Chatbot Microservices". In the "User" column, the action the user takes starts with opening the chatbot application. After that, the user starts the registration process by clicking the registration button. Next, the user fills in the registration form with the required personal information and submits it for processing by the system.

In contrast, the "Chatbot Microservices" column describes how the system responds to each user action. The Microservices Chatbot will display the registration form when the user clicks the registration button. This is to ensure that the user can fill in all the required information in the appropriate format. Once the user fills and submits the registration form, the system processes the inputted information. The process includes data verification to ensure the completeness and accuracy of the data.

Next, the microservices chatbot system will store the verified user data into the database. This step is crucial to keep the data secure and ensure that user information can be accessed and stored in a structured way.

Upon successful registration, the microservices chatbot system will display a success message to the user once their information has been stored successfully. This message indicates to the user that their account has been created and they can now use all the features of the

 doi:10.20944/preprints202407.0028.v1

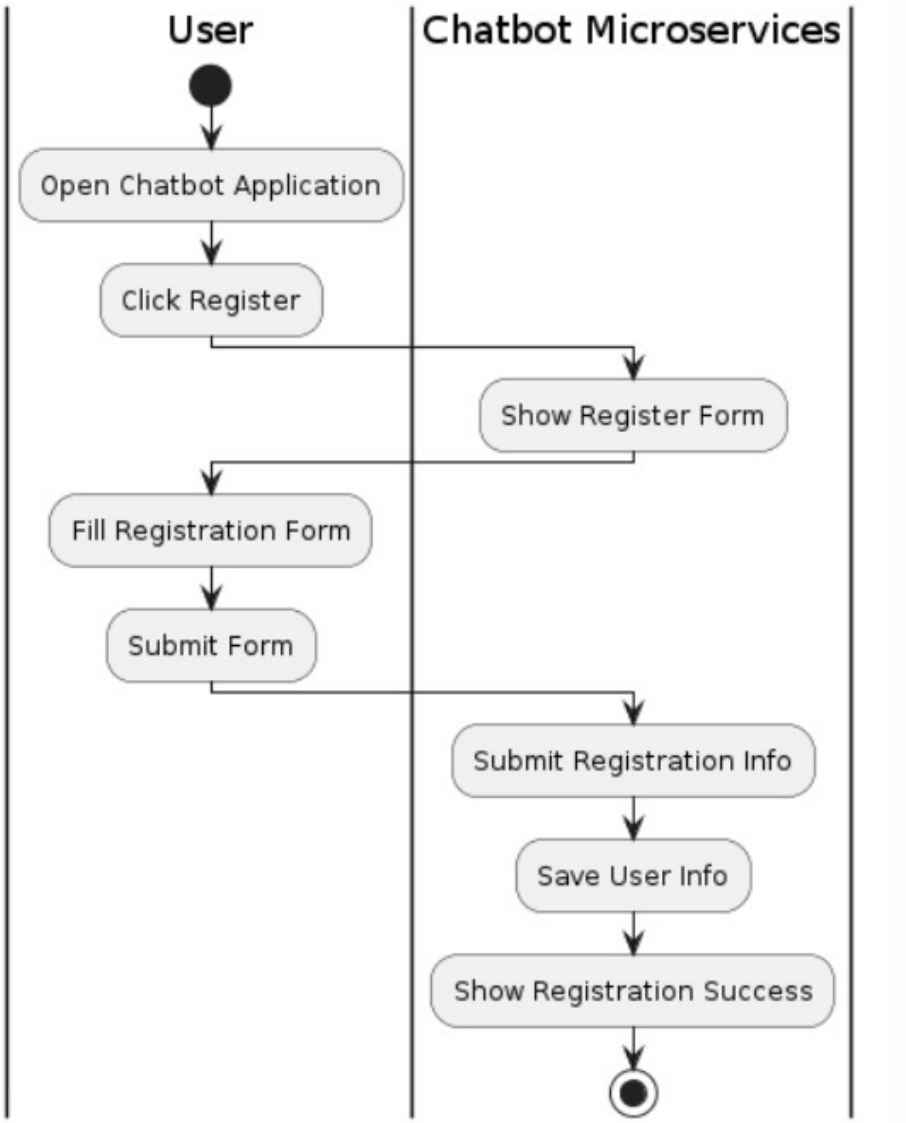

Figure 4. Activity Diagram Register

chatbot application. This message aims to enhance a smooth and effective user experience by completing the cycle of user interaction with the application during the registration process.

(b) Activity Diagram Login

Figure 5 shows an activity diagram that demonstrates how a user logs in to a chatbot application that uses microservices, which consists of various interactive steps performed by the user and the system. The start of the interaction occurs when the user opens the chatbot application, which is the first step in the "User" column. The user clicks the login button in the application interface once the application is opened. With this action, the microservices chatbot is notified to proceed to the next stage, which is to display the login form.

Once the microservice receives the user's signal, the login form is displayed on the "Chatbot Microservices" column. This step is important as it allows the user using the interface to enter their user email and password on the login form. Once the login form is displayed, the user then fills in the required information, such as the user's email and password. This is an important step where the user fills in data that will be verified by the system.

 doi:10.20944/preprints202407.0028.v1

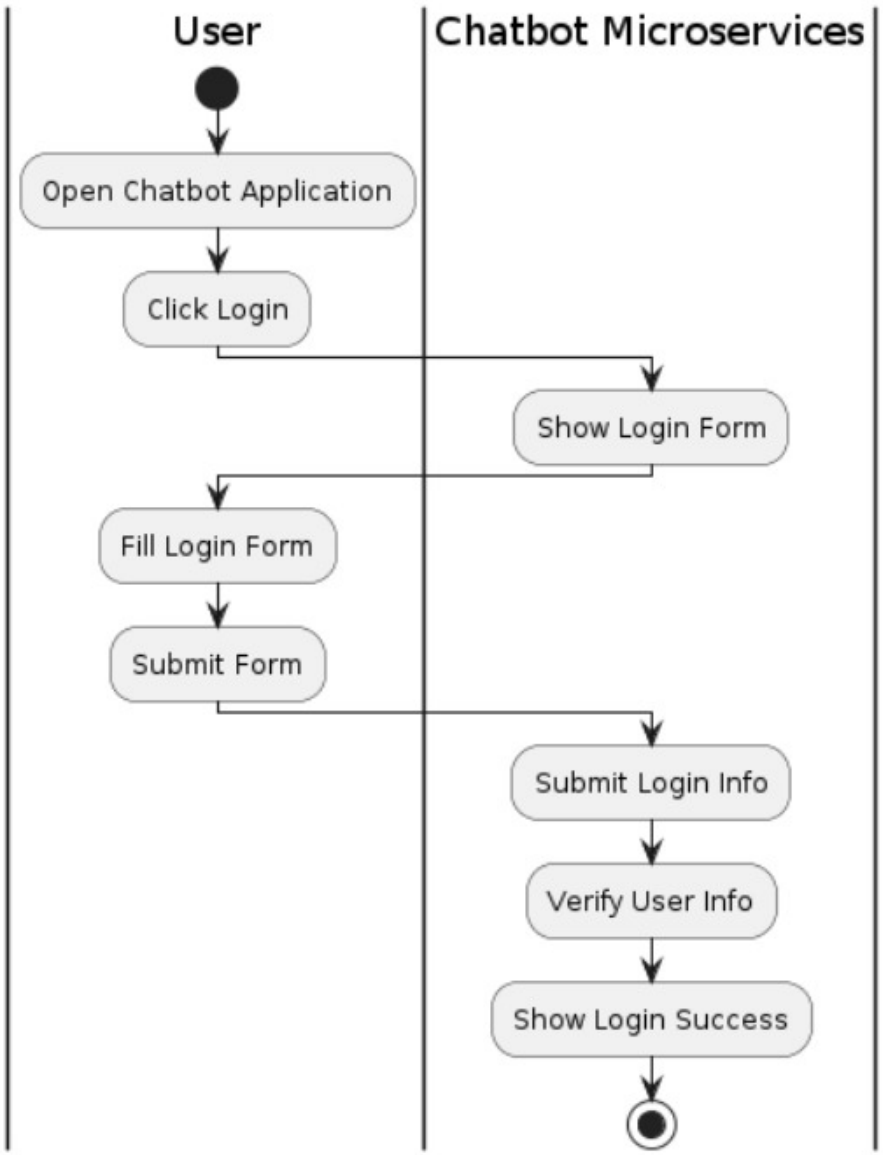

Figure 5. Activity Diagram Login

After filling out the login form, the user clicks the sign in button to submit it. By performing this action, the user's login information is sent to the microservices chatbot for further processing. Upon receiving this information, the service will verify the user's data. The verification process ensures that the user's email and password entered match the existing data in the system database. This verification process ensures that their account cannot be accessed by anyone other than the user.

The microservices chatbot will redirect the user to a prompt display when the user is successfully verified. This briefing serves as a confirmation that the user has successfully logged in and can access the available application features. In addition, this briefing provides assurance to the user that the login process has been completed correctly. This diagram clearly shows the continuous flow of interaction between the user and the system, which ensures a safe and efficient user experience.

(c) Activity Diagram Send Prompt

Figure 6 shows an activity diagram showing how the user sends commands to the chatbot application and how the microservices process those commands to provide an appropriate

 doi:10.20944/preprints202407.0028.v1

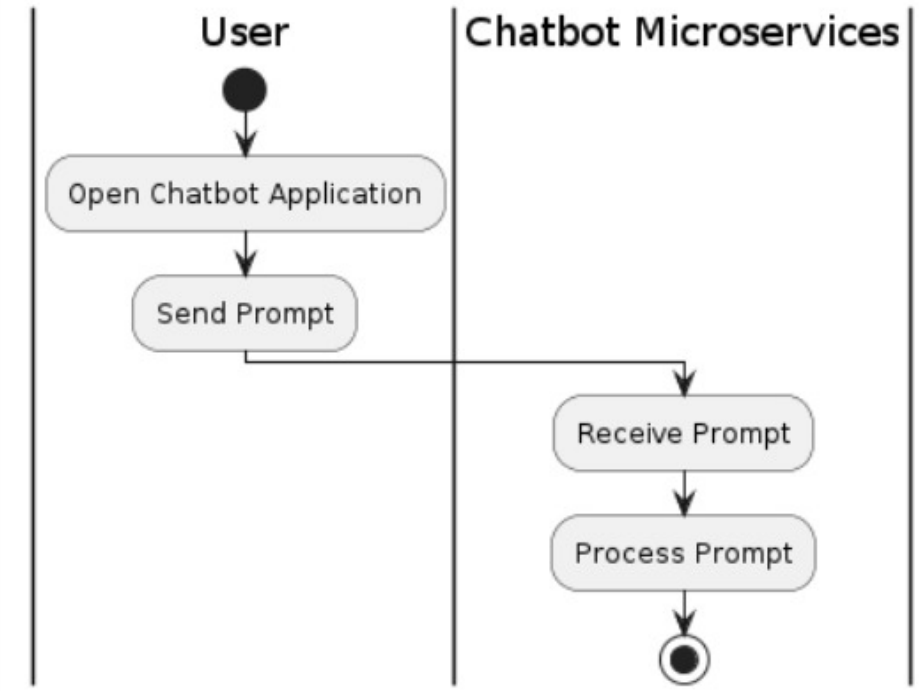

Figure 6. Activity Diagram Send Prompt

response. The chatbot application must be opened by the user, for the user to interact with the system at this stage. The user can ask questions through the application when starting it.

Furthermore, the user can send commands or questions to the chatbot. This activity will send a signal to the microservices chatbot to receive and act on the command. The user writes the question or command and sends it through the interface on this application. This is an important step because the user input will determine the next system interaction and response. At this point, the user expects the system to respond immediately with relevant information or actions.

If the user gives a prompt, the microservices chatbot processes it immediately. This activity is referred to as "Send Prompt" and is the point at which the system begins to understand and manage user input. This processing includes retrieving command data based on the user input. The microservices chatbot ensures that every command received is handled correctly and effectively.

In the last stage, the microservices chatbot process processes the commands received by the user. In this process, the system will call the Grok API to process the command data provided by the user, which then generates a response that is appropriate and relevant to the command. After the processing process is complete, the system will send the command back to the user through the application's chatbot interface.

(d) Activity Diagram Get Response

Figure 8 above shows an activity diagram depicting the interaction between "Chatbot Microservices" and "User" during the process of getting a response from the chatbot. This diagram consists of two columns displaying the two main actors, "Chatbot Microservices" and "User." The process starts in the "Chatbot Microservices" section with the "Generate Response" activity, where the chatbot processes the user input and generates the appropriate response.

Once the chatbot microservices has generated the response, the next step is the delivery of the response from the system to the user. At this stage, the information generated by

 doi:10.20944/preprints202407.0028.v1

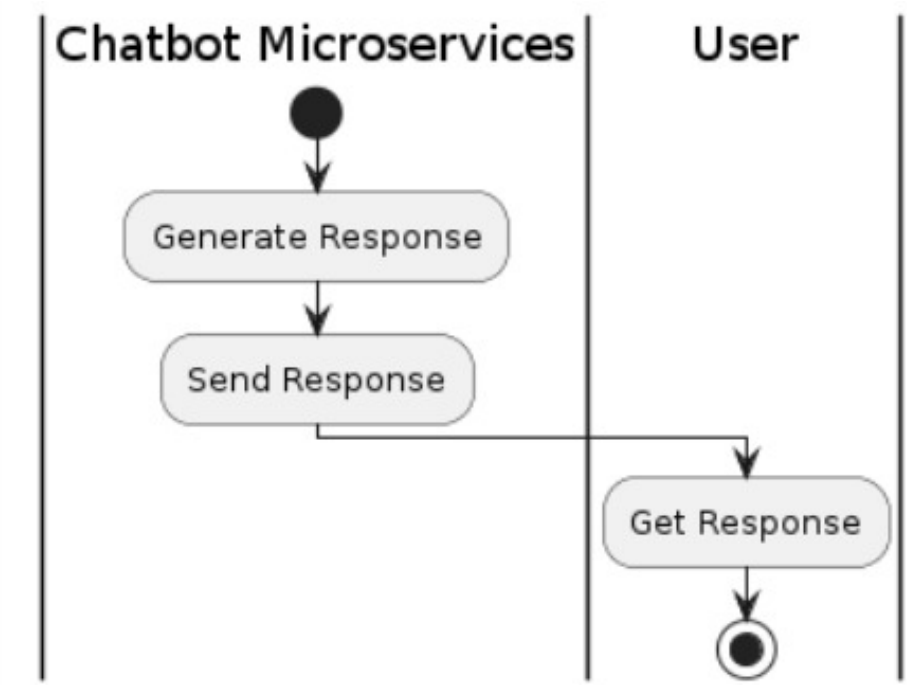

Figure 7. Activity Diagram Get Response

the chatbot is sent back to the user through a predefined communication channel. This response delivery process is crucial to ensure that the chatbot-generated information can be properly received by the user without any technical issues. This includes ensuring the response is received within a reasonable time and transmitting the data over a secure and reliable network.

On the user's side, the "Get Response" activity occurs after the chatbot has sent the response. Here, the user will receive and view the chatbot's response, which is crucial to determine if the user is satisfied with the information provided. This diagram shows a simple and direct communication flow between the chatbot and the user, showing how the user's message is processed and returned as a response.

In addition to the use case diagram and activity diagram in the development of this system, there is a class diagram that shows the class structure and the relationship between classes in the system to be created. This class diagram also shows the main entities, as well as their attributes and methods, and the relationships and dependencies between classes. The figure below shows the details of the structure :

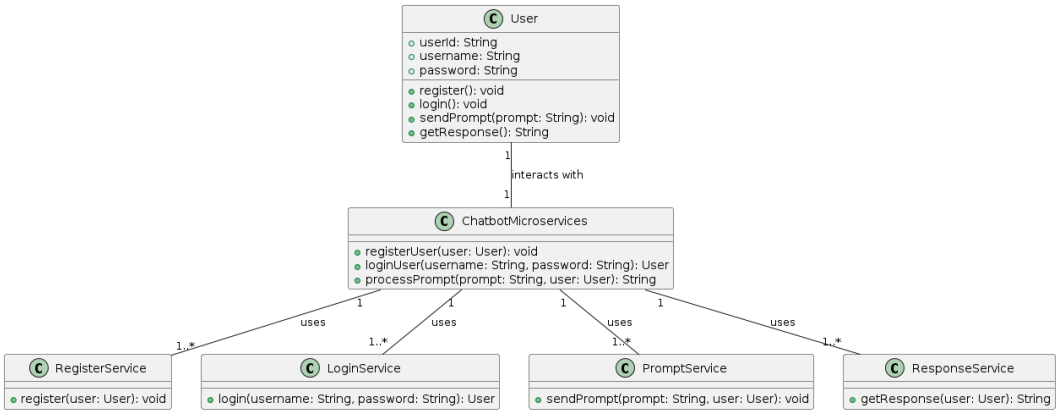

Figure 8. Class Diagram Chatbot Microservices

 doi:10.20944/preprints202407.0028.v1

In Figure 9 above there are several main classes, including User, Chatbot Microservices, Register, Login, Prompt, and ResponseService. Each class has specific attributes and methods that govern the behavior and operation of the system. Below is an explanation for each class :

(a) User Class

The User class contains attributes such as userId, username, and password that represent the identity of the user. Methods in this class include register(), login(), sendPrompt(prompt: String), and getResponse(): String, which allow users to register themselves, login to the system, send a prompt, and get a response from the chatbot. These classes act as entities that interact directly with the user interface of the chatbot system.

(b) ChatbotMicroservices Class

The ChatbotMicroservices class serves as a coordinating center that handles the interaction between the user and the various microservices that help the chatbot operate. Within this class, methods include registerUser(user: User), loginUser(username: String, password: String): User, and processPrompt(prompt: String, user: User): String. This class is responsible for calling services such as RegisterService, LoginService, PromptService, and ResponseService as needed based on user actions.

(c) RegisterService Class

The RegisterService class is one of the microservices that handles user registration. The register(user: User): void method in this class is called by ChatbotMicroservices when a new user registers. This ensures that the registration process is managed separately and modularly, which allows for greater system maintenance and development. RegisterService stores and validates the newly registered user data.

(d) LoginService Class

The LoginService class handles the user login process. Method login(username: String, password: String): User is called by ChatbotMicroservices when the user tries to log in. This service verifies the user's credentials against the data stored in the system and returns a User object if the validation is successful. Separating the login logic into different services makes it easier to organize and secure the system.

(e) PromptService Class

The PromptService class is used by ChatbotMicroservices to handle prompts submitted by users. Method sendPrompt(prompt: String, user: User): void in this class is called to process the user prompt. This service will process the prompt data which will be passed to the API and will pass it to GetResponse and determine the next step in the interaction.

(f) GetResponse Class

The ResponseService class receives and sends a response to the user based on the processed prompt. ChatbotMicroservices uses the getResponse(user: User) method to get the response corresponding to the user's prompt. The system can be easily updated and extended to improve the quality and relevance of responses by managing responses separately.

From the explanation above, it can be concluded that this class diagram shows the modular and structured design of the chatbot system, where each microservices service handles a specialized aspect of the chatbot operation. This allows for easier maintenance, greater scale, and better security as different features are separated into services that can be set up and managed independently. The ChatbotMicroservices class connects users to these services and ensures that the system workflow runs smoothly and efficiently.

3. Development

Development in the context of software engineering refers to the comprehensive process of creating, testing, deploying, and maintaining software applications. This involves several stages including planning, coding, testing, and deployment. Each stage requires different skills and tools to ensure the final product is reliable, scalable, and meets user requirements. Software

 doi:10.20944/preprints202407.0028.v1

development can be divided into two main categories: frontend development and backend development, each focusing on different aspects of the application.

In frontend development, developers concentrate on the visual and interactive parts of a web application. This includes everything that users see and interact with, such as layouts, buttons, and forms. Frontend developers use HTML, CSS, and JavaScript, along with frameworks like React, Angular, and Vue.js, to build responsive and user-friendly interfaces. Their goal is to create a seamless and engaging user experience, ensuring that the application looks good and functions well on various devices and browsers.

Backend development, on the other hand, deals with the server-side of the application. Backend developers focus on the logic, database interactions, user authentication, and server configuration that power the frontend. They use languages such as Python, Java, Ruby, and JavaScript (Node.js) along with frameworks like Django, Spring, Ruby on Rails, and Express.js. Their work ensures that data is processed efficiently, stored securely, and delivered correctly to the frontend. Backend developers also implement security measures, manage APIs, and ensure the application can scale to handle a growing number of users.

Both frontend and backend development require a collaborative effort to ensure the application works as intended. This involves constant communication and synchronization between frontend and backend teams to integrate the two sides smoothly. Additionally, modern development practices often include DevOps, which combines software development (Dev) and IT operations (Ops) to shorten the development lifecycle and provide continuous delivery with high software quality. Tools like Docker, Kubernetes, Jenkins, and CI/CD pipelines are used to automate and streamline the deployment process, ensuring that updates and new features can be delivered quickly and reliably.

(a) Frontend Development

Frontend development focuses on the client-side aspects of web development, involving everything that users see and interact with. This includes creating intuitive and attractive user interfaces (UI) using technologies like HTML, CSS, and JavaScript. HTML (HyperText Markup Language) is used to structure web pages, CSS (Cascading Style Sheets) for styling and layout, and JavaScript for adding interactivity and dynamic features to web pages.

Modern frameworks and libraries like React, Angular, and Vue.js are highly popular in frontend development. These frameworks provide tools and features to build complex web applications more efficiently and in a structured manner. For example, React allows developers to build UIs with reusable components, making maintenance and the addition of new features easier.

User experience (UX) is a primary focus in frontend development. This includes ensuring responsiveness, meaning that the website or application can be accessed well on various devices and screen sizes. Responsive design is typically achieved through the use of CSS media queries and testing across different devices and browsers to ensure consistency.

Additionally, frontend development involves performance optimization to ensure that the website or application runs quickly and smoothly. This includes techniques such as file minification, image optimization, and browser caching. Good performance not only enhances user experience but also contributes to SEO (Search Engine Optimization), improving the website's visibility and ranking in search engines.

(b) Backend Development

Backend development focuses on the server-side of a web application, managing business logic, databases, and user authentication. The backend ensures that all requests from the frontend are handled correctly and that the necessary data is provided. Common technologies used in backend development include programming languages like Python,

 doi:10.20944/preprints202407.0028.v1

Java, Ruby, and JavaScript (Node.js), along with frameworks like Django, Spring, Ruby on Rails, and Express.js.

One of the main tasks in backend development is database management. Databases store all the data used by the application, such as user information, content, and preferences. Backend developers must design and manage databases efficiently using database management systems like MySQL, PostgreSQL, MongoDB, and Redis. They must also ensure data security and integrity, including creating backups and data recovery plans.

Security is a critical aspect of backend development. This involves protecting user data through encryption, managing authentication and authorization, and preventing common attacks like SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). Backend developers must continually update their knowledge of security best practices and apply them consistently.

Scalability is another important factor in backend development. Web applications need to handle increasing numbers of users and data without degrading performance. This can be achieved through efficient architecture, such as using microservices, good server load management, and performance monitoring tools to continuously optimize the application.

i. Front API

   Front API refers to the application programming interface (API) used by the frontend to communicate with the backend. It is a crucial layer that allows the web application to fetch data from the server and send user input back to the server. Front API typically uses HTTP/HTTPS protocols and data formats like JSON (JavaScript Object Notation) for efficient and readable data exchange.

   Designing a Front API should ensure it is intuitive and easy to use for frontend developers. The API should provide clear and detailed documentation about available endpoints, required parameters, and expected data structures. This helps frontend developers understand how to use the API correctly and reduces the risk of integration errors.

   Security is an important aspect of Front API design. Sensitive data transmitted between the frontend and backend should be encrypted to prevent eavesdropping by unauthorized parties. Additionally, the API should use strong authentication and authorization mechanisms to ensure that only authorized users can access and modify data.

   Performance is also a major consideration in Front API design. API requests should be optimized to minimize latency and response time. This can include techniques such as reducing payload sizes, using caching where appropriate, and batching data requests to reduce the number of API calls needed. A fast and responsive API is crucial for a good user experience.

ii. Response API

   Response API is the part of the backend responsible for generating and sending responses to requests received from the frontend. These responses can be the requested data, confirmation of actions performed, or error messages if the request cannot be processed. The responses provided must be timely and accurate to ensure the application functions smoothly.

   The design of the Response API must ensure that the data sent back to the frontend is complete and in the correct format. For example, if the frontend requests user information, the Response API should return all related data in a predefined JSON format. Errors and exceptions should also be well-handled, providing clear and informative error messages to the frontend.

   Security in the Response API is crucial for protecting user data. Responses should be sanitized to prevent leakage of sensitive information and injection attacks. Additionally, authentication and authorization should be enforced to ensure that only authorized

 doi:10.20944/preprints202407.0028.v1

users receive specific data. Using security tokens and proper CORS (Cross-Origin Resource Sharing) settings also helps enhance security.

The performance of the Response API affects how quickly data can be displayed on the frontend. Fast and efficient responses are necessary for a good user experience. This can be achieved by optimizing database queries, using caching where appropriate, and minimizing the size of the data sent. Regular monitoring and performance tuning of the API are also important to maintain responsiveness.

iii. Groq API

Grok API refers to the process of understanding and capturing the context of data sent and received through the API. In the context of a chatbot or other AI systems, Grok API is responsible for analyzing user input, understanding its intent, and providing relevant responses. This involves using technologies like natural language processing (NLP), machine learning (ML), and artificial intelligence (AI) algorithms.

One important aspect of Grok API is its ability to handle various forms of user input. Input can be text, voice, or other data, and Grok API must be able to accurately analyze and interpret this data. For instance, in a chatbot, Grok API analyzes the user's question to determine the underlying intent and searches for the most appropriate answer in a database or AI model.

Security in Grok API is also crucial, especially since this API often handles sensitive user data. Input and output must be sanitized to prevent code injection attacks. Additionally, authentication and authorization should be enforced to ensure that only legitimate requests are processed by the API.

The performance of Grok API is key to providing quick and relevant responses. NLP and ML algorithms must be optimized to process input quickly and accurately. This can include using efficient machine learning models, caching techniques, and optimizing database queries. Good performance not only enhances user experience but also allows the system to handle more requests efficiently.

4. Testing

To ensure that the API works properly and meets the expected performance needs, we will conduct tests in the test section. Various elements are tested in these tests, such as functionality validation, performance, security, and compatibility. Before the API is deployed in a production environment, thorough testing helps us find and fix any issues that may occur.

API performance testing was done using Postman. This test involves sending multiple requests to predefined API endpoints. The purpose of this test is to measure throughput, error rate, and average response time.This performance test uses Postman to simulate the actual API workload. By sending multiple requests in various situations, we can find out how the API functions under various load conditions.

The throughput metric shows the number of requests that can be processed by the API in a unit of time, indicating the maximum capacity of the API. The average response time shows how fast the API responds to requests, which is important for ensuring a good user experience.

Meanwhile, the error rate records the percentage of requests that fail or generate an error response, which can indicate API stability or reliability issues. D This is very important to ensure that applications using the API are operating properly.The results of the API testing for the four endpoints, which include login, add prompt, add prompt & add response, and get all responses, are shown below:

According to the test results, as shown in the figure above, there were 2,014 requests sent and the API successfully handled a throughput of 28.27 requests per second. This demonstrates the API's ability to efficiently handle multiple requests, which is critical for applications that require high scale and performance.

 doi:10.20944/preprints202407.0028.v1

Total requests sent	Throughput	Average response time	Error rate
2,014	28.27 requests/second	83 ms	0.00 %

Figure 9. Summary Testing API

The average response time of 83 ms shows that the API is able to respond to requests quickly, which is an excellent performance indicator. This fast response time is important to ensure that users have a responsive and fast experience. With an error rate of 0.00%, testing showed that there were no errors. This indicates a high level of reliability, which makes the API safe to use in production.

Overall, these test results show that the API performs very well in terms of throughput, response time, and error rate. Although the results are very positive, it is recommended to conduct further testing on a larger scale and continuous monitoring to ensure performance remains consistent across different load conditions.

In addition to API testing, performance testing was also conducted on the client side. The average time is 2-3 seconds from when the user sends a prompt until the system responds, as shown in the figure below.

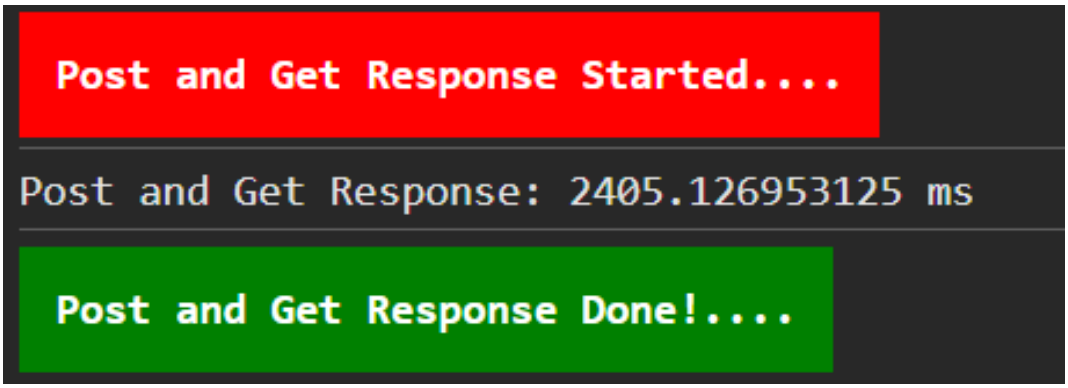

Figure 10. Testing Prompt And Response at Client Side

The process started with the message "Post and Get Response Started", as shown in the image. The recorded response time is about 2405 milliseconds, or 2.4 seconds. Once the response is received, the process ends with the message "Post and Get Response Done!". These results show that the system works well and is responsive to user requests, providing an optimal user experience with fast response times.

5. Deployment

Vercel is a popular platform for deploying web applications, offering a seamless and efficient workflow for developers. It supports a wide range of frontend frameworks such as React, Next.js, Vue.js, and Angular, making it a versatile choice for modern web development. Deployment with Vercel is straightforward; developers simply push their code to a version control system like GitHub, GitLab, or Bitbucket. Vercel then automatically detects changes, builds the project, and deploys it to a global content delivery network (CDN). This ensures that the application is highly available and performs well regardless of the user's location.

One of the standout features of Vercel is its ability to automatically optimize performance. By distributing your application across multiple edge locations, Vercel minimizes latency and

 doi:10.20944/preprints202407.0028.v1

provides fast load times for users around the world. It also offers features like image optimization and server-side rendering (SSR) for frameworks such as Next.js, enhancing both the speed and SEO of your application. Additionally, Vercel provides a user-friendly dashboard where developers can monitor deployments, manage environment variables, and roll back to previous versions if necessary, ensuring a smooth and manageable deployment process.

Vercel also supports serverless functions, which allow developers to deploy backend logic without managing traditional server infrastructure. These serverless functions, built on top of AWS Lambda, provide scalable, on-demand backend services for handling tasks like API requests and data processing. This integration simplifies the development workflow by allowing both frontend and backend code to be deployed together. Furthermore, Vercel offers automatic HTTPS, custom domain support, and built-in logging and monitoring, making it a comprehensive solution for deploying modern web applications.

(a) Frontend Deployment with Vercel

Frontend deployment using Vercel is a streamlined process designed to make deploying web applications fast and efficient. Vercel is particularly popular for its seamless integration with modern frontend frameworks like React, Next.js, Vue.js, and Angular. Developers can push their code to a version control system like GitHub, GitLab, or Bitbucket, and Vercel will automatically detect changes, build the project, and deploy it to a global content delivery network (CDN).

One of the key benefits of using Vercel for frontend deployment is its automatic scaling and performance optimization. Vercel ensures that your application is distributed across multiple edge locations, which reduces latency and ensures fast load times for users around the world. Additionally, Vercel provides features like image optimization and server-side rendering (SSR) for frameworks like Next.js, further enhancing the performance and SEO of your application.

Vercel also offers a rich set of developer tools and a user-friendly dashboard that allows developers to monitor deployments, roll back to previous versions, and manage environment variables with ease. The integration with popular Git repositories means that developers can set up a continuous deployment pipeline effortlessly. Every push to the main branch can trigger a new deployment, ensuring that the latest code is always live.

Moreover, Vercel supports custom domains and provides HTTPS out of the box, enhancing the security and professionalism of your deployed application. With its automatic SSL certification, developers don't need to worry about configuring security certificates. Vercel's platform also supports serverless functions, allowing developers to extend the functionality of their frontend applications without managing backend servers, thus simplifying the development and deployment process.

(b) Backend Deployment with Vercel

While Vercel is primarily known for frontend deployment, it also supports backend functionalities through serverless functions. This approach allows developers to deploy backend logic without managing traditional server infrastructure. Vercel's serverless functions are built on top of AWS Lambda, providing scalable, on-demand backend services that can handle various tasks like API requests, data processing, and more.

Deploying backend code on Vercel involves writing serverless functions using Node.js, Go, or Python. These functions can be placed in a designated directory in your project, and Vercel will automatically deploy them alongside your frontend code. This integration ensures a seamless connection between your frontend and backend, simplifying development and deployment workflows. Each function is exposed as an API endpoint, making it easy to interact with from your frontend application.

 doi:10.20944/preprints202407.0028.v1

One of the main advantages of using Vercel for backend deployment is the automatic scaling and cost efficiency of serverless functions. Unlike traditional servers, which run continuously and incur costs even when not in use, serverless functions only run when invoked. This means you only pay for the compute time you use, which can lead to significant cost savings, especially for applications with variable or low traffic.

Vercel also provides built-in logging and monitoring tools for serverless functions, allowing developers to track function executions, performance metrics, and errors. This visibility is crucial for maintaining the reliability and performance of your backend services. Additionally, Vercel's global infrastructure ensures that your serverless functions are executed close to your users, reducing latency and improving response times. This makes Vercel an attractive option for developers looking to deploy both frontend and backend components efficiently.

6. Review

The review phase is a critical part of both the development and deployment processes when using Vercel. This stage involves evaluating the work done, assessing the deployment process, and ensuring that the application meets the desired standards and requirements. By systematically reviewing both the development and deployment outcomes, teams can identify areas for improvement and ensure the quality and performance of their application.

During the development review, the team examines the newly implemented features and changes in the application. This includes checking the frontend UI for usability, design consistency, and responsiveness across different devices and browsers. For backend development, the review focuses on the logic, database interactions, and security measures implemented. Tools like Vercel's live previews and deployments make it easy to demonstrate and validate these changes. The team gathers feedback from stakeholders and end-users, which helps in identifying any issues or enhancements needed before moving to production.

The deployment review centers on the process and results of deploying the application using Vercel. The team assesses the automated build and deployment workflow, ensuring that the application was deployed without errors and is functioning as expected. Vercel's detailed logs and monitoring tools are used to analyze the performance and any potential issues. The review checks for successful integration of new features, verifies that previous functionalities are not broken, and evaluates the overall application performance in a live environment. This step is crucial for maintaining the integrity and reliability of the application.

Combining the development and deployment reviews ensures a thorough evaluation of the entire process from code changes to live deployment. This comprehensive review helps in maintaining high standards and continuous improvement. Feedback from these reviews is used to refine both the development practices and the deployment pipeline. By leveraging Vercel's capabilities, teams can quickly iterate based on review findings, ensuring that the application evolves effectively and remains aligned with user expectations and business goals. This iterative review process is key to delivering a robust and high-quality application.

4.2. Discussion

This software application development process utilizes Next.js for frontend development and Express.js for backend development. Next.js provides the ability to build intuitive and responsive user interfaces with high efficiency, thanks to features such as server-side rendering and static site generation. Meanwhile, Express.js provides a powerful framework for handling server-side logic, interaction with databases, as well as user authentication, which supports efficient data processing and secure storage. To enhance the functionality of the chatbot, we implemented the Grok API on the frontend. This API uses natural language processing (NLP) and machine learning (ML) algorithms to effectively understand and respond to user input. The integration of these technologies has proven successful, contributing to scalability, reliability and providing a seamless user experience.

 doi:10.20944/preprints202407.0028.v1

Thorough testing was conducted using Postman to check the performance of the API at various endpoints, such as login, adding a prompt, adding a prompt, and getting all responses. The results show that the API can handle 2,014 requests with a throughput of 28.27 per second, an average response time of 83 milliseconds, and an error rate of 0.00%. This shows that the API is efficient and reliable, and is able to support applications with a lot of traffic without compromising its performance.

In addition, client-side performance testing shows fast response times (about two to three seconds) from the time a user submits a request until the system responds. This fast response time improves user experience and ensures that both frontend and backend components are well implemented. The level of robustness and strength of our system is demonstrated by the correlation between low error rates and high throughput. Larger scale testing and continuous monitoring will be required in the future to ensure consistent and optimal performance under various load conditions and to provide a consistent and optimal user experience.

5. Conclusion

"Real-Time Chatbot: Microservices Implementation in Distributed System Architecture" reveals that the use of microservices can improve the efficiency and productivity of interactive systems. This method allows each part of the system to work separately while remaining integrated, which enables more responsive and reliable operations.

Key functional requirements, such as real-time communication, secure authentication with JSON Web Token (JWT), and integration with Groq API, are well met. The use of HTTP protocol enables instant message exchange without the need to reload the page, while JWT ensures a fast and secure authentication process that keeps user data confidential. In addition, integration with Groq API allows the system to be connected to various services and resources.

In addition, the combined use of MongoDB for unstructured data and SQL for structured data enables effective data management. This method provides flexibility in handling different types of data as it ensures data integrity and optimal system performance. This is important to ensure the chatbot can store user data properly. In addition, scalability, performance, and other non-functional needs are well catered for. With low latency and careful capacity planning mechanisms, the system enables quick responses for users.

Various diagrams demonstrate the system design, including architecture, use case, activity, and class diagrams by showing a modular and organized structure. Each microservice handles a specific part of the chatbot operation, which allows for easier maintenance and smooth upgrades. Therefore, the system is not only robust, secure, but also efficient in handling large and complex user interactions and provides a solid foundation for future development.

Acknowledgments: The author's express their sincere gratitude to Mr. Wisnu for his valuable guidance and support during this research. We would also like to express our gratitude to our teammates who have worked together with dedication on this project. The encouragement and support from friends is also unforgettable. We would also like to acknowledge the support of the Informatics Department of UIN Sunan Gunung Djati Bandung which has been instrumental in supporting this research.

References

1. C. DeVon. On chatgpt's one-year anniversary, it has more than 1.7 billion users—here's what it may do next. [Online]. Available: https://www.cnbc.com/2023/11/30/chatgpts-one-year-anniversary-how-the-viral-ai-chatbot-has-changed.html
2. Shabani, I.; Mëziu, E.; Berisha, B.; Biba, T. Design of modern distributed systems based on microservices architecture. *International Journal of Advanced Computer Science and Applications* **2021**, *12*.
3. Islam, M.R.A. Adding context awareness to chatbots with microservices: Case Forest Companion. Master's thesis, 2023.
4. Deriyanto, S.P.; Santoso, H.A. Development of Bot for Microservices Server Monitoring Using Life Cycle Approach to Network Design Method. *Juita* **2020**, *8*. doi:10.30595/juita.v8i2.8422.

5. Roca, S.; Sancho, J.; García, J.; Álvaro Alesanco. Microservice chatbot architecture for chronic patient support. *Journal of Biomedical Informatics* **2020**, *102*, 103305. doi:https://doi.org/10.1016/j.jbi.2019.103305.
6. Deriyanto, S.P.; Santoso, H.A. Development of Bot for Microservices Server Monitoring Using Life Cycle Approach to Network Design Method. *JUITA: Jurnal Informatika* **2020**, *8*, 141–147, [2579-8901].
7. Shiddiqramzy, H.; Sediyono, E. Perancangan Aplikasi Chat Realtime sebagai Media Bercerita Berbasis Android. *Jurnal JTIK (Jurnal Teknologi Informasi dan Komunikasi)* **2023**, *7*, 328–336, http://xxx.lanl.gov/abs/http://journal.lembagakita.org/index.php/jtik/article/view/782/. doi:10.35870/jtik.v7i2.782.
8. Rizki, M.; Fitriansyah, A.; Narji, M. Aplikasi Chatbot Sebagai Layanan Live Chat Untuk Penerimaan Mahasiswa Baru Menggunakan Metode Word Stemming Dengan Regular Expression Pattern Matching. *Jurnal Elektro & Informatika Swadharma (JEIS)* **2023**, *3*, 50–54, http://xxx.lanl.gov/abs/http://journal.swadharma.ac.id/index.php/jeis/article/view/123/. doi:10.1234/jeis.v3i2.123.
9. Hikmah, N.; Ariyanti, D.; Pratama, F.A. Implementasi Chatbot Sebagai Virtual Assistant di Universitas Panca Marga Probolinggo menggunakan Metode TF-IDF. *JTIM: Jurnal Teknologi Informasi dan Multimedia* **2022**, *4*, 133–148, [https://journal.sekawan-org.id/index.php/jtim/article/view/225/]. doi:10.35746/jtim.v4i2.225.
10. Swara, G.Y.; Kom, M.; Pebriadi, Y. Rekayasa perangkat lunak pemesanan tiket bioskop berbasis web. *Jurnal Teknoif Teknik Informatika Institut Teknologi Padang* **2016**, *4*, 27–39.
11. Naufal Faruq, M.; Maryam, M. IMPLEMENTASI METODE AGILE PADA PENGEMBANGAN APLIKASI MANAJEMEN PENGELOLAAN LAYANAN WIFI. *JATI (Jurnal Mahasiswa Teknik Informatika)* **2024**, *7*, 3472–3478. doi:10.36040/jati.v7i6.7868.