



PES University, Bengaluru

(Established under Karnataka Act 16 of 2013)

Department of Computer Science & Engineering
Session: Jan - May 2022

UE19CS353 – Object Oriented Analysis and Design with Java
Theory ISA (Mini Project)

Report on
Flood Relief Management System

By:

Gayathri Sunil - PES1UG19CS167

Jahnavi Sivaram - PES1UG19CS192

Jigisha M Narrain - PES1UG19CS197

6th Semester C

1. Project Description

The problem statement is to create a flood relief management system that can help the affected families and assist in disaster management by connecting NGOs and relief services to those in need.

This project implements a flood relief management system, with a focus on humanitarian aid during the disaster. It aims to connect affected persons with both official and unofficial channels of assistance during different phases of the natural disaster itself.

Possible aid during a flood includes search and rescue, evacuation, provision of shelter, food and medicines, first aid, and money. These necessities can be sourced from government bodies, emergency personnel such as police, firefighters and paramedics, established NGOs that specialise in disaster relief, and from common people.

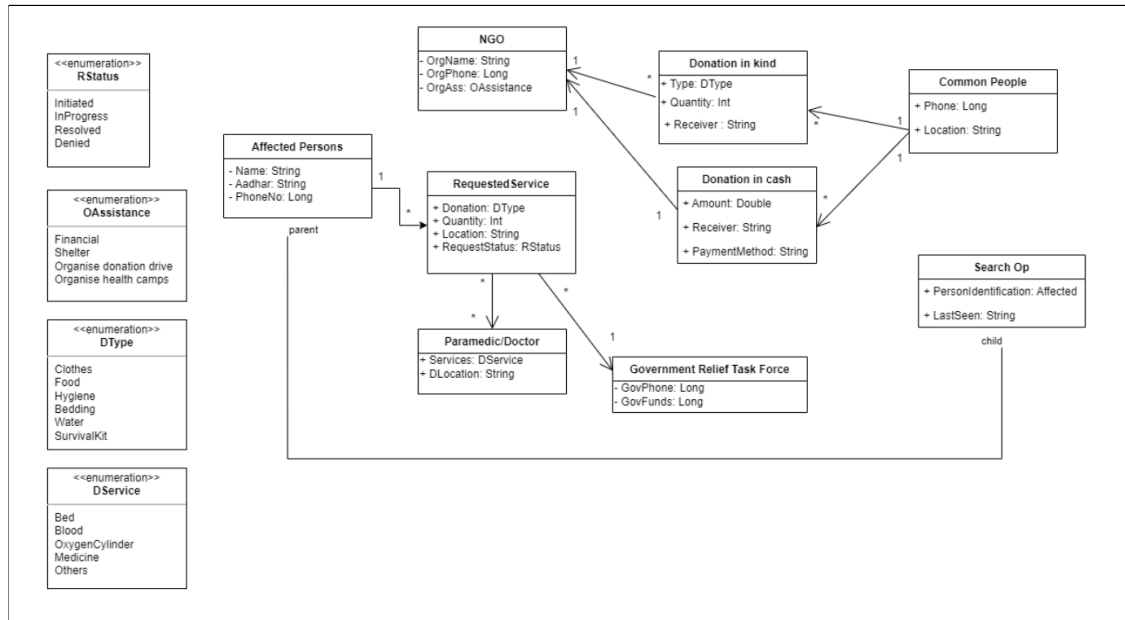
It is important to facilitate both serviceable requests and quick responses from all parties involved, and ensure that efficient channels are operational for these purposes.

Link to Github repository -

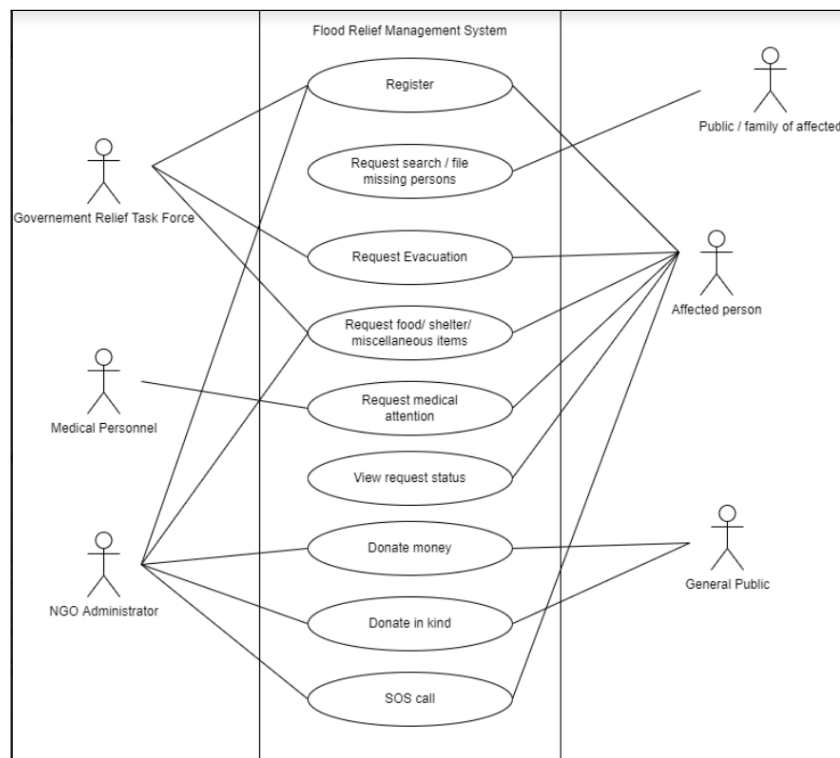
[Flood Relief Management System](#)

2. Analysis and Design Models

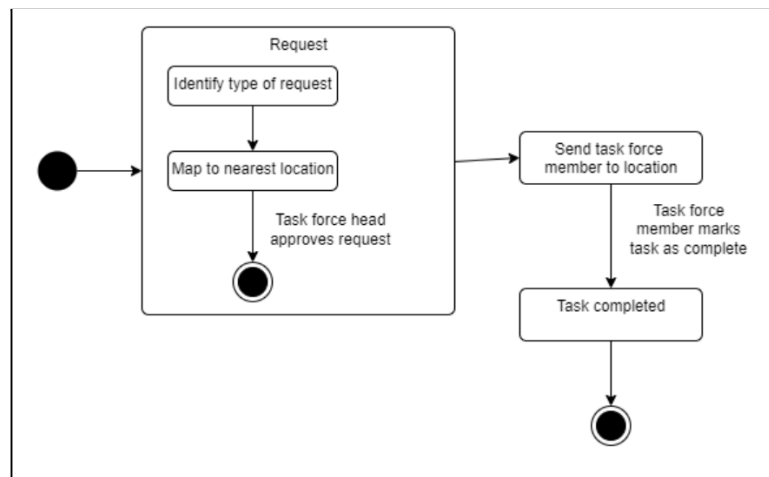
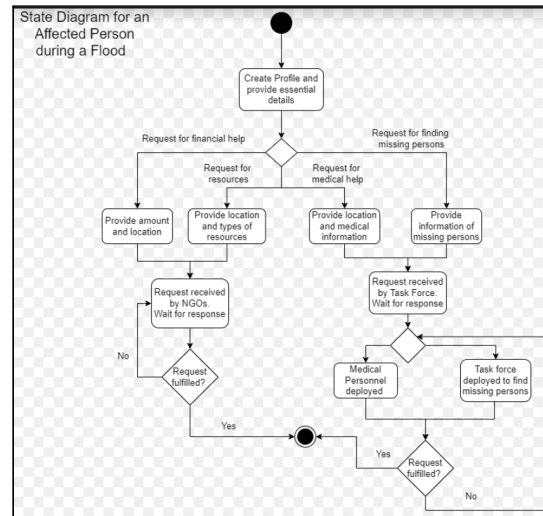
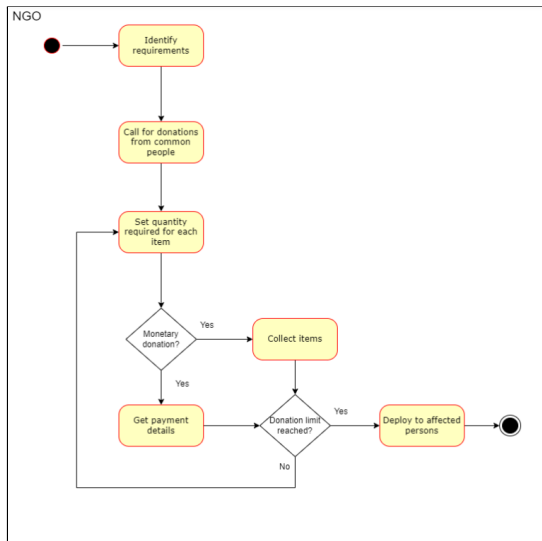
Class Diagram:



Use Case Diagram:

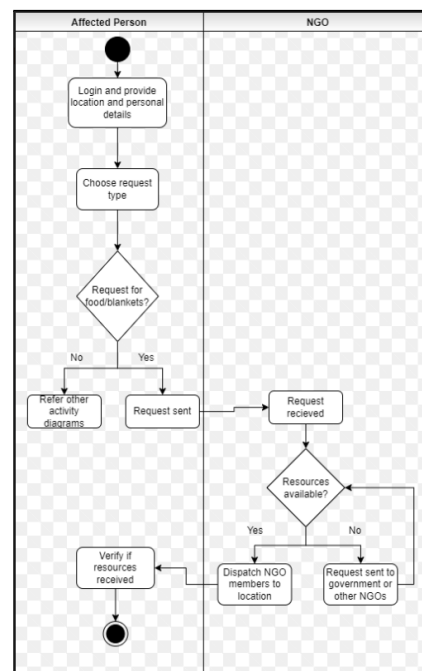
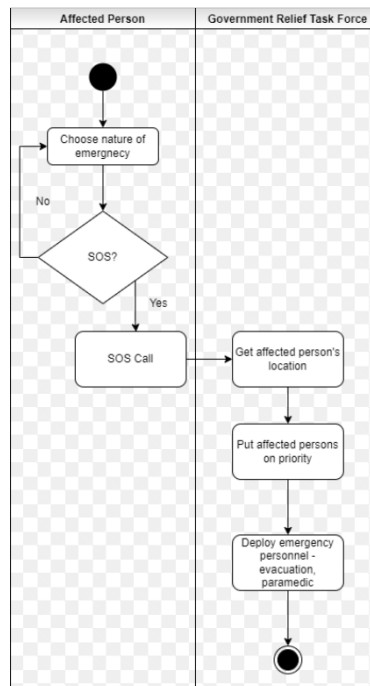
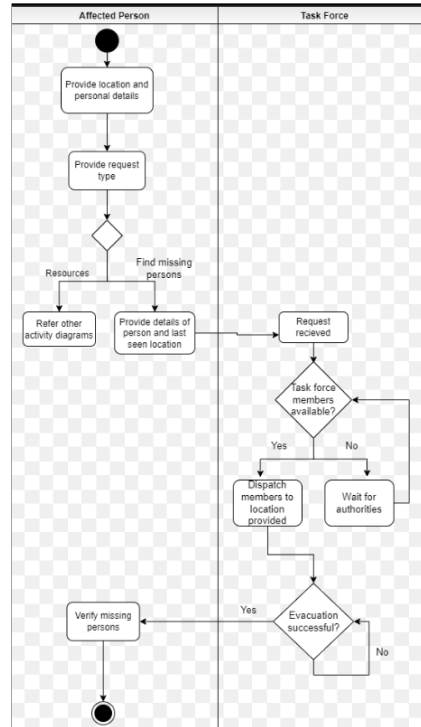
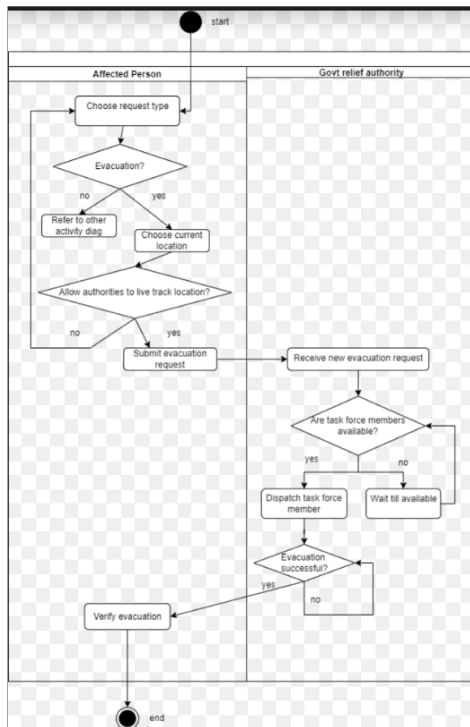


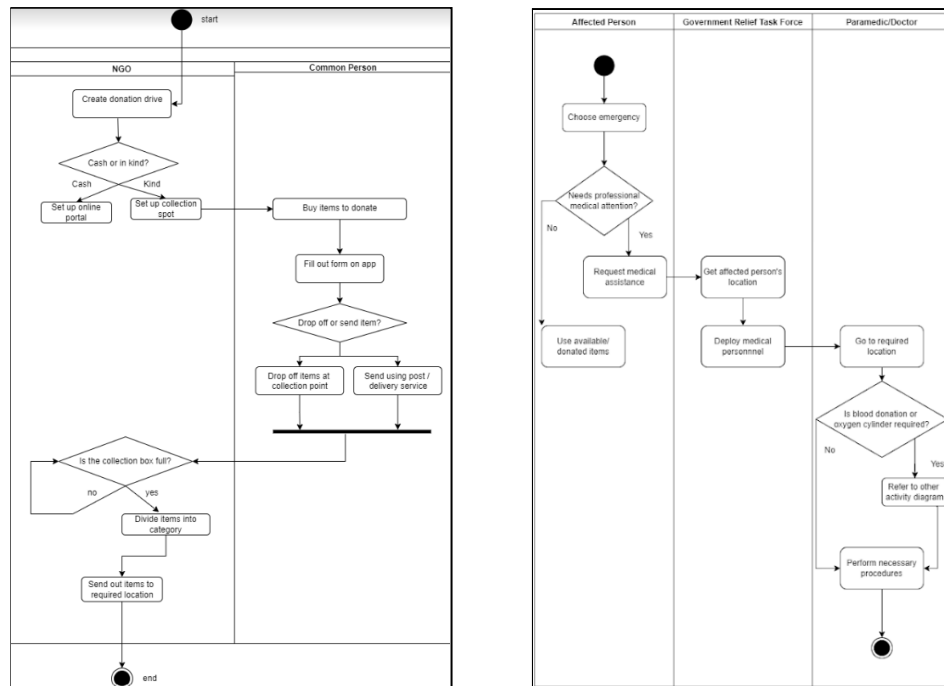
State Diagrams:



(Government Task Force)

Activity Diagrams:



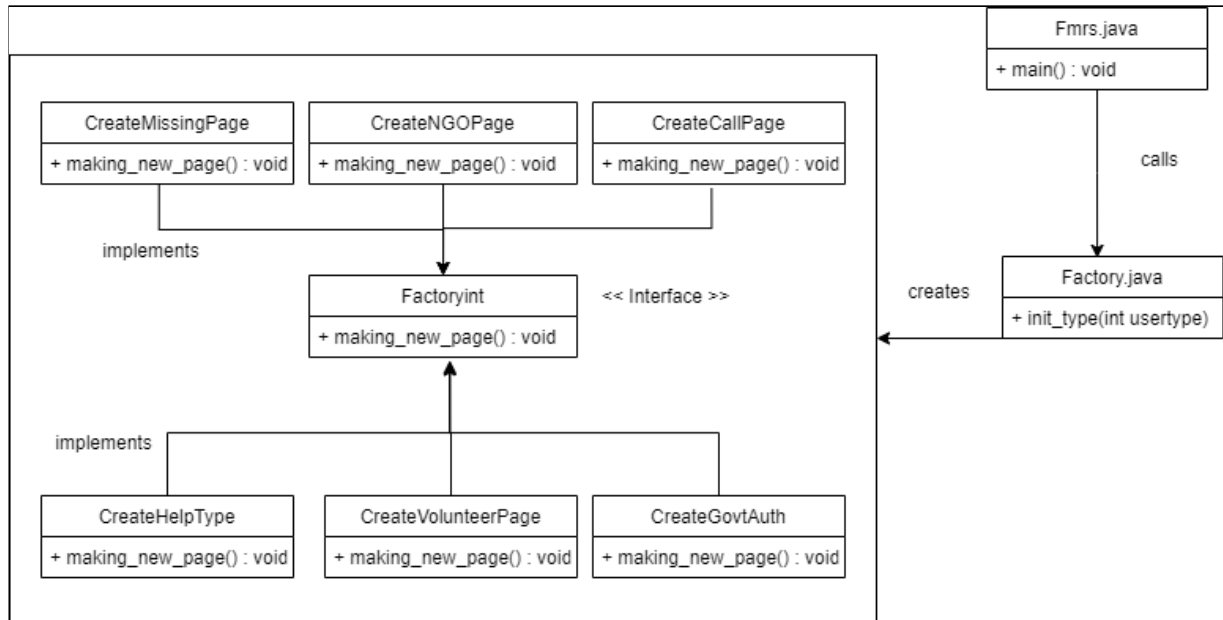


3. Tools and Frameworks Used

- javafx.swing for the GUI
- PostgreSQL database backend, using a JDBC PostgreSQL driver to connect it to the JRE
- java.awt for events and triggers
- java.sql for the database backend connection, exception and querying

4. Design Principles and Design Patterns Applied

a. Factory method



For creation of each instance in the landing page:

We have created an interface called **Factoryint**, which acts as the abstract class to create an object. Each subclass, all of whose methods are called in **Factory.java**, which is responsible for a specific functionality, is responsible for its relevant instantiation.

```
package frms;

public interface Factoryint {
    void making_new_page();
}
```

Factoryint.java

```

package frms;

public class Factory {
    public static void init_type(int usertype){
        switch(usertype){
            case 1: {
                Factoryint fil = new CreateHelpType();
                fil.making_new_page();
                break;
            }
            case 2: {
                Factoryint fil = new CreateVolunteerPage();
                fil.making_new_page();
                break;
            }
            case 3: {
                Factoryint fil = new CreateGovtAuth();
                fil.making_new_page();
                break;
            }
            case 4: {
                Factoryint fil = new CreateMissingPage();
                fil.making_new_page();
                break;
            }
            case 5: {
                Factoryint fil = new CreateNGOPage();
                fil.making_new_page();
                break;
            }
        }
    }
}

```

Factory.java

```

package frms;

public class CreateGovtAuth implements Factoryint {
    @Override
    public void making_new_page(){
        GovtAuth govpage = new GovtAuth();
        govpage.setVisible(true);
    }
}

```

CreateGovtAuth.java - one of the classes that implements Factoryint.java


```
package frms;

public class CreateHelpType implements Factoryint {
    @Override
    public void making_new_page() {
        HelpType helpchoice = new HelpType();
        helpchoice.setVisible(true);
    }
}
```

CreateHelpType.java - one of the classes that implements Factoryint.java

b. Open-Close principle from SOLID

Our database methods are only open for extension, not modification.

```
public String get_from_db(String table, String cond, String val, String fetch){
    query = "SELECT " + fetch + " FROM " + table + " WHERE " + cond + "= '" + val + "'";
    ExecutedBQ exec = new ExecuteDBQ();
    result = exec.fetch_execute(query, fetch);
    return result;
}

public String[] fetch_all(String table, int n){
    query = "SELECT * FROM " + table + ";";
    query.toString();
    ExecutedBQ exec = new ExecuteDBQ();
    arr_res = exec.fetch(query, n);
    return arr_res;
}
```

DB.java

Instead of adding condition check value in fetch_all() and modifying it, we make a new function get_from_db() so that each function can be used to get rows from the database based on whether we need to get rows on condition or not

Similarly for making the query to insert into the database-

```
public void insert_into_db_reg(String mobnum, String aadhar){
    query = "INSERT INTO reg_for_req VALUES ('" + mobnum + "', '" + aadhar + "')";
    ExecutedBQ exec = new ExecuteDBQ();
    exec.execute_query(query);
}

public void insert_into_evac(String mobnum, String name, String location){
    query = "INSERT INTO evac_req VALUES ('" + mobnum + "', '" + name + "', '" + location + "')";
    ExecutedBQ exec = new ExecuteDBQ();
    exec.execute_query(query);
}

public void insert_into_res(String mobnum, String item, String amt){
    query = "INSERT INTO resource_req VALUES ('" + mobnum + "', '" + item + "', '" + amt + "')";
    ExecutedBQ exec = new ExecuteDBQ();
    exec.execute_query(query);
}
```

DB.java

c. Single responsibility principle from SOLID

The *ExecuteDBQ.java* class is responsible only for executing queries related to the database and does not perform any other function.

```
public class ExecuteDBQ {
    String database_conn_string = "jdbc:postgresql://localhost:5432/postgres";
    String database_user_name = "postgres";
    String database_user_password = "1612";
    Connection conn = null;

    String query;
    public void execute_query(String stmt){

        query = stmt;
        try {
            conn= DriverManager.getConnection(database_conn_string, database_user_name, database_user_password);
            PreparedStatement statement = conn.prepareStatement(query);
            int rowsInserted = statement.executeUpdate();

            if (rowsInserted > 0){
                System.out.println("A new entry was inserted successfully!");
            }
        }
        catch (SQLException e){
            System.out.println(e.getMessage());
        }
    }
}
```

ExecuteDBQ.java

```
public String fetch_execute(String stmt, String cond){
    query = stmt;

    try {
        conn= DriverManager.getConnection(database_conn_string, database_user_name, database_user_password);

        Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery(query);

        while (rs.next()){
            //System.out.println(rs.getString(cond));
            return rs.getString(cond);
        }

    }
    catch (SQLException e){
        //System.out.println(e.getMessage());
    }
    return "";
}
```

ExecuteDBQ.java

d. MVC in NGO

i. **Model:**

Consists of all getters and setters.

```
public class NGO_Model {  
    String NGO_name, NGO_mob, NGO_addr, NGO_stat;  
  
    public NGO_Model(String name, String mob, String addr, String stat) {  
        NGO_name = name;  
        NGO_mob = mob;  
        NGO_addr = addr;  
        NGO_stat = stat;  
    }  
  
    public void setName(String name){  
        this.NGO_name = name;  
    }  
  
    public void setMob (String mob){  
        this.NGO_mob = mob;  
    }  
  
    public void setAddr(String addr){  
        this.NGO_addr = addr;  
    }  
  
    public void setStat(String stat){  
        this.NGO_stat = stat;  
    }  
  
    public String getName(){  
        return NGO_name;  
    }  
}
```

NGO_Model.java

ii. **View:**

Java swing code to generate the following GUI.

Register your NGO

NGO name

Contact Number

Address

Status (Y or N)

To update Status, enter NGO name and Status and click update:

NGO.java

iii. Controller:

Controller acts as an interface between View and Model.

```
public class NGO_Controller{

    String name,mob,addr,stat;
    public NGO_Controller(String NGO_name, String NGO_mob, String NGO_addr, String NGO_stat){
        name = NGO_name;
        mob = NGO_mob;
        addr = NGO_addr;
        stat = NGO_stat;
    }

    public void process_reg(){
        //call add_to_db
        DB dbreq = new DB();
        dbreq.insert_into_ngo_reg(name, mob, addr, stat);

        JOptionPane.showMessageDialog(null, "Registered NGO successfully ", "Success", JOptionPane.INFORMATION_MESSAGE);
    }

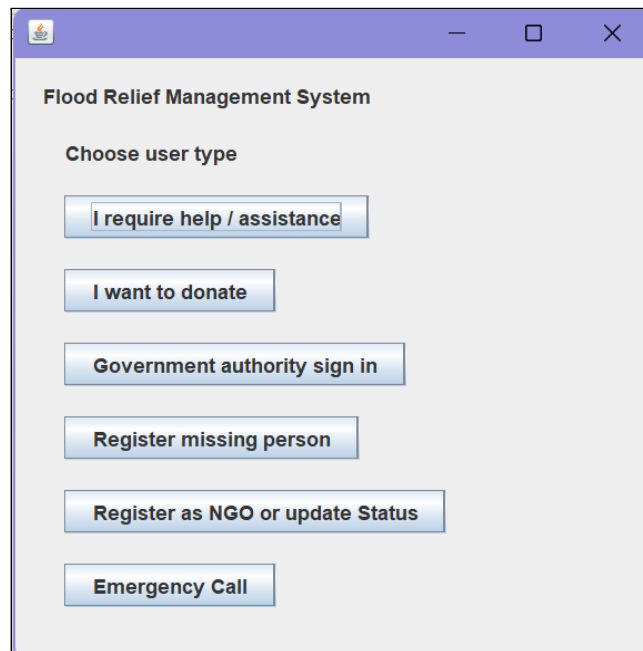
    public void update_stat(){
        //call to db to update
        DB dbreq = new DB();
        dbreq.update_ngo(name, stat);
        JOptionPane.showMessageDialog(null, "Updated status" , "Success", JOptionPane.INFORMATION_MESSAGE);
    }

}
```

NGO_Controller.java

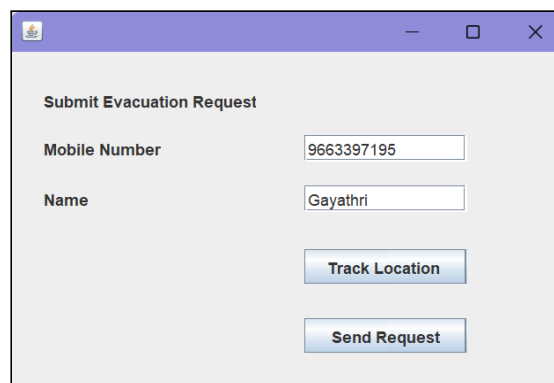
5. Application Screenshots

a. Landing page

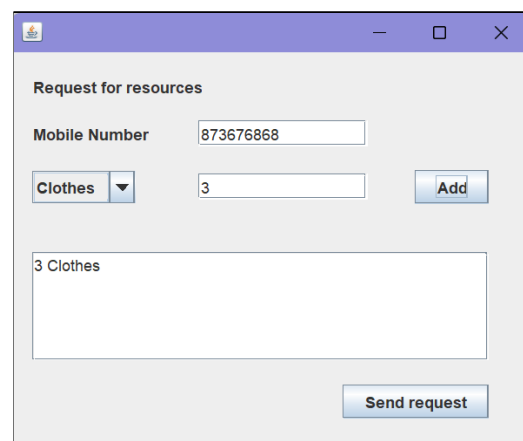


The screenshot shows a web application window titled "Flood Relief Management System". Below the title, there is a section labeled "Choose user type" with several buttons arranged vertically: "I require help / assistance", "I want to donate", "Government authority sign in", "Register missing person", "Register as NGO or update Status", and "Emergency Call".

b. Flood Victims can request for help

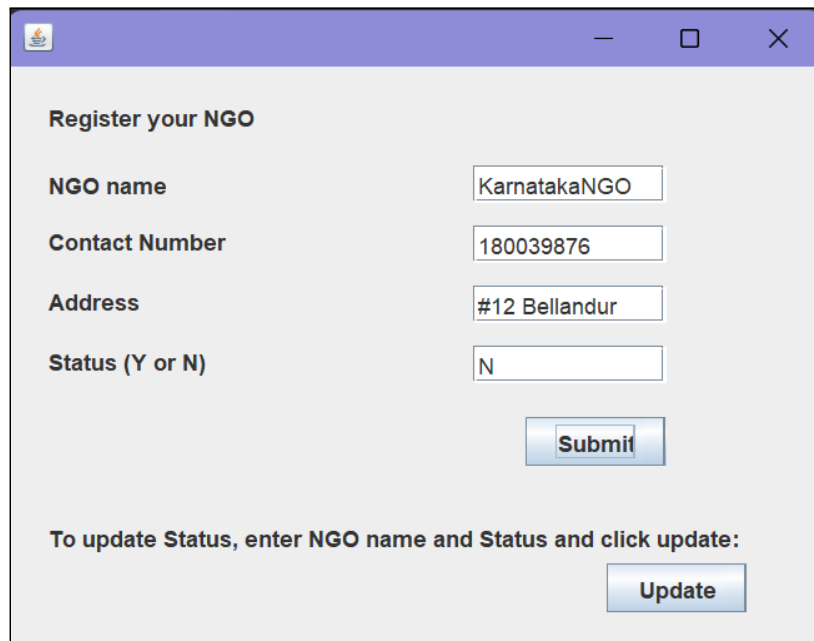


The screenshot shows a web application window titled "Submit Evacuation Request". It contains two input fields: "Mobile Number" with the value "9663397195" and "Name" with the value "Gayathri". Below these fields are two buttons: "Track Location" and "Send Request".



The screenshot shows a web application window titled "Request for resources". It contains an input field for "Mobile Number" with the value "873676868". Below this is a section for adding resources, with a dropdown menu set to "Clothes" and an input field with the value "3". To the right of the input field is an "Add" button. Below this section is a list box showing "3 Clothes". At the bottom right of the window is a "Send request" button.

NGOs can register and accept donations

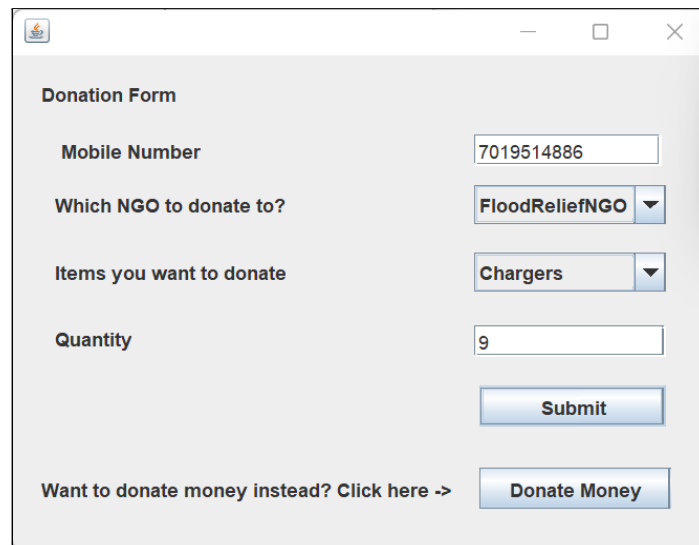


A screenshot of a web application window titled "Register your NGO". The window has a purple header bar with standard window controls (minimize, maximize, close). The main content area is light gray and contains the following fields:

- NGO name:** A text input field containing "KarnatakaNGO".
- Contact Number:** A text input field containing "180039876".
- Address:** A text input field containing "#12 Bellandur".
- Status (Y or N):** A text input field containing "N".

Below the fields are two buttons: a "Submit" button and an "Update" button. The "Update" button is disabled. At the bottom, there is a text instruction: "To update Status, enter NGO name and Status and click update:" followed by the "Update" button.

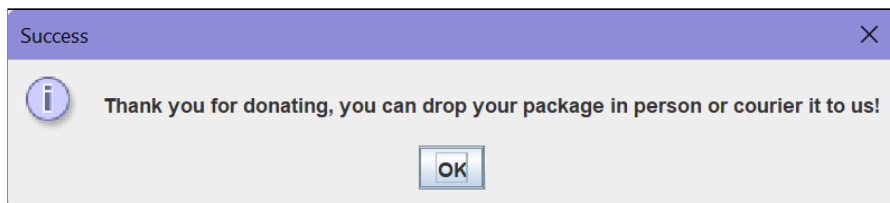
Common people can donate to NGOs or government organisations based on whether they are currently accepting donations



A screenshot of a web application window titled "Donation Form". The window has a white header bar with standard window controls. The main content area is light gray and contains the following fields:

- Mobile Number:** A text input field containing "7019514886".
- Which NGO to donate to?:** A dropdown menu with "FloodReliefNGO" selected.
- Items you want to donate:** A dropdown menu with "Chargers" selected.
- Quantity:** A text input field containing "9".

Below the fields are two buttons: a "Submit" button and a "Donate Money" button. The "Donate Money" button is disabled. At the bottom, there is a text instruction: "Want to donate money instead? Click here ->" followed by the "Donate Money" button.

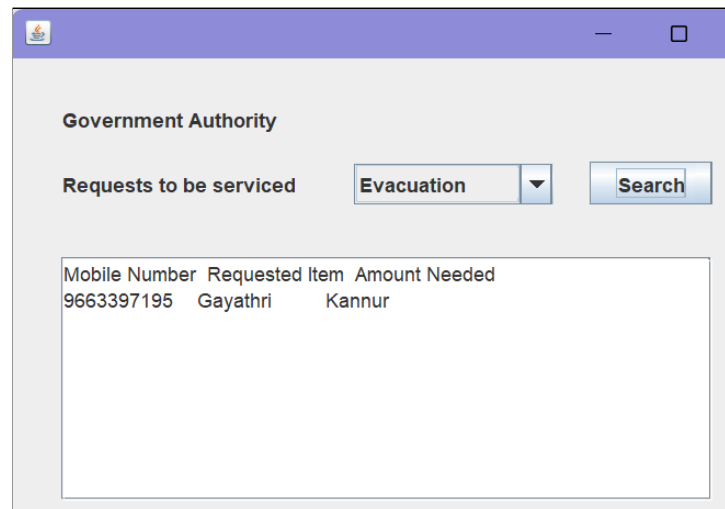


A screenshot of a "Success" message dialog box. The dialog has a purple header bar with the title "Success" and a close button. The main content area is light gray and contains the following text:

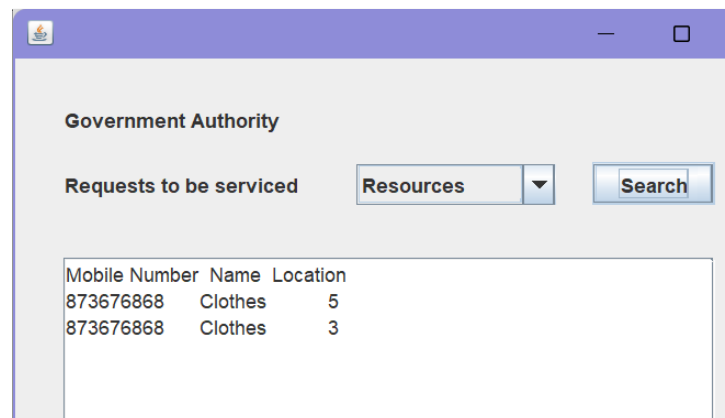
Thank you for donating, you can drop your package in person or courier it to us!

Below the text is an "OK" button.

Government relief teams can view the current requests that need to be processed based on category



Mobile Number	Requested Item	Amount Needed
9663397195	Gayathri	Kannur



Mobile Number	Name	Location
873676868	Clothes	5
873676868	Clothes	3

6. Team member contributions

Gayathri: Worked on the GUI and functionality for landing page and its menus, require assistance, government page and the volunteer donation and database methods for the same, documentation

Jahnavi: Worked on the postgres database and the java classes to connect to the database, create the necessary tables and execute queries, and on the NGO registration functionality, documentation

Jigisha: Worked on functionality for missing person registration, requesting and donating money, Medical page and on the database functionality for the same, documentation
