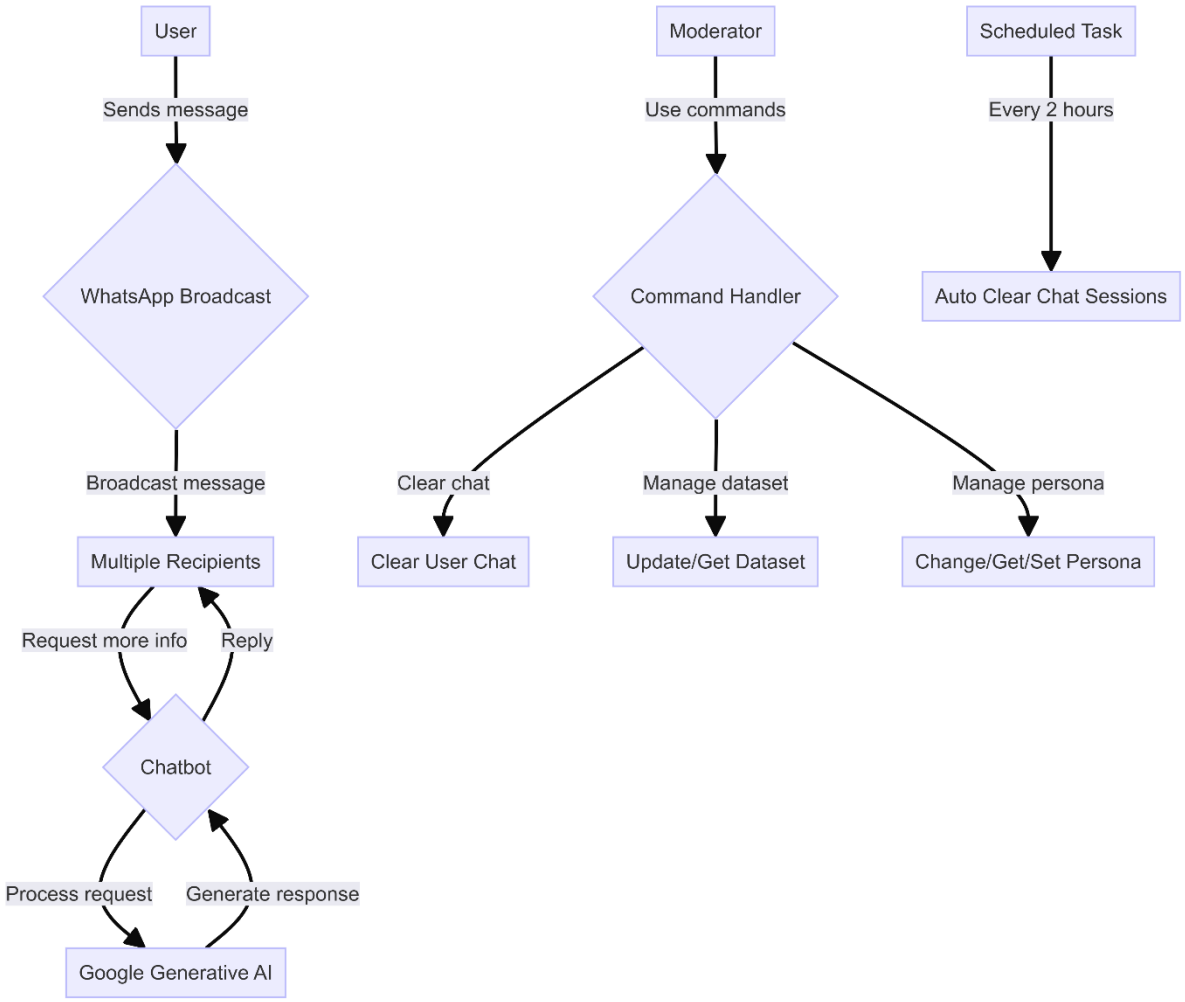
FLOWCHART



This flowchart represents the system's operational flow using graph theory principles. It's a directed graph where nodes represent processes or decision points, and edges represent the flow of information or control.

Key components:

- Initial node (User): The source of input to the system.
- Decision nodes (rhombuses): Represent branching logic (e.g., Chatbot, Command Handler).
- Process nodes (rectangles): Represent specific operations or subsystems.
- Edges: Directed connections showing the flow of information and control.

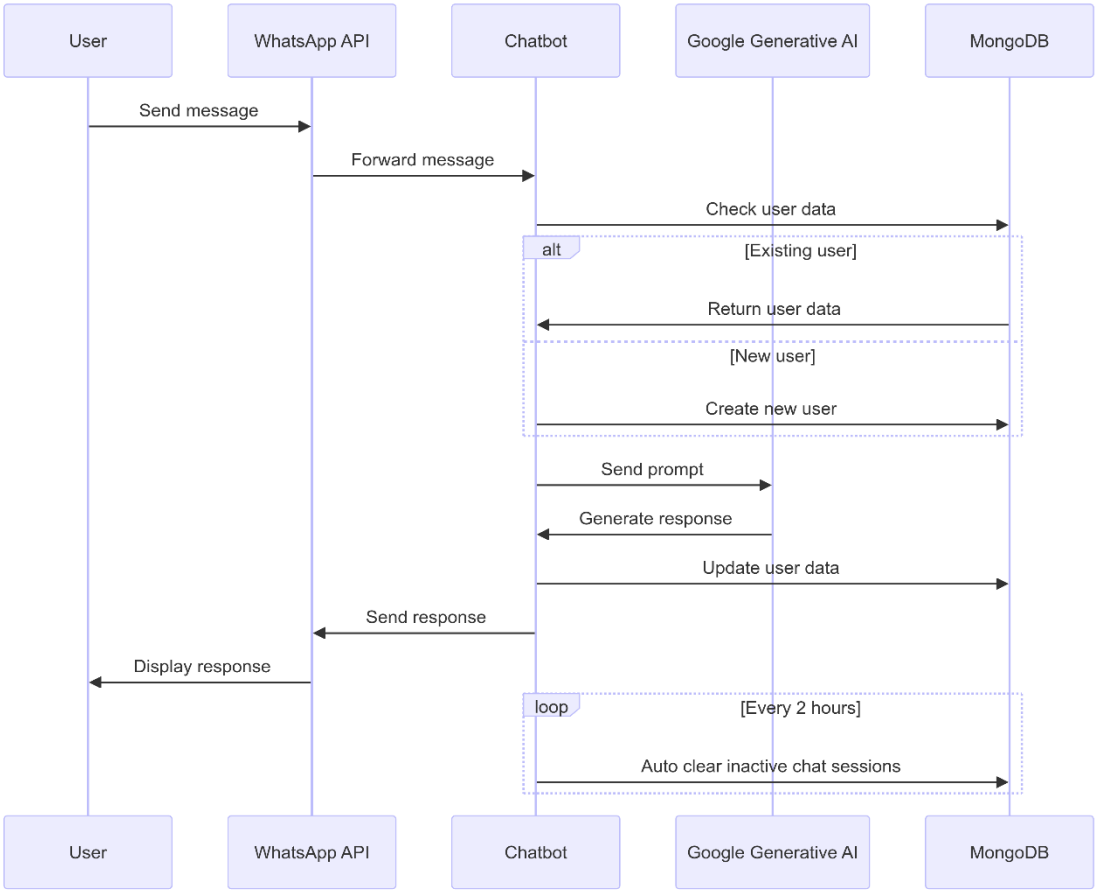
The diagram illustrates two main workflows:

1. User interaction flow: User → WhatsApp Broadcast → Recipients → Chatbot → Google Generative AI
2. Moderation flow: Moderator → Command Handler → Various administrative actions

It also includes an isolated process (Scheduled Task) that performs periodic maintenance.

This representation allows for a clear visualization of the system's logic and helps in identifying potential bottlenecks or parallel processes.

SEQUENCE



The sequence diagram is based on the Unified Modeling Language (UML) and represents the temporal ordering of interactions between system components.

Key elements:

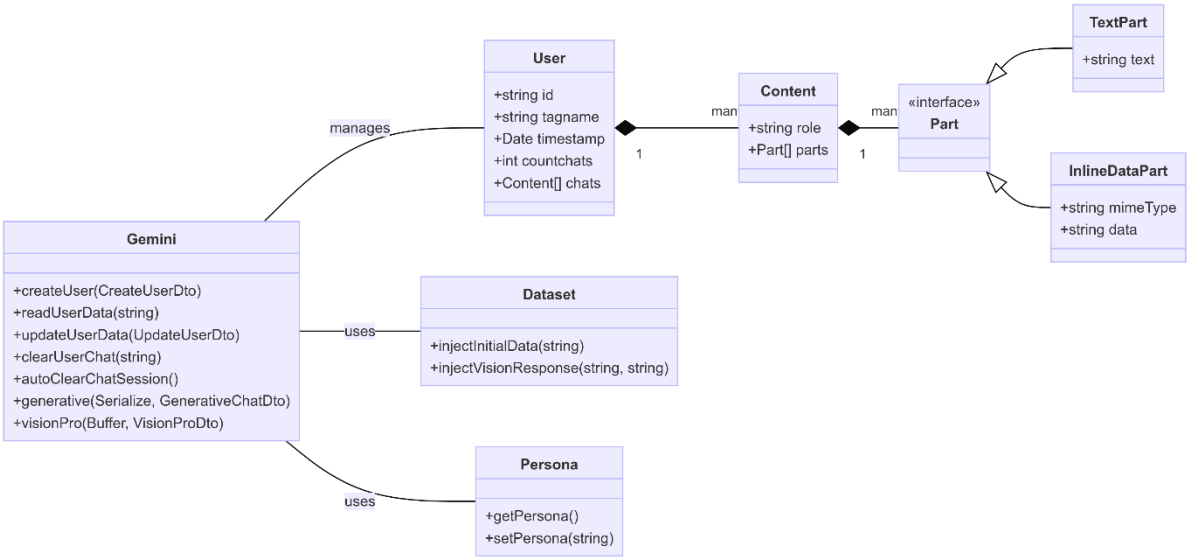
- Vertical dimension: Represents time, flowing from top to bottom.
- Horizontal dimension: Shows different participants (objects or processes).
- Arrows: Represent messages or method calls between participants.
- Activation boxes: Indicate when a participant is active or processing.

The diagram illustrates:

- Asynchronous communication: Messages don't block the sender (e.g., User to WhatsApp).
- Synchronous communication: Some operations wait for a response (e.g., Chatbot to Google Generative AI).
- Conditional logic: Represented by the alt fragment for new vs. existing users.
- Looping behavior: Shown by the loop fragment for periodic chat clearing.

This diagram is particularly useful for understanding the system's runtime behavior, message passing, and the order of operations. It helps in identifying potential race conditions or deadlocks in concurrent systems.

CLASS DIAGRAM



The class diagram is another UML diagram type, representing the static structure of the system using object-oriented programming principles.

Key components:

- Classes: Represented by rectangles (e.g., User, Content, Gemini).
- Attributes: Listed within classes (e.g., id, tagname in User).
- Methods: Also listed within classes (e.g., createUser, updateUserData in Gemini).
- Relationships:
    - Association: Simple connections between classes.
    - Composition: Represented by filled diamonds (e.g., User composed of Content).
    - Inheritance: Represented by arrows (e.g., TextPart inheriting from Part).

Scientific aspects:

- Encapsulation: Classes bundle data (attributes) and operations (methods).
- Inheritance: Demonstrated by the Part interface and its implementations.
- Polymorphism: Implied by the interface-implementation relationship of Part.
- Composition: Shows "has-a" relationships, indicating object lifecycle dependencies.

This diagram is crucial for understanding the system's architecture, dependencies between classes, and the overall data model. It aids in assessing the system's modularity, extensibility, and adherence to object-oriented design principles.