INTERNATIONAL MULTI-AWARD WINNING INSTITUTION FOR SUSTAINABILITY

KULLIYYAH OF INFORMATION & COMMUNICATION TECHNOLOGY

BICS 1304 OBJECT-ORIENTED PROGRAMMING
SEMESTER 2, 2024/2025
SECTION 10

GROUP D

ASSIGNMENT 1
IIUM ACCIDENT DETECTION SYSTEM

PREPARED BY:

NAME	MATRIC NO.	ROLE	PARTICIPATION
SARAH YASMIN BINTI RODZMAN	2413034	PROJECT MANAGER	100%
NUR BALQIS BINTI MOHD KAMARULZAMAN	2410006	FILE HANDLER	100%
PUTRI AIMI BATRISYIA BINTI MUHAMMAD YUSRI	2320206	BACKEND DEVELOPER	100%
NURIN SOFINA BINTI YUSDI	2221372	GUI DESIGNER	100%

LECTURER

DR. DINI OKTARINA DWI HANDAYANI

DUE

2 MAY 2025

Table of Contents

List of Figures

List of Tables

1. Introduction

IIUM Accident Detection and Alert System is a smart safety solution that uses IoT and camera sensors to detect accidents in real-time across the campus. It integrates with an AI engine to analyze sensor data, assess the severity of incidents, and automatically send SMS or call alerts to appropriate emergency responders, including OSEM and IIUM ambulance. The system features a centralized monitoring dashboard for administrators and a user dashboard for authorized staff to review incidents and alerts. The IIUM Safety and Emergency Unit, responders, IT department, system administrators, and the IIUM community are key stakeholders.

2. Problem(s) of the Existing System

The current accident reporting and emergency response system in IIUM is largely manual, relying on eyewitness reports or delayed communication through phone calls. This results in slower response times, especially in remote or less monitored areas of the campus. There is also a lack of centralized data collection and real-time monitoring, which hinders the ability to assess the severity of incidents promptly. Emergency responders may receive incomplete or unclear information, leading to confusion and delays in action. Additionally, the absence of automated alerts and integrated monitoring tools increases the risk of human error and reduces the overall efficiency of campus safety management.

3. Goals of the System

The main goal of the Accident Detection and Alert System for IIUM is to enhance campus safety by enabling real-time accident detection, severity analysis, and rapid emergency response. The system aims to reduce response time through automated alerts and ensure that accurate, timely information reaches the appropriate emergency teams. It also seeks to centralize monitoring activities, improve the coordination of responders, and provide a reliable platform for administrators to oversee and analyze incident trends. Ultimately, the system aspires to minimize the impact of accidents and protect the well-being of the IIUM community.

4. Objectives of the System

The objectives of the Accident Detection and Alert System are to: (1) deploy IoT and camera sensors across IIUM to continuously monitor for accident indicators; (2) implement an AI engine capable of analysing sensor data to detect and evaluate the severity of incidents; (3) automate the alert process through SMS and call notifications to ensure quick and accurate communication with the right emergency responders; (4) develop a centralized dashboard for administrators to track incidents in real-time and review alert history; and (5) enhance coordination between emergency teams by providing them with timely and relevant information for effective decision-making.

5. Use Case Diagram

Figure 1: Use Case Diagram for Accident Detection and Severity Analysis Subsystem by Sarah Yasmin

As shown in Figure 1, it is a use case diagram for the Accident Detection and Severity Analysis Subsystem, which indicates the four main actors who play roles in this subsystem to detect and analyze the severity of an accident. The process begins with accident detection through physical impact data gathered by the Sensor and its subclass IoTSensor, as well as image capture performed by the CameraSensor. These actors are responsible for initiating the data collection process. Once the data is collected, it is transmitted to the AIEngine, which acts as the central processor to analyze and validate whether the incident qualifies as an accident. Upon validation, the AIEngine determines the severity level based on predefined parameters.

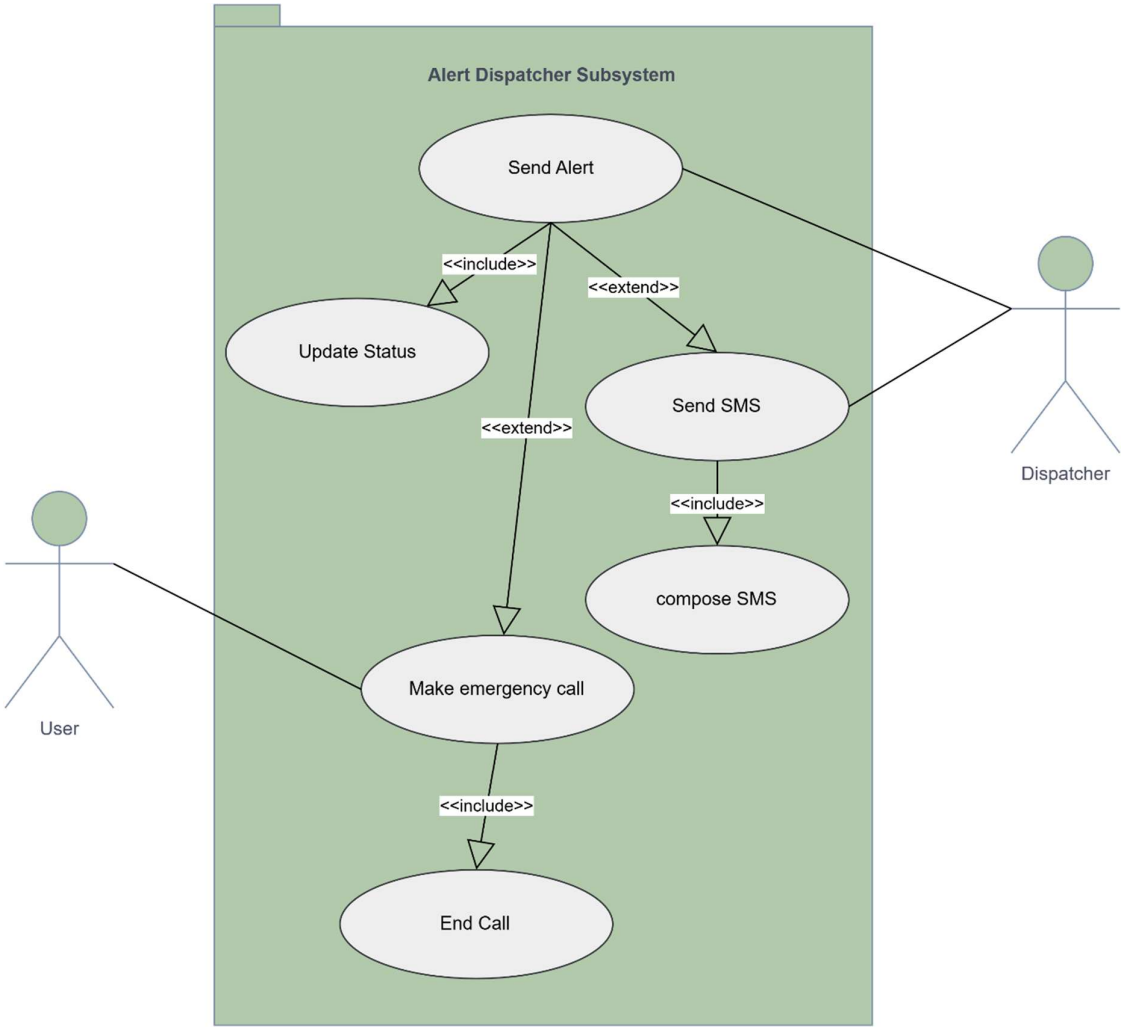

Figure 2: Use Case Diagram for Alert Dispatcher Subsystem by Nur Balqis

As shown in Figure 2, the use case diagram represents the Alert Dispatcher Subsystem, which involves two primary actors: the User and the Dispatcher. The main function is "Send Alert," which can optionally extend to "Send SMS" and "Make Emergency Call," depending on the situation. In emergency situations, the system may initiate a call through "Make Emergency Call".

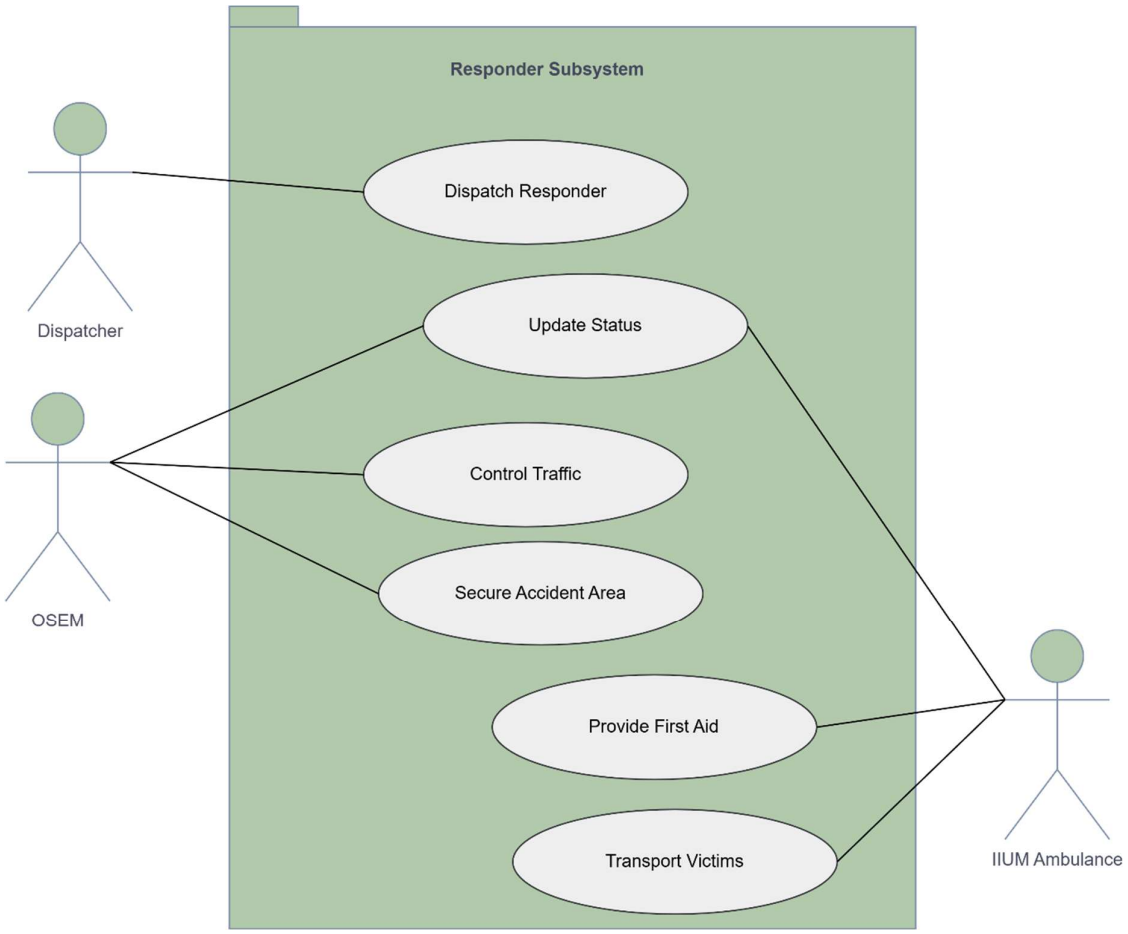

Figure 3: Use Case Diagram for Responder Subsystem by Nurin Sofina

Based on Figure 3, the use case diagram shows how three main actors consists of Dispatcher, OSEM, and IIUM Ambulance interact with Responder Subsystem. It started with the Dispatcher who is responsible for the “Dispatch Responder” use case where it will determine whether to assign OSEM or IIUM Ambulance to respond to an accident. If it is assigned to OSEM, the OSEM officer will perform the “Update Status” use case by providing updates to the system about their current activity. They will then perform the “Control Traffic” use case to manage the vehicle flow around the accident scene. Lastly, the “Secure Accident Area” use case will be implemented to ensure the accident zone is safe and secured. On the other hand, if it is dispatched towards IIUM Ambulance, their personnel will first use the “Update Status” use case, like OSEM where they update their current operational status. They will then proceed to use the “Provide First Aid” use case to deliver necessary medical treatment for victims involved. After that, “Transport Victims” use case will be implemented to transfer injured individuals to the nearby clinic or hospital for further treatments.

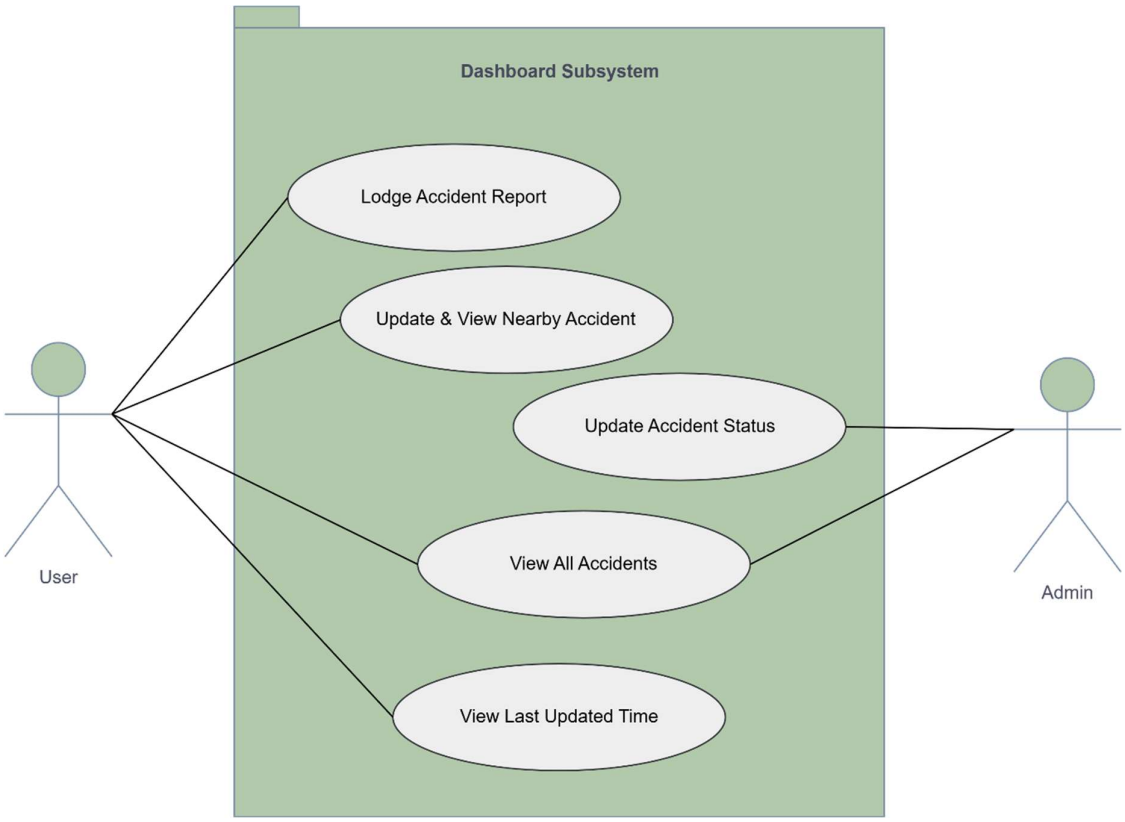

Figure 4: Use Case Diagram for Dashboard Subsystem by Putri Aimi

As shown in Figure 4, the use case diagram illustrates the main interactions between user and admin in the subsystem. The user lodge an accident report, update and view accident nearby the user location. Meanwhile, the admin manages to update the accident status, display all accident and last updated time by the specific admin. Finally, the user can view all the accidents happened and view the last updated time.

6. Class Diagram

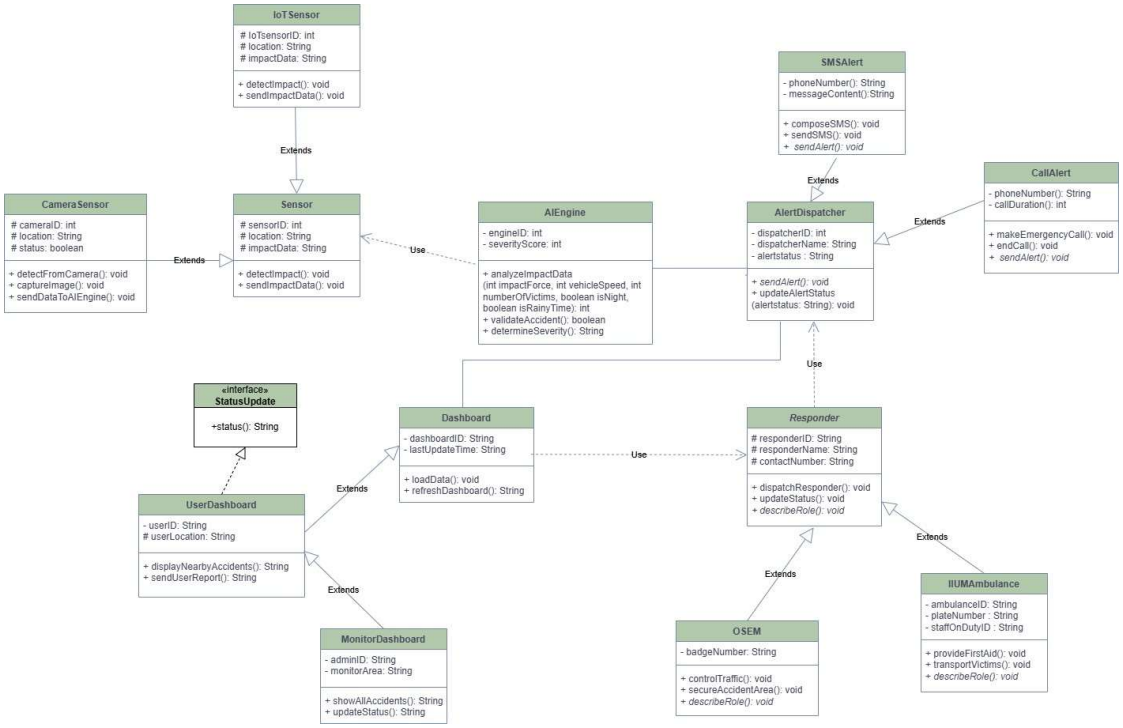

Figure 5: Class Diagram for IIUM Accident Detection System

As shown in Figure 5, the class diagram is the overall system of working classes to achieve the goal of this project, which is to enhance campus safety by enabling real-time accident detection, severity analysis, rapid emergency response, and providing monitoring dashboards. The system is designed using Object-Oriented Programming (OOP) principles, particularly inheritance, association, and dependency relationships. Sensor is the parent class that provides common attributes and methods to its child classes, IoTSensor and CameraSensor, which are responsible for detecting impact data and capturing visual evidence respectively. These data are then sent to the AIEngine, which depends on input from sensors to analyze accident severity. Once an accident is confirmed, the AIEngine triggers the AlertDispatcher class, which serves as the parent for SMSAlert and CallAlert, allowing alerts to be sent through different communication channels. The system maintains an association with the Responder class (parent of IIUMAmbulance, and OSEM), which models the emergency units that respond to alerts. Additionally, a Dashboard class, with a specialized UserDashboard subclass, is responsible for displaying real-time monitoring data and responder updates to authorized users and MonitorDahsboard is subclass of UserDashboard that for displaying data to administrators. Each relationship between classes supports smooth data flow and interaction, ensuring timely and coordinated response to any accident detected on campus.

7. Activity Diagram

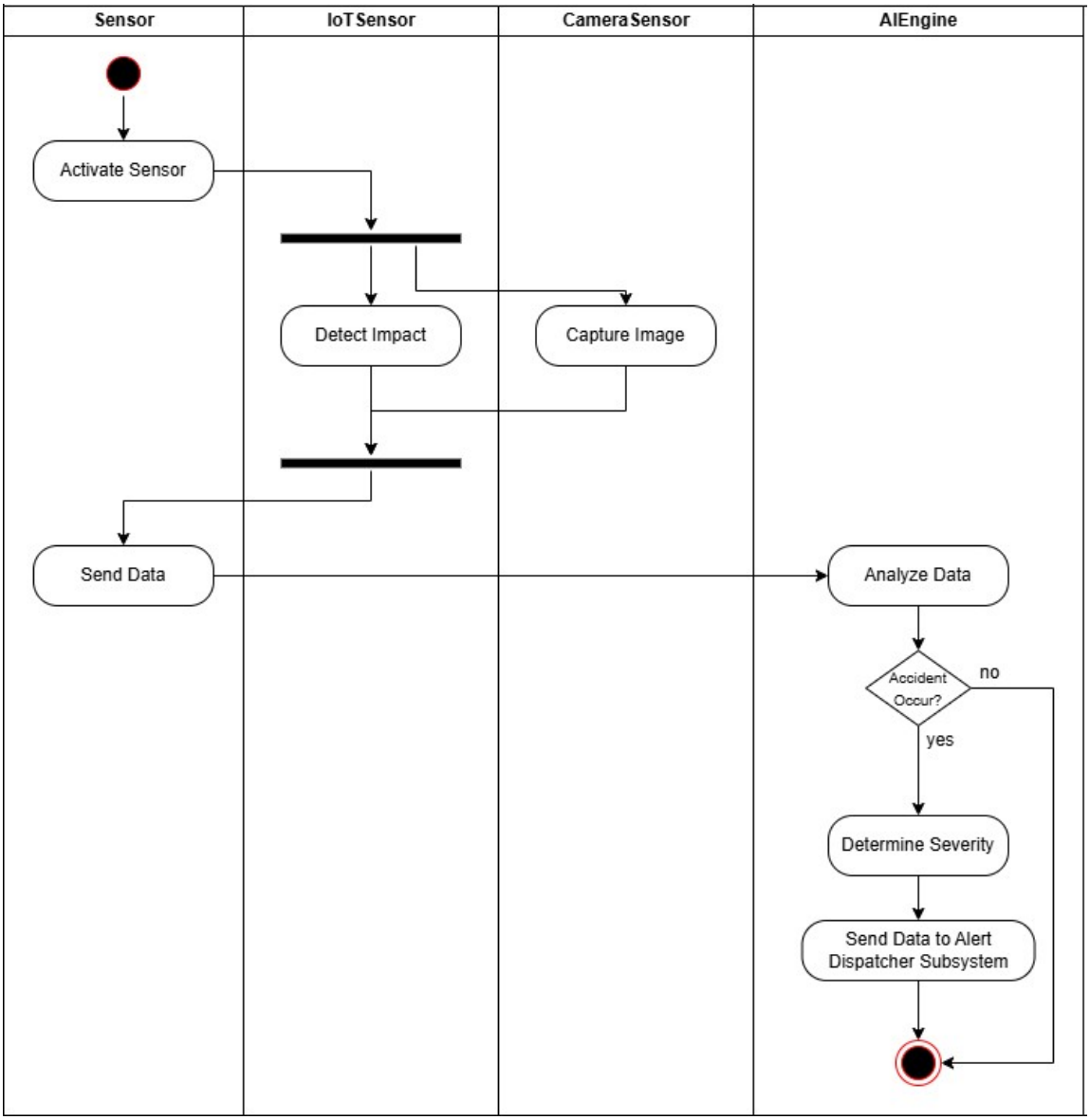

Figure 6: Activity Diagram for Accident Detection and Severity Analysis Subsystem by Sarah Yasmin

As shown in Figure 6, it is an activity diagram for the Accident Detection and Severity Analysis Subsystem that shows the flow of how the system is initiated by activating the sensor to detect an impact or capture an image of an incident happened and send that data to the AIEngine to be analyze and validate the accident. If the accident is considered as an accident, then the AIEngine will determine the severity level of the accident using predefined logic based on various factors such as impact force, speed, number of victims, and time of occurrence. Once the severity level is assessed, the AIEngine forwards this information to the Alert Dispatcher Subsystem, which is responsible for initiating the notification process to the relevant emergency responders based on the severity.

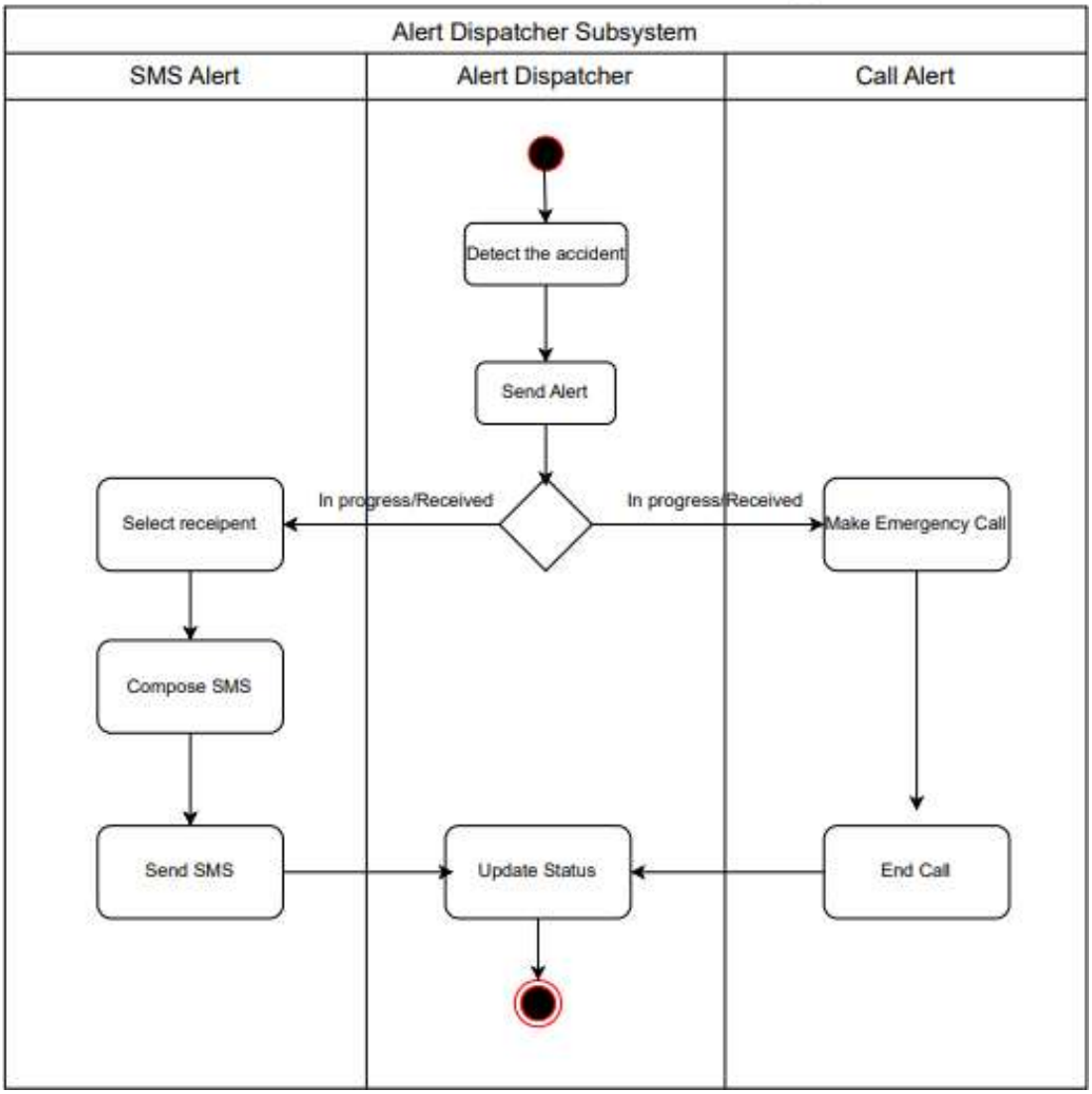

Figure 7: Activity Diagram for Alert Dispatcher Subsystem by Nur Balqis

As shown in Figure 7, the activity diagram for Alert Dispatcher Subsystem, the system detecting an accident, after which an alert is sent. A decision node evaluates the alert type, triggering either the SMS or call process. If the alert is handled via SMS, the system selects the recipient, composes the message, and sends it. For calls, the system makes an emergency call.

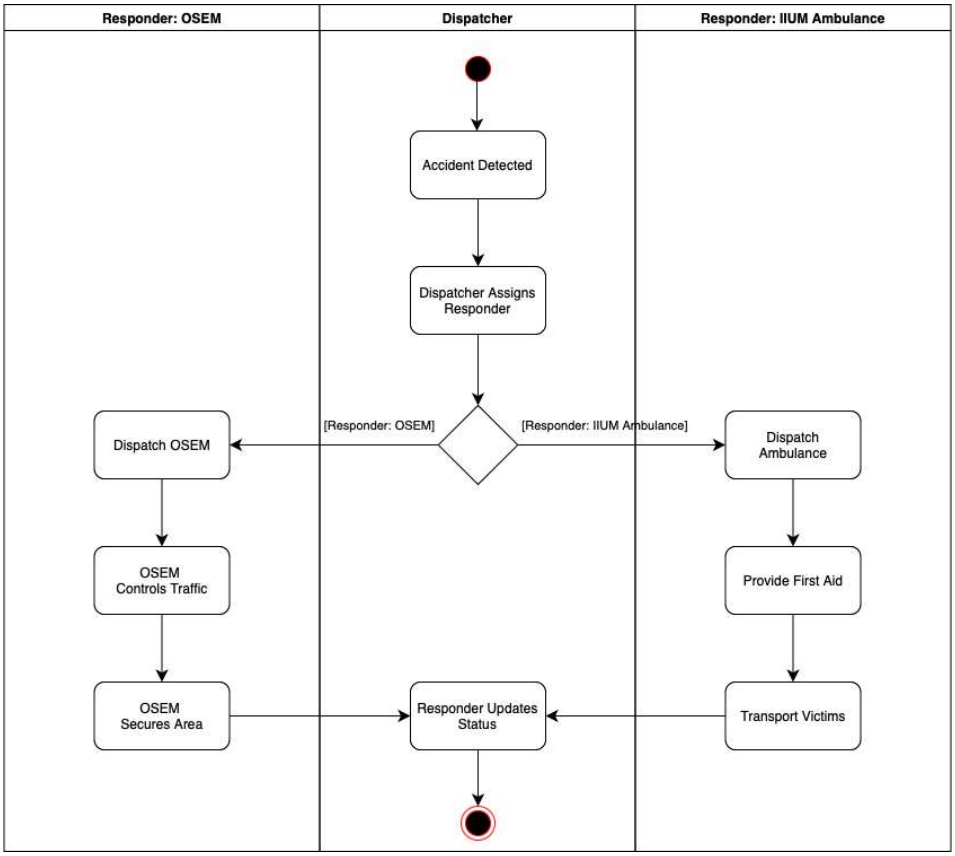

Figure 8: Activity Diagram for Responder Subsystem by Nurin Sofina

As shown in Figure 8, the activity diagram for Responder Subsystem occurs when an accident is being detected. The Dispatcher will first assign a responder either OSEM or IIUM Ambulance based on the situation. If OSEM is dispatched, they will proceed with controlling the traffic and secure the area to ensure safety. If IIUM Ambulance is dispatched, they provide first aid to victims and transport the victims to nearby medical facilities. After they complete their tasks, all responders will update their current status to the system.

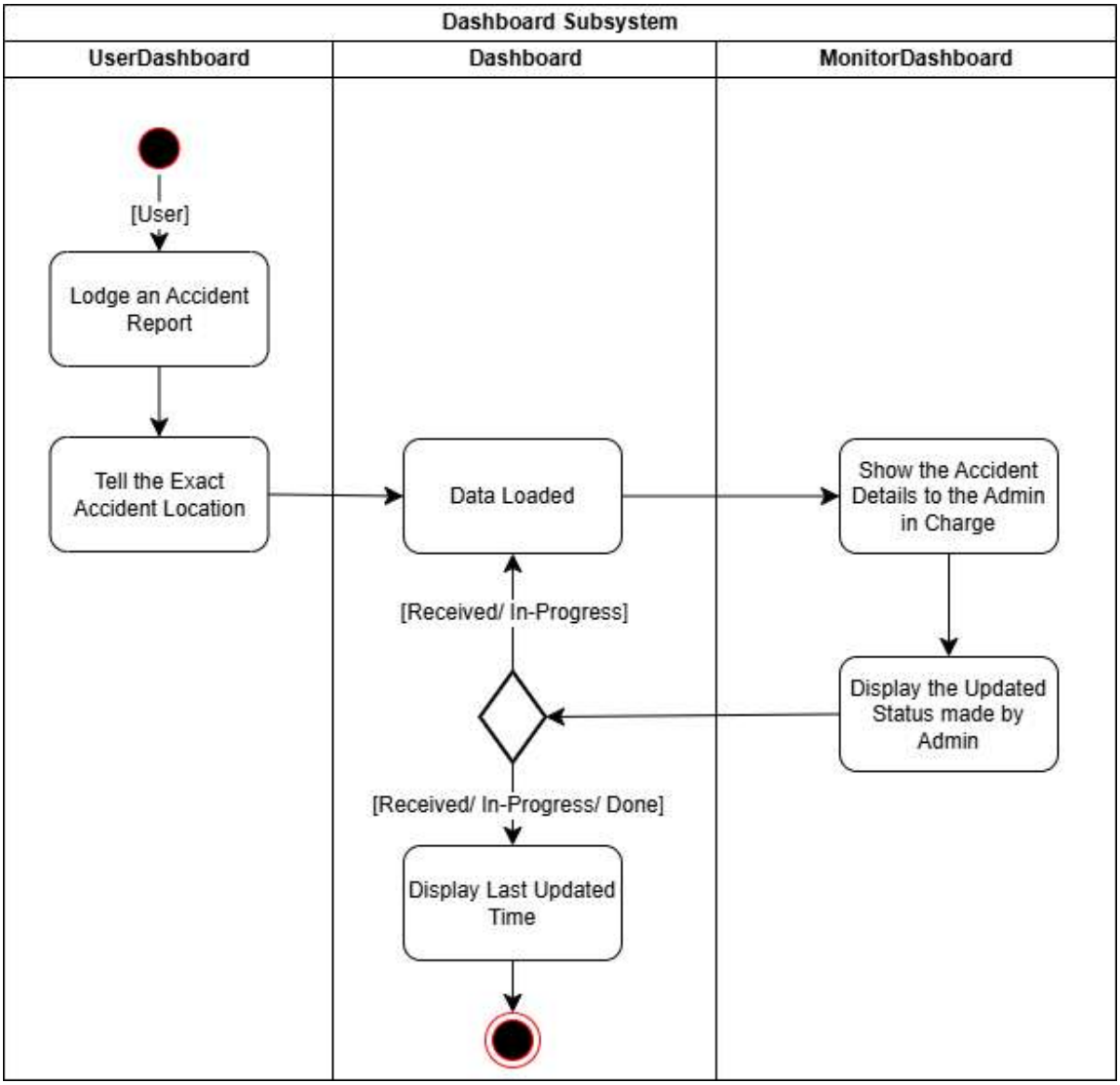

Figure 6: Activity Diagram for Dashboard Subsystem by Putri Aimi

As shown in Figure 9, the activity diagram for Dashboard Subsystem consists of 3 classes: Dashboard (Parent), Userdashboard (Child), and MonitorDashboard (Child). The program starts at the UserDashboard where it receives the user report, detect the accident nearby based on the user location. Then, the data loaded to the Dashboard before passing to the MonitorDashboard where it will display the accident report to the admin in charge. The admin will display the accident status (Received/ In-Progress/ Done). The program automatically includes last updated time in Dashboard. The status of accident iterates until it is “Done”.

8. CODING DISTRIBUTION

The coding distribution in Table 1 reflects how the project team strategically divided the system into four key subsystems, which are Accident Detection and Severity Analysis, Alert Dispatcher, Dashboard, and Responder to effectively manage the workload and achieve the overall goal of enhancing campus safety. Each subsystem was assigned to a specific team member based on their role and area of expertise. This clear division allowed the team to work in parallel, ensuring smooth development and integration of all components. By handling diagrams and coding responsibilities within their assigned subsystems, each member contributed to the system's functionality from detecting and analyzing accidents, to dispatching alerts, coordinating emergency responders, and managing real-time monitoring to ensuring a complete and well-organized system.

Table 1: Distribution of Coding

Name and Matric No	Role	Responsibility	Status (In-progress, completed, Pending)
SARAH YASMIN BINTI RODZMAN (2413034)	PROJECT MANAGER	Accident Detection and Severity Analysis Subsystem (Use-Case Diagram, Activity Diagram, Coding)	Completed
NUR BALQIS BINTI MOHD KAMARULZAMAN (2410006)	FILE HANDLER	Alert dispatcher Subsystem (Use-Case Diagram, Activity Diagram, Coding)	Completed
NURIN SOFINA BINTI YUSDI (2221372)	GUI DESIGNER	Responder Subsystem (Use-Case Diagram, Activity Diagram, Coding)	Completed
PUTRI AIMI BATRISYIA BINTI MUHAMMAD YUSRI (2320206)	BACKEND DEVELOPER	Dashboard Subsystem (Use-Case Diagram, Case Diagram, Activity Diagram, Coding)	Completed

9. LESSON LEARNED

9.1 Lesson 1

Through this project, we learned how to implement the object-oriented programming elements that we have learned throughout this course. The elements include concepts like inheritance, polymorphism, interface and abstraction.

9.2 Lesson 2

We also learned how to model system behavior with UML Class Diagram, Use-Case Diagram, and Activity Diagram. These three Diagram conduct a difference visualisation yet it could be an effective and efficient to initiate a system flow. It also shows a clear and simple concept to describe how the system works for the stakeholder.

9.3 Lesson 3

Most importantly, we learned how various classes or components such as sensor, dispatcher, responder, and dashboard play a big role in creating a coordinated and efficient accident detection system for the university. We need to ensure all the subsystems integrate with each other to avoid data silos. Entirely, we hope that this system will run smoothly and well-integrated.

10. References

Handayani, D. O. D. (2025). Topic 2: Introduction to Unified Modelling Language (UML). Slides BICS 1304. International Islamic University Malaysia.

Liang, Y. D. (2025). Introduction to Java programming and data structures (12th ed.). Pearson

11. APPENDIX-1: CODING

- **Accident Detection and Severity Analysis Subsystem (Sarah Yasmin)**

Sensor Class (Parent class)

```
package sensor;
public class Sensor {
    protected int sensorID;
    protected String location;
    protected String impactData;

    public Sensor(int sensorID, String location, String impactData) {
        this.sensorID = sensorID;
        this.location = location;
        this.impactData = impactData;
    }

    public void detectImpact() {
        System.out.println("Impact detected at " + location);
    }

    public void sendImpactData() {
        System.out.println("Sending impact data: " + impactData);
    }
}
```

IoTSensor Class (inherits Sensor Class)

```
package sensor;
public class IoTSensor extends Sensor {

    public IoTSensor(int sensorID, String location, String impactData) {
        super(sensorID, location, impactData);
    }

    @Override
    public void detectImpact() {
        System.out.println("IoT sensor detecting impact at " + location);
    }

    @Override
    public void sendImpactData() {
        System.out.println("IoT sensor sending impact data: " + impactData);
    }
}
```

CameraSensor Class (inherits Sensor Class)

```
package sensor;
public class CameraSensor extends Sensor {
```

```
    protected boolean status;

    public CameraSensor(int sensorID, String location, String impactData, boolean status) {
        super(sensorID, location, impactData);
        this.status = status;
    }

    public void detectFromCamera() {
        System.out.println("Camera detected movement at " + location);
    }

    public void captureImage() {
        System.out.println("Capturing image at " + location);
    }

    public void sendDataToAIEngine() {
        System.out.println("Sending image data to AI Engine...");
    }
}
```

AIEngine Class

```
public class AIEngine {
    private int engineID;
    private int severityScore;

    public AIEngine(int engineID) {
        this.engineID = engineID;
    }

    public int analyzeImpactData(int impactForce, int vehicleSpeed,
                                 int numberOfVictims,boolean isNight,
                                 boolean isRainyTime){

        /* to calculate severityScore, impactForce, vehicleSpeed and
           numberOfVictims are essential to be measured */
        severityScore = (impactForce * 2) +
                        (vehicleSpeed * 1) +
                        (numberOfVictims * 10);

        if (isNight) {
            severityScore += 10;
        }

        // severityScore will be added if it is raining
        if (isRainyTime) {
            severityScore += 20;
        }

        System.out.println("Severity Score: " + severityScore);
        return severityScore;
    }
```

```
    public boolean validateAccident() {
        return severityScore > 50; //an accident occured
    }

    public String determineSeverity() {
        if (severityScore >= 80) {
            return "High"; // more serious responder need to be alert
        } else if (severityScore >= 50) {
            return "Moderate";
        } else {
            return "Low";
        }
    }
}
```

- **Alert Dispatcher Subsystem (Nur Balqis)**

Alert Dispatcher (Parent and abstract)

```
package alert;

public abstract class AlertDispatcher {
   protected int dispatcherID;
   protected String dispatcherName;
   protected String contactInfo;

   public AlertDispatcher(int dispatcherID, String dispatcherName, String contactInfo) {
      this.dispatcherID = dispatcherID;
      this.dispatcherName = dispatcherName;
      this.contactInfo = contactInfo;
   }

public int getDispatcherID() {
      return dispatcherID;
   }
   public void setDispatcherID(int dispatcherID) {
      this.dispatcherID = dispatcherID;
   }

   public String getDispatcherName() {
      return dispatcherName;
   }
   public void setDispatcherName(String dispatcherName) {
      this.dispatcherName = dispatcherName;
   }
   public String getContactInfo() {
      return contactInfo;
   }
   public void setContactInfo(String contactInfo) {
      this.contactInfo = contactInfo;
   }

   public abstract void sendAlert();
}
```

SMS Alert (Child class)

```
package alert;

public class SMSAlert extends AlertDispatcher {
   private String message;

   public SMSAlert(int dispatcherID, String dispatcherName, String contactInfo, String
message) {
      super(dispatcherID, dispatcherName, contactInfo);
      this.message = message;
```

```
    }

 public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }

    @Override
    public void sendAlert() {
        System.out.println("Sending SMS to " + contactInfo + ": " + message);
    }
}
```

Call Alert (Child Class)

```
package alert;

public class CallAlert extends AlertDispatcher {
    private int durationInSeconds;

    public CallAlert(int dispatcherID, String dispatcherName, String contactInfo, int
durationInSeconds) {
        super(dispatcherID, dispatcherName, contactInfo);
        this.durationInSeconds = durationInSeconds;
    }
  public int getDurationInSeconds() {
        return durationInSeconds;
    }
    public void setDurationInSeconds(int durationInSeconds) {
        this.durationInSeconds = durationInSeconds;
    }

    @Override
    public void sendAlert() {
        System.out.println("Placing call to " + contactInfo + " for " + durationInSeconds + "
seconds.");
    }
}
```

- **Responder Subsystem (Nurin Sofina)**

Responder Class (Parent & Abstract Class)

```
package responder;
public abstract class Responder {
    protected String responderID;
    protected String responderName;
    protected String contactNumber;
    public Responder (String responderID, String responderName, String contactNumber) {
        this.responderID = responderID;
        this.responderName = responderName;
        this.contactNumber = contactNumber;
    }
    public void dispatchResponder() {
        System.out.println("Dispatching responder: " + responderName + " (ID: " +
responderID + ") ");
    }

    public void updateStatus(String status) {
        System.out.println("Responder " + responderName + " updated status: " + status);
    }

    public abstract void describeRole();
}
```

OSEM (Child Class)

```
package responder;
public class OSEM extends Responder {
    private String badgeNumber;

    public OSEM(String responderID, String responderName, String contactNumber, String
badgeNumber) {
        super(responderID, responderName, contactNumber);
        this.badgeNumber = badgeNumber;
    }

    @Override
    public void describeRole() {
        System.out.println("OSEM " + responderName + " (Badge: " + badgeNumber + ")
ensures scene safety and manages traffic.");
    }

    public void secureAccidentArea() {
        System.out.println("OSEM " + responderName + " is securing the accident area.");
    }

    public void controlTraffic() {
        System.out.println("OSEM " + responderName + " is controlling traffic.");
    }
```

```
}
```

IIUMAmbulance (Child Class)

```
package responder;
public class IIUMAmbulance extends Responder {
    private String vehicleID;
    private String staffID;

    public IIUMAmbulance(String responderID, String responderName, String
contactNumber, String vehicleID, String staffID) {
        super(responderID, responderName, contactNumber);
        this.vehicleID = vehicleID;
        this.staffID = staffID;
    }

    @Override
    public void describeRole() {
        System.out.println("IIUM Ambulance " + responderName + " (Vehicle: " + vehicleID +
", Staff: " + staffID + ") provides medical aid.");
    }

    public void provideFirstAid() {
        System.out.println("Ambulance " + responderName + " is providing first aid.");
    }

    public void transportVictims() {
        System.out.println("Ambulance " + responderName + " is transporting victims.");
    }
}
```

- **Dashboard Subsystem (Putri Aimi)**

Dashboard (Parent Class)

```
package dashboard;
public class Dashboard {private String dashboardID; private String lastUpdateTime;
//Parametered Constructor
public Dashboard(String dashboardID, String lastUpdateTime){
    this.dashboardID = dashboardID;
    this.lastUpdateTime = lastUpdateTime;
}

/**
* @return the dashboardID
*/
public String getDashboardID() {
    return dashboardID;
}

/**
* @param dashboardID the dashboardID to set
*/
public void setDashboardID(String dashboardID) {
    this.dashboardID = dashboardID;
}

/**
* @return the lastUpdateTime
*/
public String getLastUpdateTime() {
    return lastUpdateTime;
}

/**
* @param lastUpdateTime the lastUpdateTime to set
*/
public void setLastUpdateTime(String lastUpdateTime) {
    this.lastUpdateTime = lastUpdateTime;
}

//Method to load the Accident Data
public void loadData(){
    System.out.println("Loading data from ---" +dashboardID+ "---");
}

//Method to display last updated time
public String refreshDashboard(){
    return "Last updated at: " +lastUpdateTime;
}
```

UserDashboard (Child Class of Dashboard)

```
package dashboard;
```

```
public class UserDashboard extends Dashboard implements StatusUpdate {
   private String userID;
   protected String userLocation;

   public UserDashboard(String dashboardID, String lastUpdateTime, String userID, String
userLocation) {
      super(dashboardID, lastUpdateTime);
      this.userID = userID;
      this.userLocation = userLocation;
   }

   public String displayNearbyAccidents() {
      return "nearby the " + userLocation;
   }

   public String sendUserReport() {
      return userID + " lodged an accident report.";
   }

   @Override
   public String toString() {
      return "User ID: " + sendUserReport() + " " + displayNearbyAccidents();
   }

   @Override
   public String status() {
      return "Accident Status: Active";
   }
}
```

MonitorDashboard (Child Class of UserDashboard)

```
package dashboard;
public class MonitorDashboard extends UserDashboard {private String adminID; private
String monitorArea;
//Parametered Constructor
public MonitorDashboard(String dashboardID,String lastUpdateTime,String userID, String
userLocation, String adminID, String monitorArea){
   super(dashboardID, lastUpdateTime, userID, userLocation);
   this.adminID = adminID;
   this.monitorArea = monitorArea; //general location
}

/**
* @return the adminID
*/
public String getAdminID() {
   return adminID;
}

/**
```

```
* @return the monitorArea
*/
public String getMonitorArea() {
    return monitorArea;
}

//Method to display the Accident Details to the admin in charge
public String showAllAccidents(){
    return "--- Accident Reports ---"
            +"\nMonitored Area: " +monitorArea
            +"\nAccident Location: " +userLocation
            +"\nAdmin in Charge: " +adminID;
}

//Display the updated status by admin
public String updateStatus(){
    return refreshDashboard() +" by Admin ID: " +adminID;
}

//Polymorphism
@Override
public String toString(){
    return showAllAccidents()+ "\n" +status()+ "\n\n" +updateStatus();
}
```