WasteSense - Efficient Waste Management System for Urban Areas

1. Introduction

The WasteSense Project is a comprehensive waste management system designed to optimize urban waste collection, segregation, and complaint management. It leverages IoT, microservices architecture, and cloud-based solutions to provide real-time data collection and efficient communication between waste management stakeholders. The system aims to reduce overflow situations, streamline collection schedules, and encourage responsible waste disposal through advanced technology.

2. Project Background

Efficient waste management is a critical challenge faced by urban areas due to population growth and increasing waste generation. Overflowing waste bins lead to health hazards, environmental pollution, and logistical issues for waste management services. Current systems often lack real-time monitoring, resulting in delayed actions and inefficient resource allocation.

WasteSense aims to address these challenges by providing a system that integrates IoT sensors, cloud services, and user interfaces. This system allows for real-time monitoring of waste bin levels, automated alerts, optimized collection routes, and user participation in waste management through a complaint mechanism. The project benefits city municipalities, waste management agencies, and the general public by enhancing the efficiency of waste collection and ensuring cleaner environments.

3. Project Objectives

The primary goals of the WasteSense Project are:

- **Real-time Monitoring**: Implement IoT-enabled waste bins that report their fill levels to a central server, enabling continuous tracking of waste levels.
- **Optimized Collection Schedules**: Develop a scheduling system that reduces unnecessary collection trips and ensures timely waste disposal.

- **User Engagement**: Create an interface for users to report issues or view nearby bins, enhancing public participation in waste management.
- **Automated Alerts**: Notify administrators and relevant authorities of critical events such as full waste bins or adverse weather conditions that may impact collection.
- **Secure Authentication**: Use JWT-based authentication to ensure secure access to system features and data, protecting user information.
- **Modular Architecture**: Implement a scalable microservices-based system to facilitate future enhancements and maintenance.

4. Functional Requirements

The WasteSense system is designed to meet specific functional requirements that ensure it serves its intended purpose effectively. These include:

1. User Management:

- Users can register and log in to the system.
- Admins have the ability to register, log in, and manage user access.

2. Waste Bin Monitoring:

- The system should allow users to view the nearest waste bins based on their location.
- Users can check the real-time status of waste bins (e.g., current fill level).
- Admins can receive alerts when bins are full or approaching capacity.

3. Notification Service:

- Notifications are sent to users and admins regarding bin status, collection schedules, and alerts (e.g., weather-related delays).
- Users receive timely reminders for collection schedules and important system updates.

4. Complaint Management:

- Users can register and track the status of complaints.
- Admins can manage and respond to user complaints.

5. Waste Segregation Guide:

- The system provides users with detailed waste segregation guides for specific items.
- Users can search for and view information on the category of a particular waste item.

6. Schedule Management:

- Admins can create and manage waste collection schedules.
- Users can view collection schedules for their locality.

7. IoT Data Processing:

- IoT devices collect data on waste bin levels and transmit it to AWS IoT Core.
- The system processes this data and updates the Waste Bin Service accordingly.

5. Non-Functional Requirements

WasteSense must also meet non-functional requirements that ensure reliability, usability, and performance:

1. Performance:

- The system should respond to user requests within a defined time frame.
- The IoT data processing pipeline should handle data in near real-time for prompt status updates.

2. Scalability:

- The architecture must support an increasing number of users, bins, and data points without a drop in performance.
- The microservice-based architecture and load balancers enable horizontal scaling as needed.

3. Security:

- Strong data encryption and secure access protocols should be maintained at all levels.
- The JWT Identity Service must ensure that only authenticated and authorized users can access specific features and data.

4. Availability:

- The system should have high availability, ideally 99.9% uptime, with backup strategies and redundancy.
- It should be deployed in cloud environments that support failover and disaster recovery.

5. Usability:

- The user interface should be intuitive and easy to navigate, allowing users to access core features with minimal training.
- User feedback mechanisms should be in place to continually improve user experience.

6. Maintainability:

- Code should be modular, well-documented, and follow industry standards for ease of maintenance and updates.
- The system should use version control to manage code changes and deployments.

7. Reliability:

- The system must reliably process and store data, ensuring that updates and transactions are consistent and fault-tolerant.
- Regular testing should be performed to identify and fix potential issues before they affect users.

6. System Architecture

The architecture of WasteSense follows a microservices approach, ensuring modularity, scalability, and ease of deployment. The primary components include:

- **Client Interface**: Front-end web application for users and administrators to interact with the system.
- **API Gateway**: Acts as a single-entry point for all client requests, managing routing and load balancing between microservices.
- **JWT Identity Service**: Handles user authentication and authorization, issuing and validating JWT tokens.
- **Microservices**:
    - **User Service**: Manages user registration, authentication, and user details.
    - **Waste Bin Service**: Monitors waste bin statuses and locations.
    - **Notification Service**: Sends alerts for bin statuses and weather conditions.
    - **Complaint Management Service**: Allows users to register and track complaints.
    - **Collection Schedule Service**: Manages and optimizes the collection schedule.
    - **Data Ingestion Service**: Processes real-time data from IoT devices.
- **IoT Core**: Collects real-time data from IoT sensors installed in waste bins and routes it to the data ingestion service.
- **Service Registry**: Ensures all services can discover and communicate with each other.
- **Database Layer**: Stores all user, bin, complaint, schedule, and notification data securely.

7. Technologies Used

- **Programming Languages**:
  - **Java**: For building the back-end microservices using Spring Boot.
  - **TypeScript**: For the front-end development (web application).
- **Frameworks**:
  - **Spring Boot**: Provides the foundation for building the microservices.
  - **Angular**: Used for building the client-facing user interface.
- **Cloud and IoT**:
  - **AWS IoT Core**: Manages real-time data collection from IoT devices.
  - **Amazon S3**: Used for static file storage.
  - **Amazon EC2**: Hosts the deployed microservices.
- **Messaging and Communication**:
  - **Kafka**: Used for handling event-based communication between services.
- **Security**:
  - **JWT (JSON Web Token)**: Secures user authentication and authorization.
  - **Spring Security**: Provides security configuration and management.
- **Databases**:
  - **MongoDB**: Serves as the main database for user data, bin data, complaints, schedules, and notifications.
- **API Management**:
  - **Spring Cloud Gateway**: Handles routing, load balancing, and API gateway functions.
- **Service Discovery**:
  - **Eureka**: Registers and discovers microservices to enable smooth inter-service communication.

8. Features and Use Cases

WasteSense provides an extensive set of features tailored for both users and administrators:

Features:

- **User Registration and Login**: Secure user authentication with JWT tokens.
- **Bin Monitoring**: Real-time updates on the status of waste bins.
- **Notifications**: Automatic alerts for bin fill levels and weather warnings.
- **Complaint Management**: Users can submit complaints and track their statuses.
- **Collection Scheduling**: Administrators can plan and modify collection schedules.
- **Waste Segregation Guide**: Provides users with information on waste categorization.

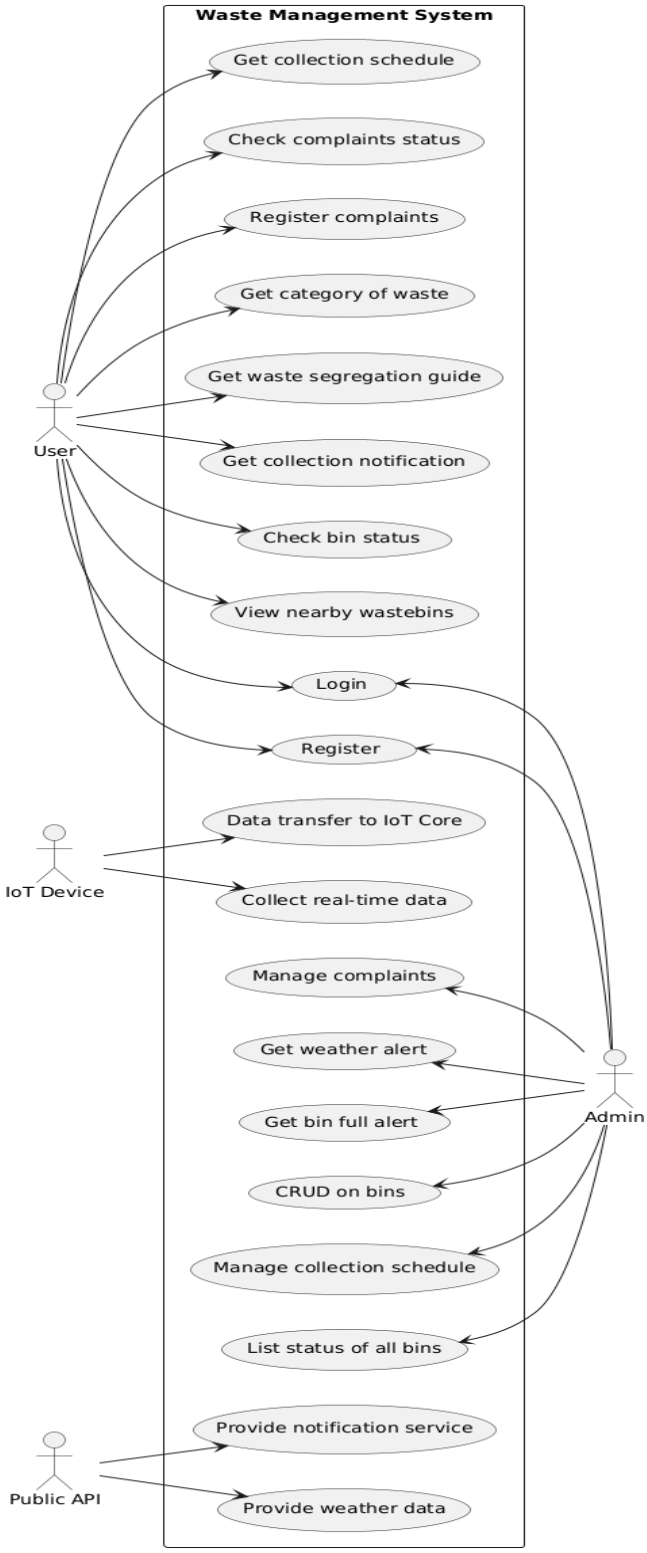
- **Weather Integration**: Incorporates weather data to inform collection operations.

Use Cases:

- **User**: Register, log in, view nearby bins, check bin status, receive notifications, submit complaints, view collection schedules, and access the waste segregation guide.
- **Admin**: Register, log in, manage bin statuses, update collection schedules, handle alerts, process complaints.
- **IoT Device**: Collects and sends real-time data to the IoT Core.
- **Public API**: Provides weather data and notification services to integrate with the system.

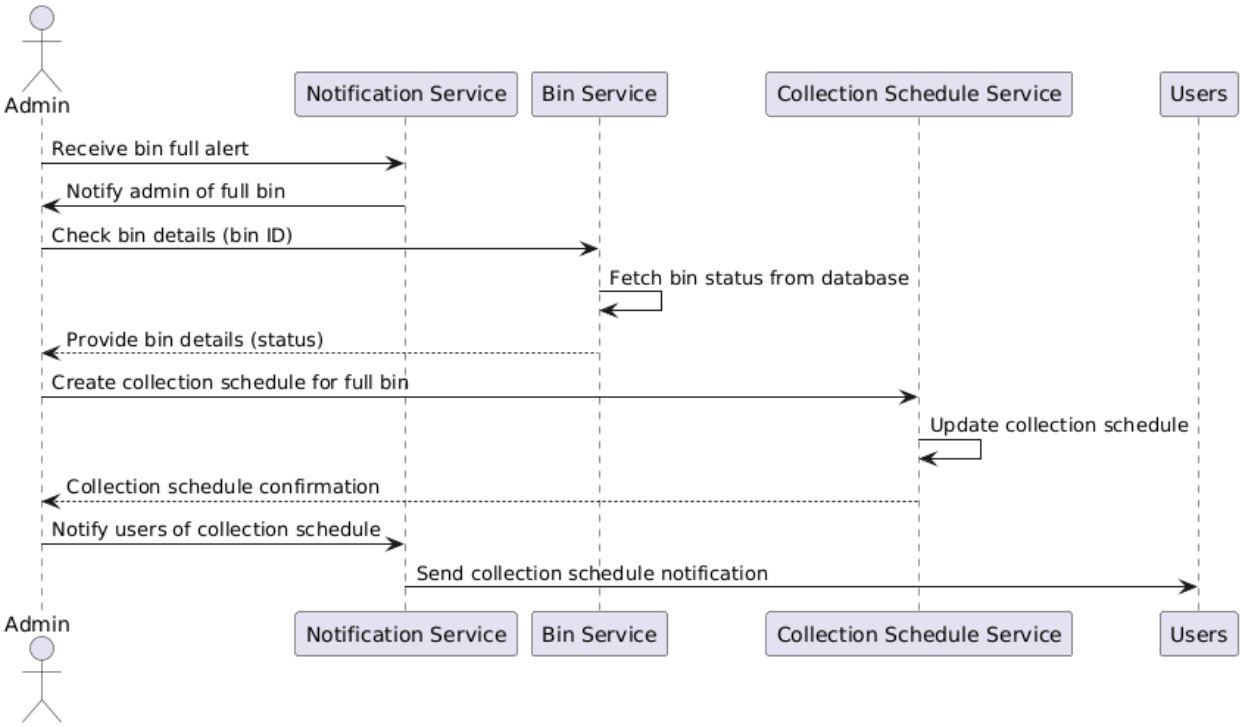
9. Design And Implementation

1. Use-Case Diagram:

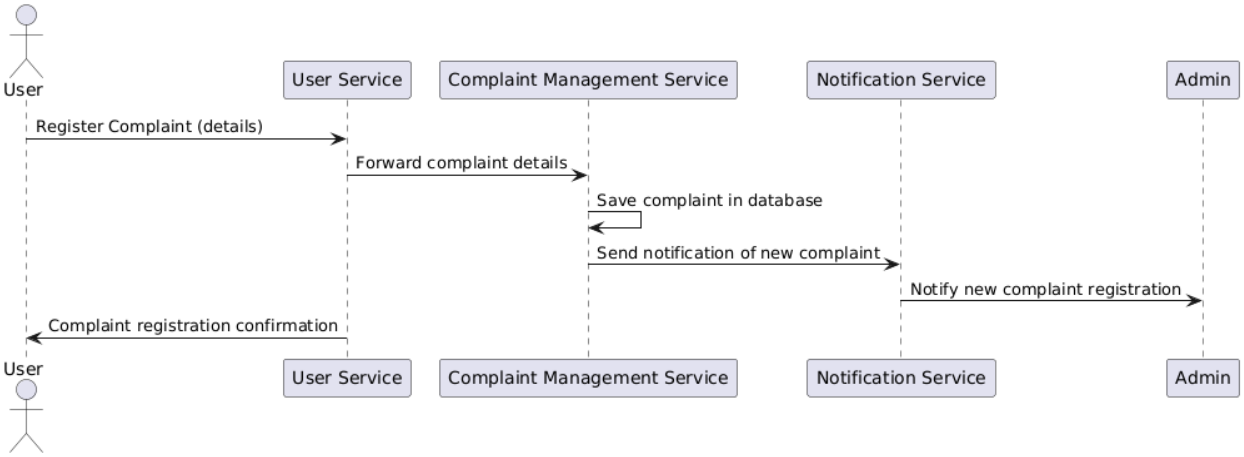


2. Sequence Diagrams:

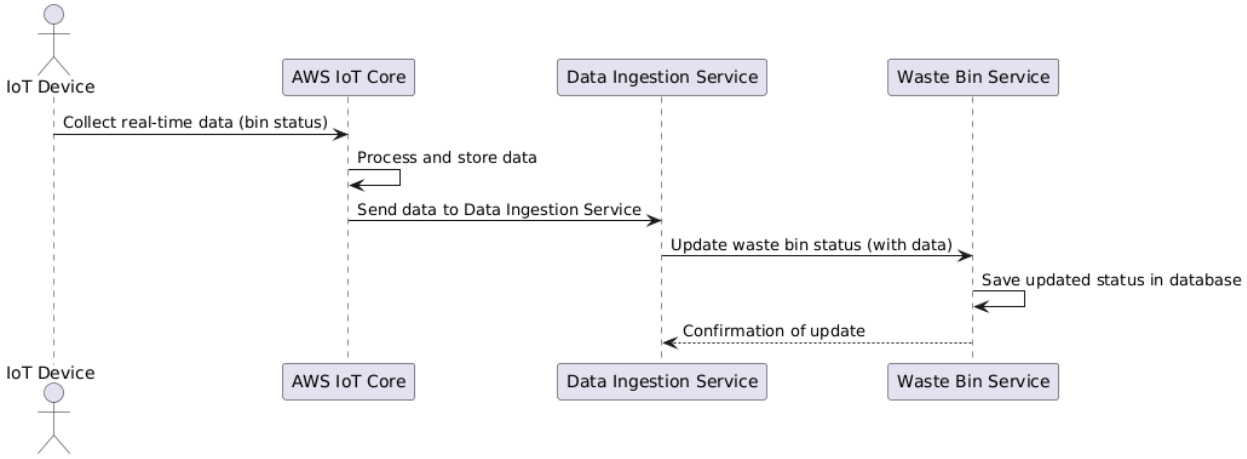
- Bin-Full Notification Sequence:



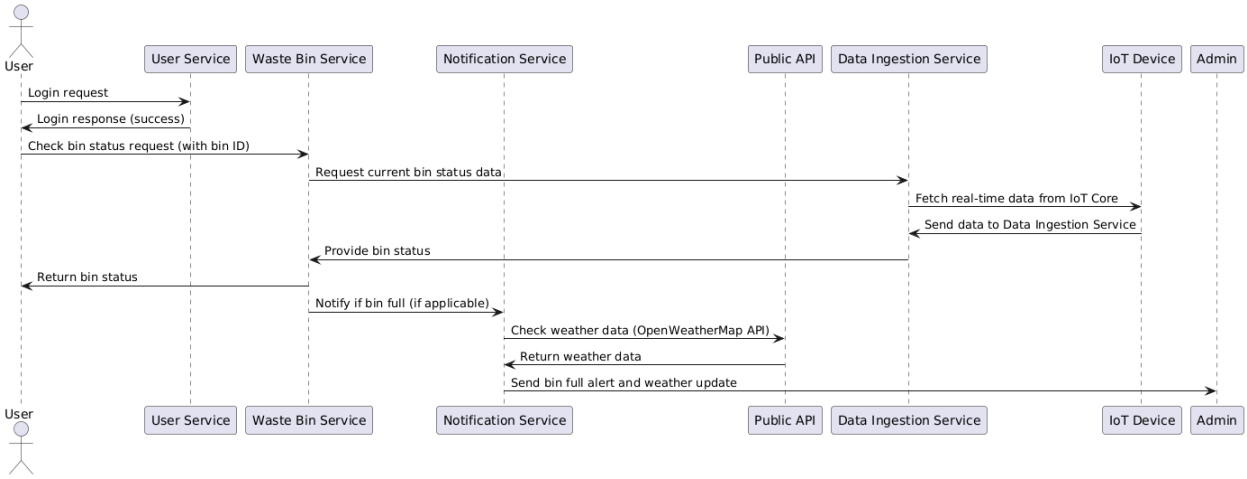
- Complaint Sequence:



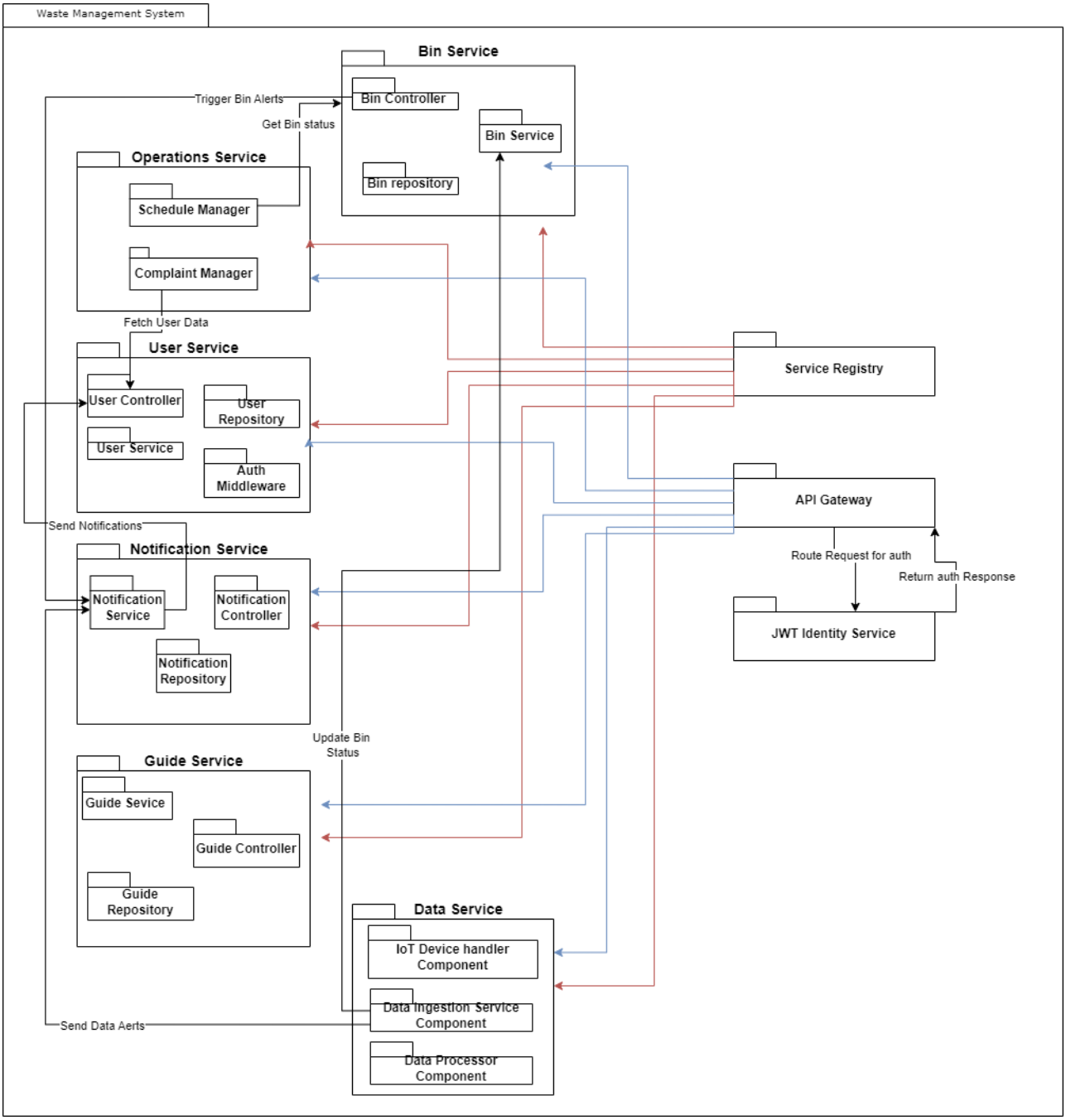
- Getting Data from IoT Sequence:



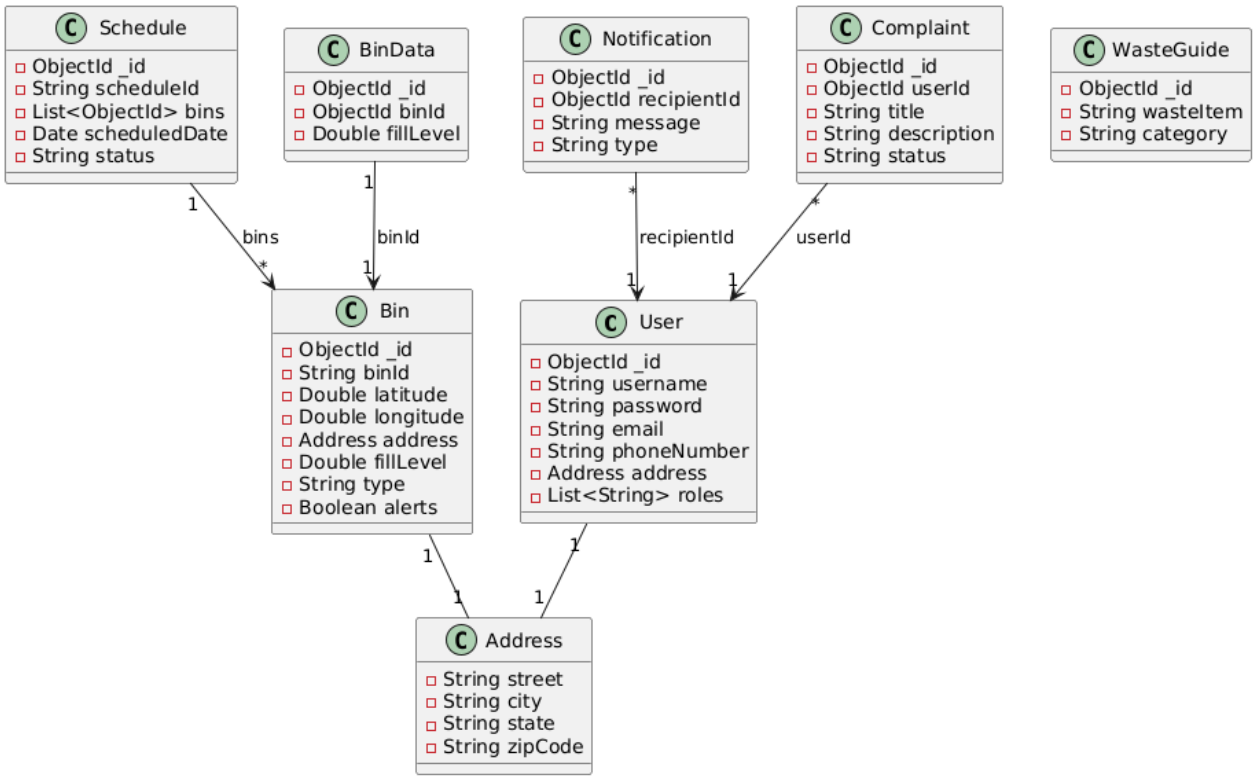
- Request Bin Status Sequence:



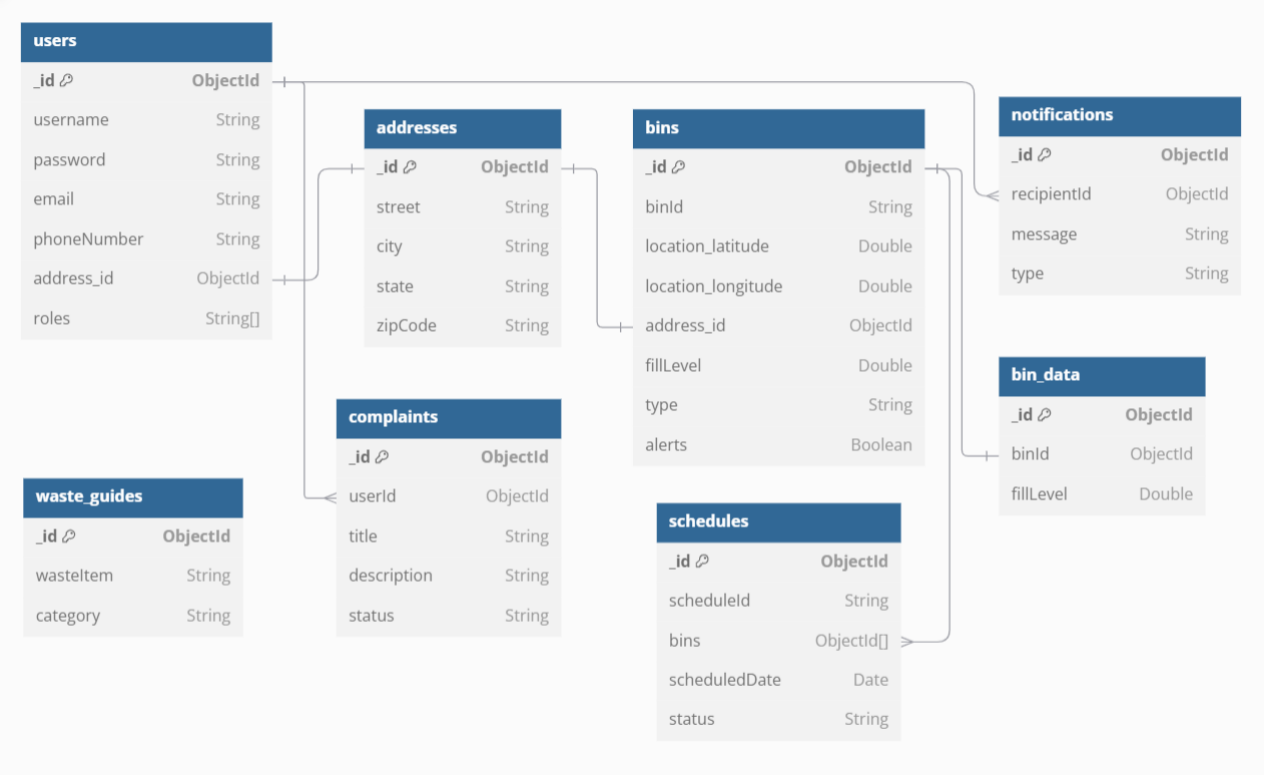
3. Package Diagram:



4. Class Diagram:


Schedule
ObjectId _id
String scheduleId
List<ObjectId> bins
Date scheduledDate
String status
BinData
ObjectId _id
ObjectId binId
Double fillLevel
Notification
ObjectId _id
ObjectId recipientId
String message
String type
Complaint
ObjectId _id
ObjectId userId
String title
String description
String status
WasteGuide
ObjectId _id
String wasteItem
String category
1
bins
*
1
binId
1
*
recipientId
1
*
userId
1
Bin
ObjectId _id
String binId
Double latitude
Double longitude
Address address
Double fillLevel
String type
Boolean alerts
User
ObjectId _id
String username
String password
String email
String phoneNumber
Address address
List<String> roles
1
1
1
1
Address
String street
String city
String state
String zipCode

5. ER Diagram:



10. Modules Description

WasteSense is divided into several well-defined modules, each responsible for handling specific functionality:

1. User Service:

- Handles user registration, authentication, profile management, and role assignment.
- Manages user data storage and retrieval with MongoDB.
- Integrates with the JWT Identity Service for authentication.

2. Waste Bin Service:

- Monitors waste bins' status, including real-time updates from IoT devices.
- Provides CRUD operations for bin details (admin only).
- Triggers alerts when bins are full or near full.

3. Notification Service:

- Sends alerts and notifications to users and admins regarding bin status, weather updates, and collection schedules.
- Integrates with external APIs for push notifications or email services.

4. Complaint Management Service:

- Allows users to file complaints regarding waste management issues.
- Enables admins to manage and respond to complaints.
- Tracks the status of complaints from submission to resolution.

5. Collection Schedule Service:

- Facilitates scheduling for waste collection.
- Provides a comprehensive view of planned and completed schedules.
- Integrates with the Waste Bin Service to check the bin fill levels before scheduling.

6. IoT Data Ingestion Service:

- Collects real-time bin status data from IoT devices via AWS IoT Core.
- Updates the Waste Bin Service with current fill levels and alerts.

7. API Gateway:

- Acts as a single entry-point for client requests.
- Routes traffic to respective microservices while ensuring secure access through JWT verification.

8. Service Registry (Eureka):

- Enables the discovery of microservices for internal communication.
- Ensures high availability and load balancing across services.

9. JWT Identity Service:

- Manages user authentication and token generation for secure access.
- Works with the User Service to validate credentials and issue tokens.

11. API Documentation

WasteSense includes a detailed API documentation to facilitate integration and usage:

1. User Service APIs:

- **POST /users/register**: Registers a new user.
- **POST /users/login**: Authenticates user and provides JWT token.
- **GET /users/profile**: Retrieves user profile details (secured).

2. Waste Bin Service APIs:

- **GET /bins**: Lists all bins with their current status.
- **POST /bins**: Creates a new bin (admin only).
- **PUT /bins/{binId}**: Updates bin details (admin only).
- **GET /bins/{binId}/status**: Checks the status of a specific bin by ID.
- **GET /bins/all-status**: Lists all bins with their status (admin only).
- **GET /bins/urgent**: Lists bins requiring immediate action (admin only).
- **DELETE /bins/{binId}**: Deletes a specific bin (admin only).

3. Notification Service APIs:

- **GET /notifications/user/{userId}**: Retrieves notifications for a user.
- **POST /notifications**: Sends a new notification to specified users.

4. Complaint Management APIs:

- **POST /complaints**: Submits a new complaint.
- **GET /complaints**: Gets all pending complaints(admin) only.
- **GET /complaints/{userId}**: Retrieves all complaints made by a user(admin only).
- **PUT /complaints/{complaintId}/status**: Updates the status of a complaint (admin only).

5. Collection Schedule APIs:

- **GET /schedules**: Lists all collection schedules.
- **POST /schedules**: Creates a new collection schedule (admin only).

6. IoT Data Ingestion Service APIs:

- **POST /bin-data**: Receives and stores real-time bin data from IoT devices.
- **GET /bin-data/{binId}**: Fetches the latest data for a specific bin.

7. JWT Identity Service APIs:

- **POST /auth/validate**: Validates JWT token.
- **POST /auth/issue**: Issues a new JWT token after user authentication.

12. Data Flow and Communication

WasteSense ensures seamless data flow and communication between microservices using REST APIs, backed by the Service Registry for service discovery. The typical communication flow includes:

1. Client to API Gateway:

- All client requests are routed through the API Gateway, which acts as a mediator and ensures that requests are authenticated via the JWT Identity Service.

2. API Gateway to Microservices:

- Once authentication is confirmed, the API Gateway forwards requests to the appropriate microservice, such as User Service, Waste Bin Service, or Complaint Management Service.

3. Internal Communication:

- The Service Registry (Eureka) facilitates microservices communication, allowing them to locate and interact with each other efficiently.
- Example: When the Complaint Management Service needs to fetch user details, it communicates with the User Service via service discovery.

4. IoT Data Flow:

- IoT devices collect real-time data on waste bin levels and send this data to AWS IoT Core.
- The IoT Data Ingestion Service pulls this data, processes it, and updates the Waste Bin Service.
- The Waste Bin Service then notifies the Notification Service if an alert threshold is met, prompting a user or admin notification.