B.M.S. COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

OOMD Mini Project Report

SMART WASTE MANAGEMENT

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

Harsh Ghiya (1BM21CS073)
Harshitha R (1BM21CS075)
Jigar D Patel (1BM21CS081)

Faculty In Charge

Sheetal V A
Assistant Professor

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2023-2024

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, HARSH GHIYA (1BM21CS073), HARSHITHA R (1BM21CS075) and JIGAR D PATEL (1BM21CS081) students of 6th Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this OOMD Mini Project entitled "SMART WASTE MANAGEMENT" has been carried out in Department of CSE, B.M.S. College of Engineering, Bangalore during the academic semester March - July 2024. I also declare that to the best of our knowledge and belief, the OOMD mini Project report is not from part of any other report by any other students.

Signature of the Candidate

HARSH GHIYA (1BM21CS073)

HARSHITHA R (1BM21CS075)

JIGAR D PATEL (1BM21CS081)

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOMD Mini Project titled "SMART WASTE MANAGEMENT" has been carried out by HARSH GHIYA (1BM21CS073), HARSHITHA R (1BM21CS075) and JIGAR D PATEL (1BM21CS081) during the academic year 2023-2024.

Signature of the Faculty in Charge

TABLE OF CONTENTS

Chapter 1: Problem Statement

Introduction:

Waste management is a crucial aspect of urban planning and environmental conservation. The traditional methods of waste collection are often inefficient, leading to various challenges such as overfilled bins, unnecessary fuel consumption, increased operational costs, and adverse environmental impacts. To address these issues, integrating modern technology into waste management processes has become essential.

Current Challenges:

- **Inefficiency in Waste Collection**: Traditional waste collection systems follow predetermined routes and schedules, which often results in either underutilization or overflow of waste bins. This inefficiency leads to unnecessary trips and missed collections.
- **High Operational Costs**: The costs associated with waste collection, including fuel, labor, and vehicle maintenance, are significant. Inefficient routing further escalates these expenses.
- **Environmental Impact**: Frequent trips by waste collection trucks contribute to air pollution and carbon emissions. Inefficient waste collection practices also lead to littering and unsanitary conditions.
- **Lack of Real-Time Monitoring**: There is often no real-time data available on the status of waste bins, making it challenging to manage the collection process effectively.

Proposed Solution:

To overcome these challenges, we propose the implementation of a smart waste management system. This system leverages Internet of Things (IoT) technology to digitalize and automate waste collection processes. Key components of the solution include:

1. **Smart Bins**: Equipping waste bins with sensors to monitor the level of fullness and their location.
2. **Tracking Devices**: Installing tracking devices on collection trucks to monitor their movement and optimize routes.
3. **Smart Dashboard**: Developing a central dashboard to visualize real-time data from sensors and tracking devices.
4. **Optimization Application**: Creating an application to control and optimize the path of collection trucks, ensuring timely and efficient waste collection.

By adopting this solution, we can achieve a more efficient, cost-effective, and environmentally friendly waste management system.

Chapter 2: Software Requirement Specification

Introduction

1.1 Purpose of this Document:

This document outlines the Software Requirement Specification (SRS) for a Smart Waste Management System. The purpose is to define the functional and non-functional requirements, design constraints, and overall scope of the project. This document serves as a guide for developers, stakeholders, and users to ensure all parties have a clear understanding of the system's capabilities and limitations.

1.2 Scope of this Document:

The scope includes the overall working and main objectives of the Smart Waste Management System. This document details the value it will provide to customers, including cost and time required for development. It aims to improve efficiency in waste collection, reduce operational costs, and enhance environmental sustainability.

1.3 Overview:

The Smart Waste Management System will utilize IoT technology to digitalize waste collection processes. It includes smart bins with sensors, tracking devices on collection trucks, and a centralized dashboard for real-time monitoring and optimization. This system aims to provide a more efficient, cost-effective, and environmentally friendly waste management solution.

2. General Description

2.1 Product Functions:

- Real-time monitoring of waste bin levels and locations.
- Optimization of waste collection routes.
- Centralized dashboard for data visualization.
- Application control for path optimization and efficient waste collection.

2.2 User Characteristics:

- Admins: Responsible for system management and data manipulation.
- Operators: Handle waste collection operations based on optimized routes.
- General Users: Community members who benefit from improved waste collection services.

2.3 Features and Benefits:

- Cost reduction through optimized routes and schedules.
- Improved efficiency with real-time monitoring.
- Enhanced environmental sustainability by reducing emissions.
- Better service quality for the community.

3. Functional Requirements

3.1 Data Storage and Management:

- The system must store required information and provide admin privileges for data manipulation.
- The system should generate and update rules for optimal operation.

3.2 Data Processing and Calculation:

- Calculate optimal routes for waste collection.
- Process real-time data from sensors and tracking devices.

3.3 System Instructions:

Provide instructions for waste collection based on data analysis.

4. Interface Requirements

4.1 User Interface:

- The application should have a user-friendly interface with great ergonomics.
- Admin dashboard for data visualization and system management.

4.2 Software Interfaces:

- Communication protocols for sensors, tracking devices, and central dashboard.
- Integration with existing waste management software, if any.

5. Performance Requirements

5.1 System Performance:

- Real-time data processing capability.
- Minimal response time for route optimization.
- Efficient memory usage and low error rates.

5.2 Resource Utilization:

- Optimal use of network bandwidth for data transmission.
- Efficient use of processing power and storage.

6. Design Constraints

6.1 Hardware Constraints:

- Compatibility with existing waste bins and collection trucks.
- Sensor and tracking device integration limitations.

6.2 Software Constraints:

- Use of specific algorithms for route optimization.
- Limitations imposed by the underlying operating system and network infrastructure.

7. Non-Functional Attributes

7.1 Security:

- Ensure data security and privacy.
- Implement access controls and encryption.

7.2 Portability: Compatibility with various hardware and software environments.

7.3 Reliability: High availability and fault tolerance.

7.4 Reusability: Modular design for easy updates and enhancements.

7.5 Application Compatibility: Seamless integration with other relevant applications.

7.6 Data Integrity: Ensure accuracy and consistency of data.

8. Preliminary Schedule and Budget

8.1 Schedule:

Initial development phase: 3 months

Testing phase: 2 months

Deployment and training phase: 1 month

8.2 Budget:

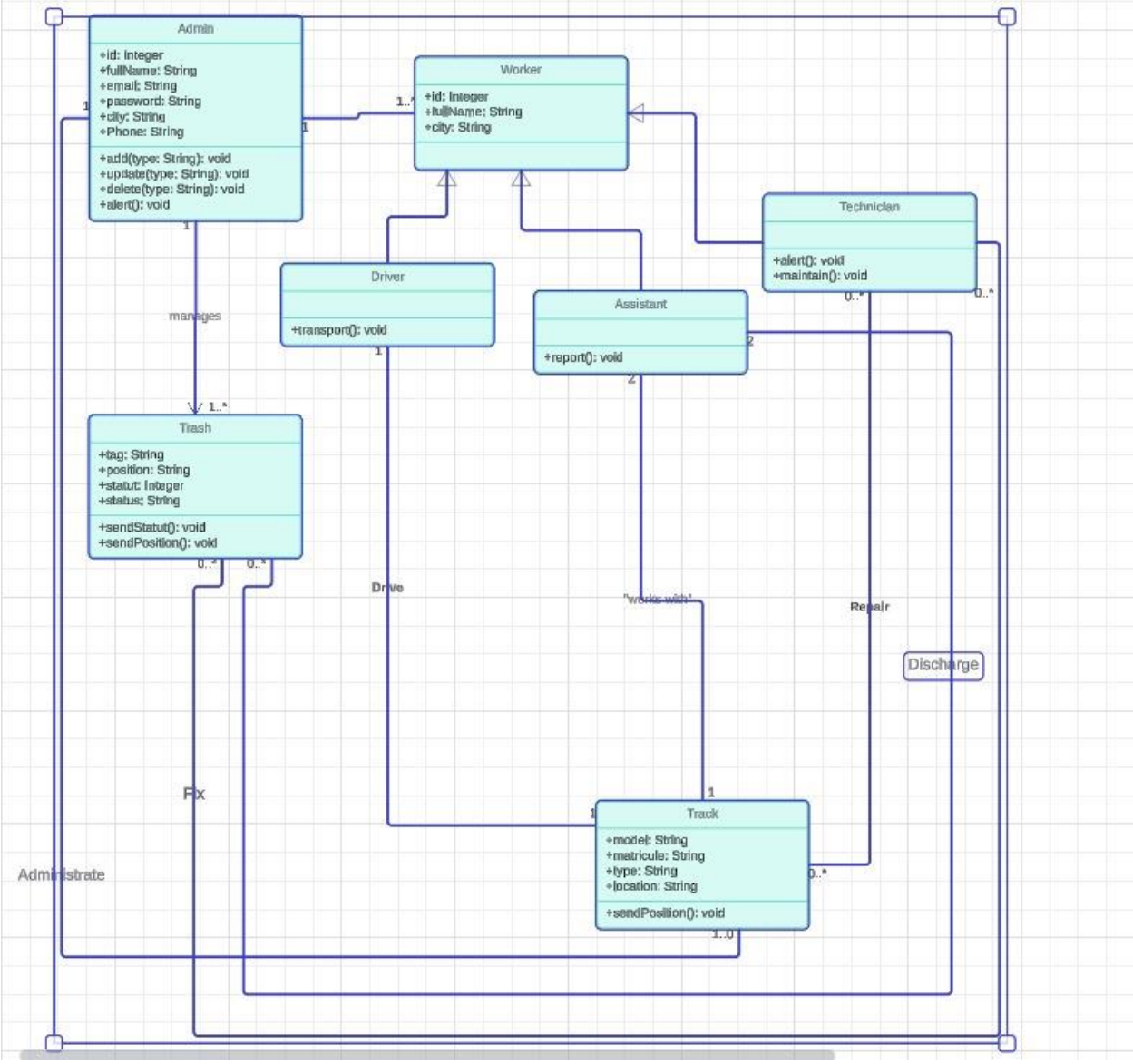
Estimated total cost: $100,000

Development: $60,000

Testing: $20,000

Deployment and training: $20,000

Chapter 3: Class Modeling



1. Admin Class

Attributes:

- id: Unique identifier for each admin.
- fullName: Name of the admin.
- email: Email address for communication.
- password: Secure password for authentication.
- city: City where the admin is operating.
- phone: Contact number for the admin.

Methods:

- add(type: String): Method to add new entries to the system (e.g., workers, bins).
- update(type: String): Method to update existing entries.
- delete(type: String): Method to delete entries.
- alert(): Method to send alerts or notifications.

Relevance: The admin class represents the system administrators who manage the overall operations. They have the highest level of privileges, allowing them to add, update, or delete records and send alerts.

2. Worker Class

Attributes:

- id: Unique identifier for each worker.
- fullName: Name of the worker.
- city: City where the worker operates.

Relevance: This is a generalized class for all types of workers involved in the system. Specific worker types (e.g., technicians, drivers, assistants) inherit from this class.

3. Technician Class

Methods:

- alert(): Method to alert the system or admin about issues.
- maintain(): Method to perform maintenance tasks on equipment or vehicles.

Relevance: Technicians are responsible for maintaining and repairing the equipment, such as smart bins and tracking devices. They ensure that the system components are functional and address any technical issues.

4. Driver Class

Methods:

- transport(): Method for handling the transportation of waste.

Relevance: Drivers operate the collection trucks. They follow optimized routes generated by the system to collect waste efficiently.

5. Assistant Class

Methods:

- report(): Method for creating and submitting reports on operations.

Relevance: Assistants support the drivers and technicians by helping with various tasks and providing reports on the system's performance, issues, and other relevant data.

6. Track Class

Attributes:

- model: Model of the waste collection truck.
- matricule: License plate or identifier for the truck.
- type: Type of truck (e.g., regular, compacting).
- location: Current location of the truck.

Methods:

- sendPosition(): Method to send the current position of the truck.

Relevance: This class represents the collection trucks used in the system. It includes attributes and methods necessary for tracking the trucks' movements and ensuring they follow the optimized routes.

7. Trash Class

Attributes:

- tag: Unique identifier for each smart bin.
- position: Geographical position of the bin.
- statut (should be status): Current fill level of the bin.
- status: Current status of the bin (e.g., full, empty, malfunctioning).

Methods:

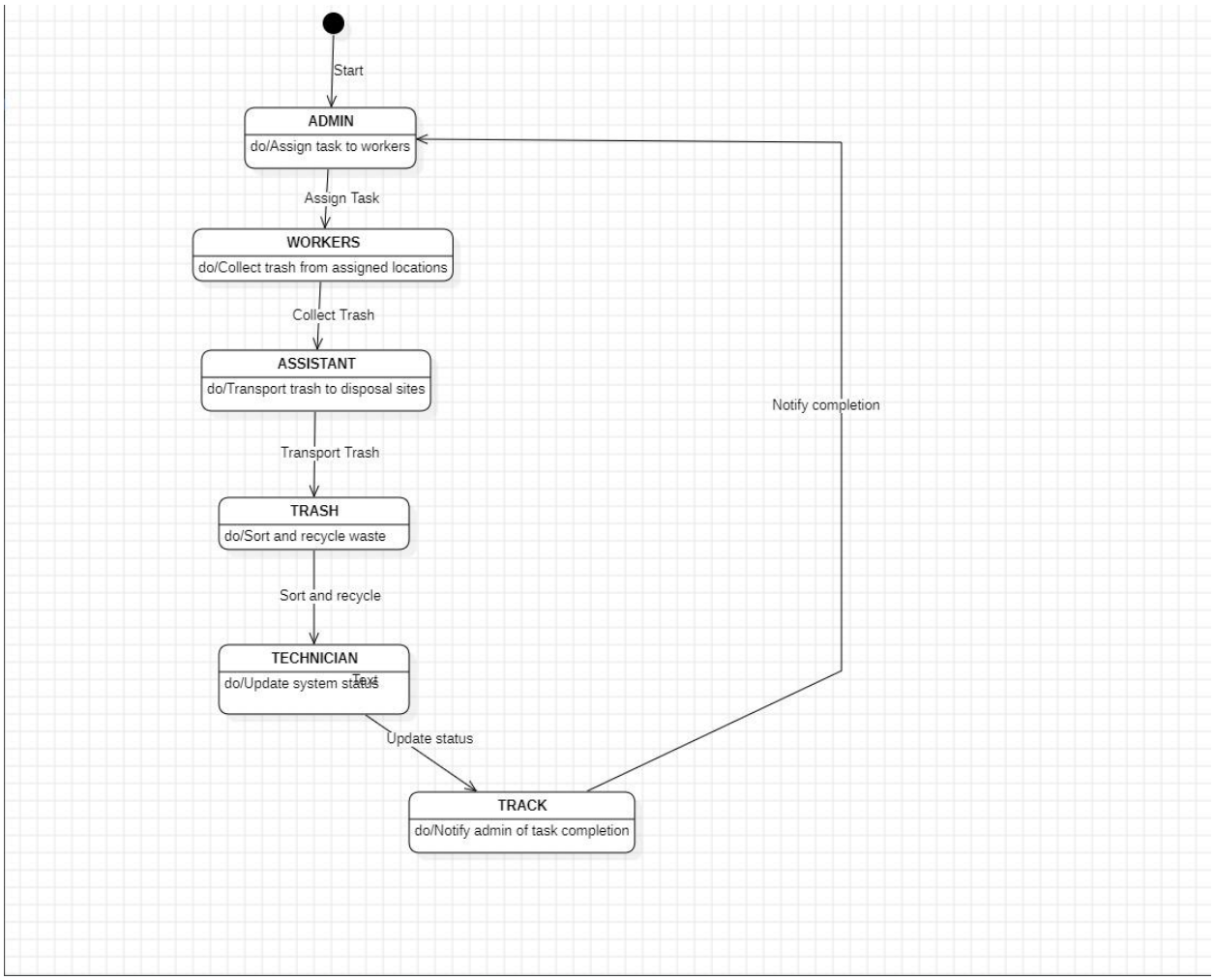
- sendStatut(): Method to send the current fill level.
- sendPosition(): Method to send the current position of the bin.

Relevance: The trash class represents the smart bins equipped with sensors to monitor their fill levels and positions. These bins communicate their status to the system, allowing for efficient waste collection planning.

Relationships:

- **Admin-Manages-Trash**: Admins manage the smart bins, ensuring they are monitored and maintained.
- **Admin-Manipulates-Worker**: Admins can add, update, or remove worker records.
- **Worker-Types (Technician, Driver, Assistant)**: Workers are specialized into technicians, drivers, and assistants, each with specific roles.
- **Technician-Administrates-Track**: Technicians maintain the collection trucks.
- **Driver-Drives-Track**: Drivers operate the collection trucks.
- **Assistant-Works-With-Driver**: Assistants support drivers in their operations.
- **Track-Discharges-Trash**: Collection trucks (Track) are responsible for emptying the smart bins (Trash).

Chapter 4: State Modeling



States and Their Relevance

1. **Admin**
    - **State: Assign tasks to workers**
        - **Relevance:** This state represents the admin's role in assigning specific tasks to various workers, such as drivers, technicians, and assistants, ensuring that each task is managed efficiently.
2. **Worker**
    - **State: Collect trash from assigned locations**
        - **Relevance:** In this state, workers are responsible for visiting assigned locations to collect trash. This is a critical part of the waste management process, as it ensures that waste is gathered efficiently from different areas.
3. **Assistant**
    - **State: Transport trash to disposal sites**
        - **Relevance:** Assistants play a vital role in moving the collected trash to designated disposal or recycling sites. This state highlights the logistics and transportation aspect of the waste management system.

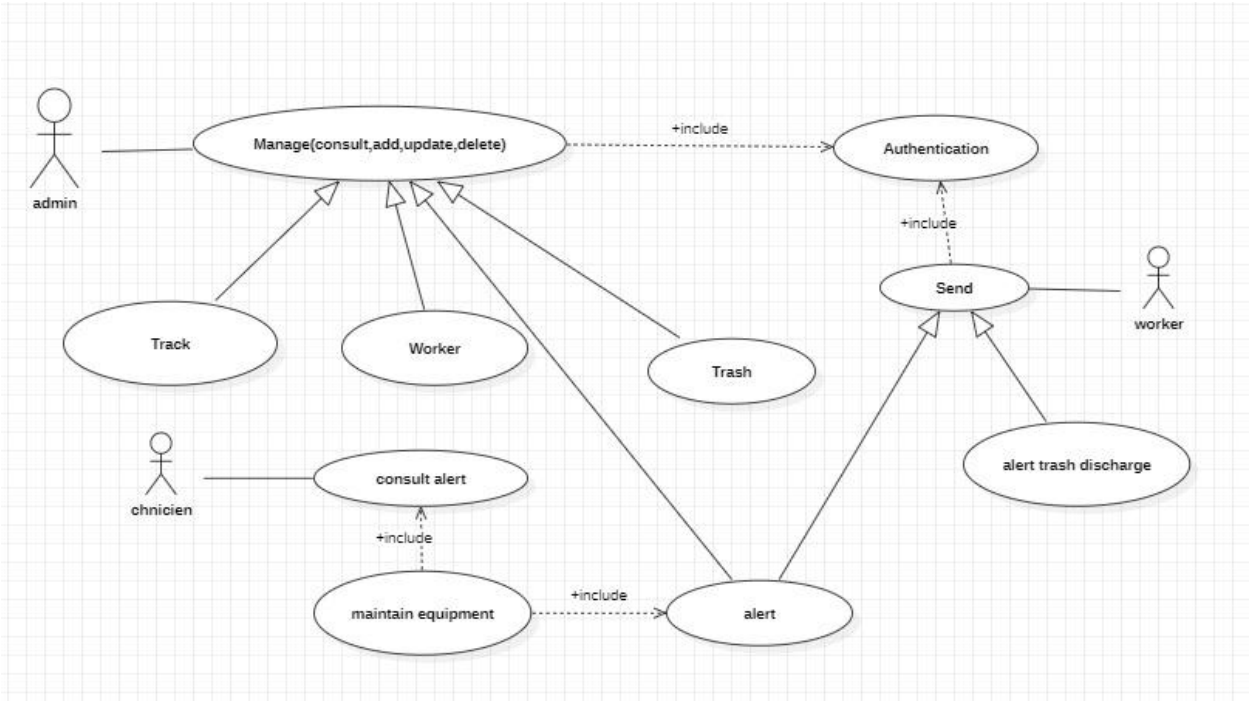
4. **Trash**
    - **State: Sort and recycle waste**
        - **Relevance:** This state represents the smart bins' ability to categorize and recycle waste. Sorting waste helps in proper disposal and recycling, promoting environmental sustainability.
5. **Technician**
    - **State: Update system status**
        - **Relevance:** Technicians ensure that the system is up-to-date by maintaining and repairing equipment and updating the system status. This state is crucial for the smooth functioning of the overall waste management system.
6. **Track**
    - **State: Notify admin of task completion**
        - **Relevance:** The track represents the collection trucks, and this state involves notifying the admin once a task, such as collecting trash from a location, is completed. This state ensures that the admin is informed about the progress and status of tasks.

Events and Their Relevance

1. **Assign task**
    - **Relevance:** This event triggers the transition of workers from an idle state to actively collecting trash. It represents the admin's directive to start a specific task.
2. **Collect trash**
    - **Relevance:** This event indicates that the worker has started or is in the process of collecting trash from the assigned locations. It’s an action that changes the state of the worker to active engagement in their duty.
3. **Transport trash**
    - **Relevance:** This event signifies the movement of collected trash by assistants to the disposal sites. It changes the state of the trash from being in the collection phase to being transported for disposal.
4. **Sort and recycle**
    - **Relevance:** This event is related to the trash state, where the collected waste is sorted and recycled. It represents the process of handling waste after it has been transported to disposal sites.
5. **Update status**
    - **Relevance:** This event represents the action taken by technicians to update the system status. It ensures that the current state of the system and its components is accurately reflected and maintained.
6. **Notify completion**
    - **Relevance:** This event signifies the completion of a task and the subsequent notification to the admin. It marks the transition of the track state to having completed its assigned duty and updating the admin about the same.

Chapter 5: Interaction Modeling

USE CASE DIAGRAM



Actors:

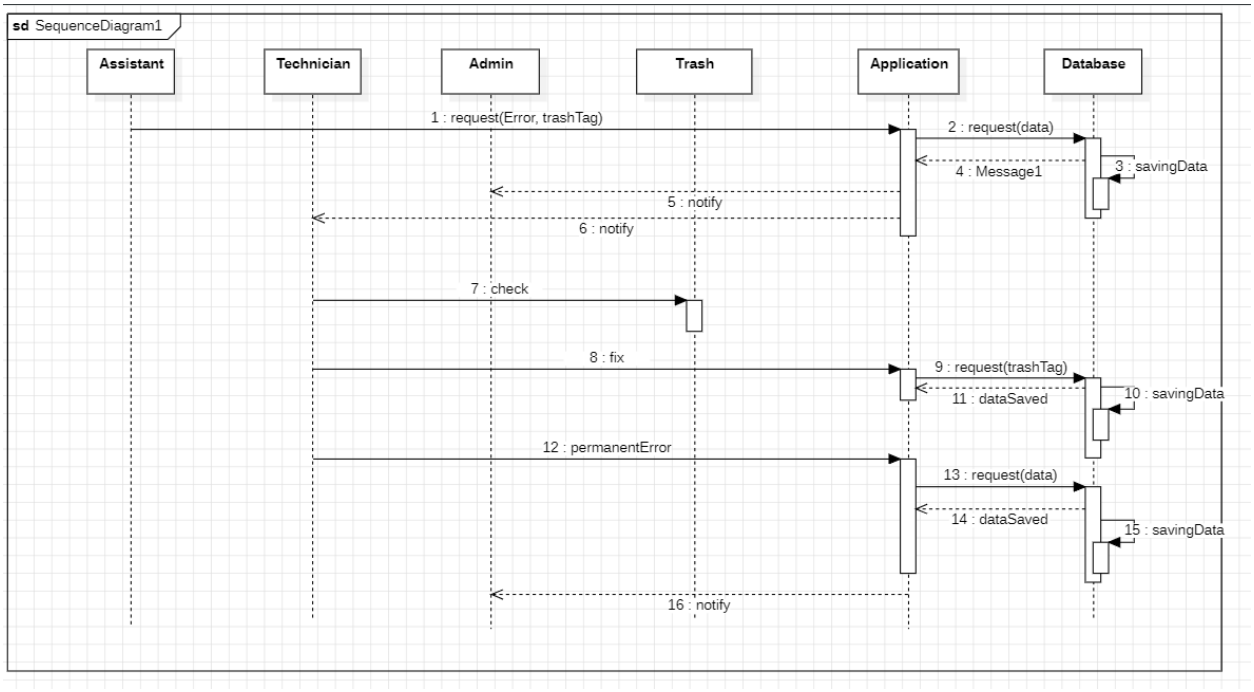
1. **Admin:** The Admin is a primary actor who has the ability to manage various aspects of the system, such as consulting, adding, updating, and deleting records. This role is crucial for the overall maintenance and administration of the system.
2. **Worker:** The Worker is an actor who interacts with the system primarily to send information. This role is likely involved in the operational aspects and execution of tasks within the system.
3. **Technician:** The Technician is responsible for consulting alerts and maintaining equipment. This role is critical for ensuring the technical aspects and maintenance needs of the system are met, thereby supporting operational continuity and safety.

Use Cases:

1. **Manage (consult, add, update, delete):** Use case involves managing various records in the system. The Admin can perform CRUD (Create, Read, Update, Delete) operations on different entities such as tracks, workers, and trash. This is fundamental for keeping the system's data accurate and up-to-date.
2. **Track:** This use case allows the Admin to track various elements or activities within the system. Tracking could involve monitoring performance, usage, or any other relevant metrics.

3. **Worker:** This is likely related to managing worker-related data or activities. It could involve scheduling, assignments, or other worker management tasks.
4. **Trash:** This use case involves managing trash-related data. This might include monitoring trash levels, collection schedules, and ensuring proper disposal processes are followed.
5. **Send:** The Worker uses this use case to send information, which might include alerts, status updates, or other operational data. This is essential for communication and coordination within the system.
6. **Alert:** Alerts are critical for notifying relevant parties about important events or issues. This use case ensures timely communication of alerts to prevent or address problems swiftly.
7. **Alert Trash Discharge:** This specific alert use case pertains to the discharge of trash. It is important for ensuring that trash discharge is managed properly and any issues related to it are promptly addressed.
8. **Consult Alert:** The Technician uses this use case to consult or review alerts. This is important for diagnosing issues, planning maintenance, or responding to operational alerts.
9. **Maintain Equipment:** This use case involves performing maintenance on equipment. It is crucial for ensuring that all equipment is functioning properly and safely, thereby preventing downtime and ensuring efficient operations.
10. **Authentication:** This use case involves verifying the identity of users (both Admin and Worker) before they can access the system. Authentication is a foundational security measure to protect the system from unauthorized access.

SEQUENCE DIAGRAM



1. **Initiating Request:**
   - Assistant sends a request indicating an error with a trash tag to the **Trash** component (request(error, trashTag)).
   - Trash forwards the request to the Application (1.1: request(data)).
   - Application saves the data to the Database (1.1.1: savingData()).
2. **Notification:**
   - The Database responds to the Application indicating the data has been saved (1.1.2: dataSaved()).
   - The Application notifies the Trash component that the data has been saved (1.2: dataSaved()).
   - The Trash component then notifies both the Technician and the Admin of the error (2: notify to Technician**,** 3: notify to Admin**)**.
3. **Checking and Fixing:**
   - The Technician checks the issue (2: check()) and decides on a fix (3: fix()).
   - The Technician sends a fix request to the Trash component (3.1: request(fixTrashTag)).
   - The Trash component forwards the fix request to the Application (3.1: request(fixTrashTag)).
   - The Application again saves the fix information to the Database (3.1.1: savingData()).
4. **Alternate Flow (Permanent Error):**
   - If a permanent error is detected (alt block), a permanent error is reported (4: permanentError()).
   - The Admin may initiate another data request (4.1: request(data)).
   - The Application saves this new data request to the Database (4.1.1: savingData()).

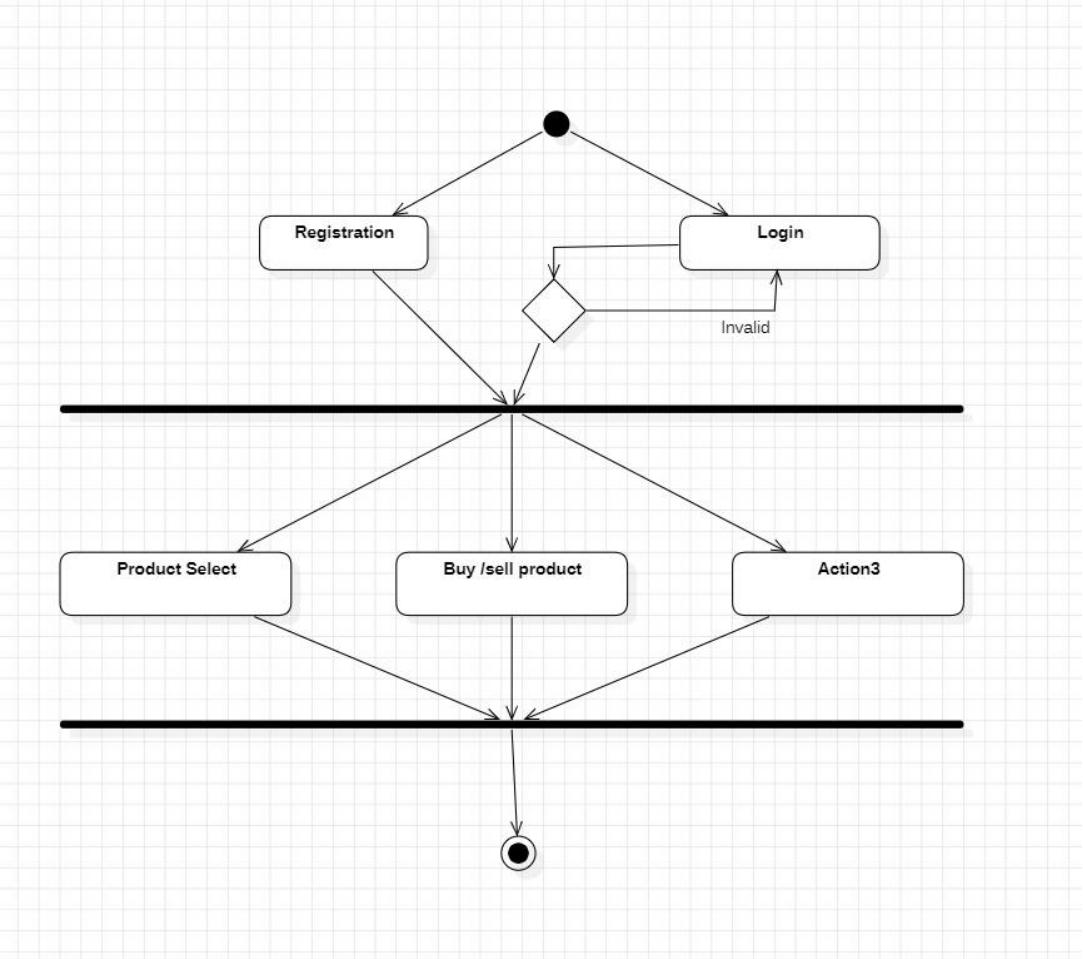
5. **Final Notification:**
    - The Database responds to the Application (4.1.2: dataSaved()).
    - The Application notifies the Trash component of the data save (4.2: dataSaved()).
    - Finally, the Trash component notifies the Admin of the final state (5: notify).

Summary:

- Assistant reports an error related to a trash tag.
- Technician and Admin are notified of the error.
- Technician checks and fixes the error.
- The fix is processed and saved in the system.
- If the error is permanent, further steps are taken by the Admin.
- Notifications are sent at various stages to keep all relevant parties informed.

This sequence diagram effectively models the handling of errors within the system, ensuring appropriate steps are taken to resolve issues and communicate status updates to involved personnel.

ACTIVITY DIAGRAM



- **Start Node**:
    - The process begins at the start node.
- **Registration/Login**:
    - The first activity is either **Registration** or **Login**. The user can choose to register if they are a new user or log in if they already have an account.
- **Decision Node**:
    - After attempting to log in, there is a decision point:
        - If the login is **invalid**, the flow goes back to the **Login** activity, prompting the user to attempt to log in again.
        - If the login is **valid**, the flow continues to the next synchronization bar.

- **Synchronization Bar** (Fork):
    - After successfully logging in or registering, the flow is split into three parallel activities:
        - **Product Select**
        - **Buy/Sell Product**
        - **Action3**
- **Parallel Activities**:
    - The user can perform any or all of the following activities in parallel:
        - **Product Select**: Selecting a product.
        - **Buy/Sell Product**: Buying or selling a product.
        - **Action3**: An unspecified action.
- **Synchronization Bar** (Join):
    - After completing the parallel activities, the flows converge back into a single flow at the synchronization bar.
- **End Node**:
    - The process ends at the end node.