SAFE-AID : EMERGENCY RESOURCE LOCATOR

A Project Report for Industrial Training and Internship

submitted by

ANKITA DHAR

BHISHAL SIKDAR

In the partial fulfillment of the award of the degree of

B-TECH

in the

ECE , CSE

Of

CALCUTTA INSTITUTE OF ENGINEERING AND MANAGEMENT

INSPIRING SUCCESS

Ardent Computech Pvt. Ltd.

ARDENT® GROUP OF COMPANIES

ARDENT COMPUTECH PVT. LTD.

High-End Technology Training and Project

INTERNSHIP | DEVELOPMENT | PROJECT

An ISO 9001 : 2015 Certified Company

ARDENT
COMPUTECH PVT. LTD.
High-End Technology Training and Project

Ardent Computech Pvt. Ltd.
Drives you to the Industry
Module 132, SDF Building, Sector V, Salt Lake, Pin - 700091
www.ardentcollaborations.com

ISO 9001:2015 CERTIFIED

MSME

N·E·A·T
National Educational Alliance for Technology

CERTIFICATE FROM SUPERVISOR

This is to certify that
BHISHAL SIKDAR(16500123009) and **ANKITA DHAR(16500323002)**
have completed the project titled Technician Assignment Manager under my supervision during the period from 05/07/2025 to 19/08/2025 which is in partial fulfillment of requirements for the award of the B-TECH degree and submitted to the Department of **CSE** and **ECE** of **CALCUTTA INSTITUTE OF ENGINEERING AND MANAGEMENT**.

Signature of the Supervisor

Date: **19/08/2025**

Name of the Project Supervisor: Subhojit Santra

22 YEARS OF EXCELLENCE

ARDENT
COMPUTECH PVT. LTD.
High-End Technology Training and Project

Ardent Computech Pvt. Ltd.
Drives you to the Industry
Module 132, SDF Building, Sector V, Salt Lake, Pin - 700091
www.ardentcollaborations.com

ISO 9001:2015 CERTIFIED

N·E·A·T
National Educational Alliance for Technology

BONAFIDE CERTIFICATE

Certified that this project work was carried out under my supervision

Technician Assignment Manager is the bonafide work of

Name of the student: **BHISHAL SIKDAR** ***Signature:***

Name of the student: **ANKITA DHAR** ***Signature:***

SIGNATURE

Name:

PROJECT MENTOR

SIGNATURE

Name:

EXAMINERS

Ardent Original Seal

ARDENT
COMPUTECH PVT. LTD.
High-End Technology Training and Project

Ardent Computech Pvt. Ltd.
Drives you to the Industry
Module 132, SDF Building, Sector V, Salt Lake, Pin - 700091
www.ardentcollaborations.com

ISO 9001:2015

MSME

N·E·A·T
National Educational Alliance for Technology

ACKNOWLEDGEMENT

The achievement that is associated with the successful completion of any task would be incomplete without mentioning the names of those people whose endless cooperation made it possible. Their constant guidance and encouragement made all our efforts successful.

We take this opportunity to express our deep gratitude towards our project mentor, ***Subhojit Santra*** for giving such valuable suggestions, guidance and encouragement during the development of this project work.

Last but not the least we are grateful to all the faculty members of **Ardent Computech Pvt. Ltd.** for their support.

CONTENT PAGE

1. COMPANY PROFILE

ARDENT (Ardent Computech Pvt. Ltd.), formerly known as Ardent Computech Private Limited, is an ISO 9001:2015 certified Software Development and Training Company based in India. Operating independently since 2003, the organization has recently undergone a strategic merger with ARDENT Technologies, enhancing its global outreach and service offerings.

ARDENT Technologies
ARDENT Technologies delivers high-end IT services across the UK, USA, Canada, and India. Its core competencies lie in the development of customized application software, encompassing end-to-end solutions including system analysis, design, development, implementation, and training. The company also provides expert consultancy and electronic security solutions. Its clientele spans educational institutions, entertainment companies, resorts, theme parks, the service industry, telecom operators, media, and diverse business sectors.

ARDENT Collaborations
ARDENT Collaborations, the Research, Training, and Development division of ARDENT (Ardent Computech Pvt. Ltd.), offers professional IT-enabled services and industrial training programs. These are tailored for freshers and professionals from B.Tech, M.Tech, MBA, MCA, BCA, and MSc backgrounds. ARDENT (Ardent Computech Pvt. Ltd.) provides Summer Training, Winter Training, and Industrial Training to eligible candidates. High-performing students may qualify for stipends, scholarships, and additional benefits based on performance and mentor recommendations.

Associations and Accreditations
ARDENT (Ardent Computech Pvt. Ltd.) is affiliated with the National Council of Vocational Training (NCVT) under the Directorate General of Employment & Training (DGET), Ministry of Labour & Employment, Government of India. The institution upholds strict quality standards under ISO 9001:2015 certification and is dedicated to bridging the gap between academic knowledge and industry skills through innovative training programs.

2. INTRODUCTION

In today’s rapidly changing world, the frequency and intensity of natural disasters and crises have underscored the urgent need for effective emergency management solutions. With the rapid advancement of technology, digital platforms have emerged as powerful tools to enhance accessibility, coordination, and responsiveness during emergencies. This project, SafeAid, aims to create a comprehensive and interactive platform that caters to communities and individuals affected by crises, providing essential resources and real-time updates.

Powered by the MERN stack, our project SafeAid is designed to offer a user-friendly interface for users to access a wide range of emergency resources, submit and view community updates, and receive timely information. The platform also enables administrators to manage resources, oversee update submissions, and provide critical support. By leveraging dynamic content and smart features, the app ensures an efficient and reliable response experience.

This project not only bridges the gap between affected individuals and available aid but also promotes community resilience, making it ideal for people seeking to stay informed and contribute to safety efforts at their convenience.

2A.OBJECTIVE

The primary objective of SafeAid is to transform traditional emergency response systems by providing a flexible, accessible, and reliable digital platform that serves the diverse needs of communities and individuals affected by crises across various regions and demographics. The app aims to bridge the gap between available aid and those in need by offering real-time resource tracking, community update submissions, and secure communication tools that enhance coordination and safety.

Ultimately, SafeAid seeks to empower individuals and communities by equipping them with the tools and information they need to respond effectively to emergencies—anytime, anywhere.

2B.SCOPE

Our project focuses on creating a web-based application that supports a wide range of emergency management tools and community resources for individuals and communities affected by crises across various regions.

1. User Registration & Login:
   Secure sign-up and login for users and admins.

2. Resource Management:
   Interactive listing and search for emergency resources.

3. Community Updates:
   Tools for users to submit and view crisis updates.

4. Admin Panel:
   Controls for resource approval, user management, reports, and analytics.

5. Multiplatform Access:
   Available on web browsers.

6. Authentication:
   Support for secure access using JWT-based authentication.

3. SYSTEM ANALYSIS

3A.IDENTIFICATION OF NEED

System analysis is a crucial phase in the development of our project SafeAid, involving the creation of an efficient and user-friendly platform.

With the rapid increase in natural disasters and crises, coupled with the growing demand for accessible emergency management solutions, there is a pressing need for a flexible, efficient, and interactive platform that supports real-time resource coordination and communication. Traditional emergency response methods often face limitations in terms of location, timeliness, and resource availability. SafeAid addresses these challenges by providing a scalable digital solution for communities and administrators.

Key Needs Identified:

Accessibility to Resources:

Individuals in remote or disaster-affected areas need quick access to quality emergency resources.
Users require flexibility to access aid and updates at their own pace and during critical times.

Real-Time Updates:

Traditional communication often lacks dynamic tools to disseminate timely information.
There is a need for real-time updates, alerts, and community submissions to keep users informed and engaged.

Efficient Management:

Administrators need an efficient way to upload, manage, and distribute resource information.
Automation in resource tracking and reporting saves time and increases accuracy during crises.

3B.FEASIBILITY STUDY

The feasibility study of our SafeAid project indicates that the concept is viable and promising across several key areas.
The SafeAid project is feasible and practical across all key domains. Technically, it can be developed using widely available tools like the MERN stack (MongoDB, Express.js, React, Node.js), with scalable backend solutions to handle real-time data. Economically, it has strong potential through partnerships with NGOs or government agencies, and possible premium features for advanced analytics. Operationally, it is easy to use, scalable, and meets the critical needs of communities during emergencies. Legally, it is viable with compliance to data protection laws and emergency communication regulations. The estimated development time is 4 to 6 months, making it achievable within a reasonable timeframe and budget.

Overall, the SafeAid project is both realistic and promising, offering a sustainable and impactful solution in the growing field of emergency management technology.

3C.WORKFLOW

This Document plays a vital role in the development life cycle (SDLC) as it describes the complete requirements of the system. It is meant for use by the developers and will be the basic during the testing phase. Any changes made to the requirements in the future will have to go through a formal change approval process.

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

The waterfall model is the earliest SDLC approach that was used for software development. The waterfall Model illustrates the software development process in a linear sequential flow; hence it is also referred to as a linear-sequential life cycle model. This means that any phase in the development process begins only if the previous phase is complete. In the waterfall model phases do not overlap.

- **Waterfall Model Design:**

The waterfall approach was the first SDLC Model to be used widely in Software Engineering to ensure the success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In the Waterfall model, typically, the Outcome of one phase acts as the input for the next phase sequentially.

- **Iterative Waterfall Design:**

Definition: The Iterative Waterfall Model is a variation of the traditional Waterfall model, which is a linear and sequential software development methodology. In the Iterative Waterfall Model, the development process is divided into small, manageable cycles, allowing for the revisiting and refinement of phases before progressing to the next stage. It combines the systematic structure of the Waterfall model with the flexibility of iterative development.

The sequential phases in Iterative Waterfall model are:

➢ **Requirement Gathering and Analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.

➢ **System Design:** The requirement specifications from the first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.

➢ **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.

➢ **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing each unit. Post integration the entire system is tested for any faults and failures.

➢ **Deployment of the system:** Once the functional and non-functional testing is done, the product is deployed in the customer environment or released into the market.

➢ **Maintenance:** Some issues come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in progress and are seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for the previous phase and it is signed off, so the name "Iterative Waterfall Model". In this model, phases do not overlap.

- **<u>Advantages:</u>**

1 . **Flexibility:** Iterations permit adjustments based on feedback.

2 . **Early Delivery:** Partial systems can be delivered incrementally.

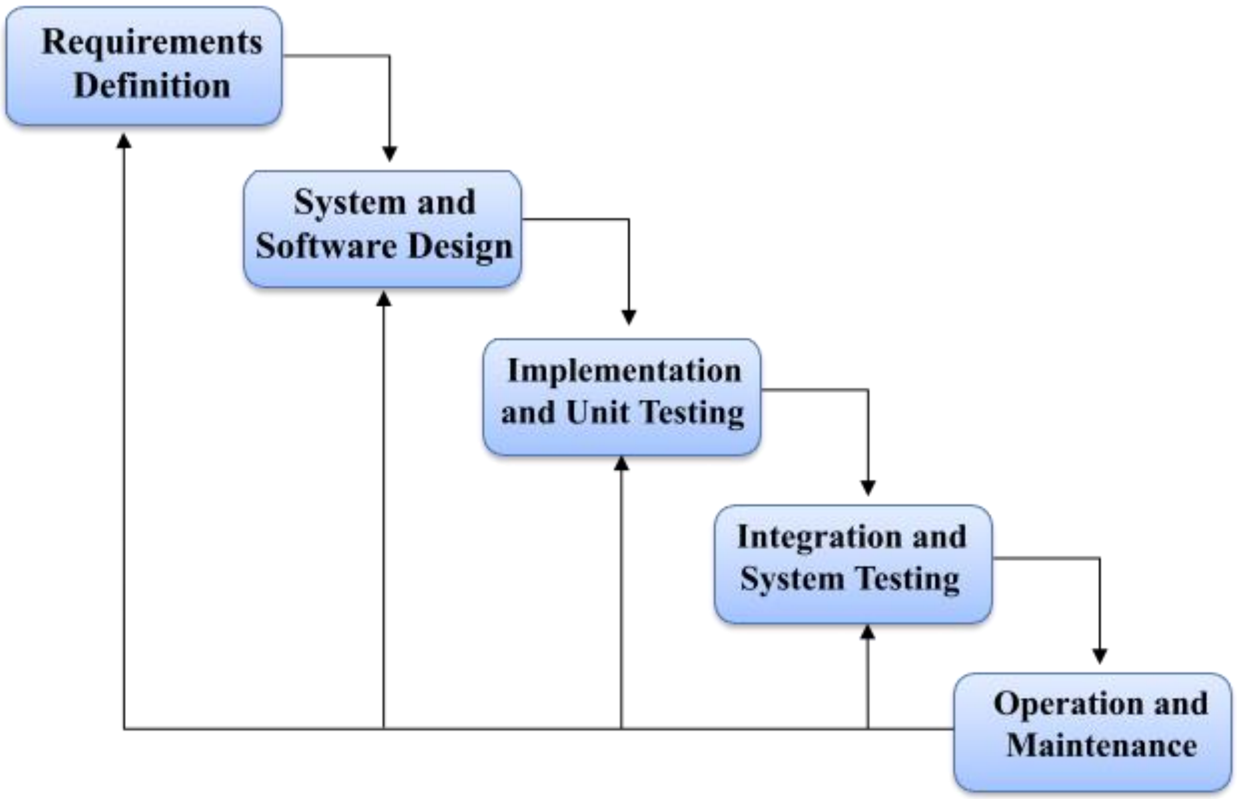
3 . **Risk Management:** Identifying and addressing issues early in the process.

- **<u>Disadvantages</u>:**

1. **Increased Complexity:** The iterative nature can make the process more complex.

2. **Potential for Scope Creep:** Frequent iterations may lead to scope changes.

3. **Resource Intensive:** Continuous revisiting of phases may demand more resources.


Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

- **<u>Applications</u>:**

The Iterative Waterfall Model is suitable for projects with evolving or unclear requirements. It is commonly used in software development projects where regular feedback and refinement are essential.

Additionally, it is applicable in scenarios where partial system delivery is beneficial, allowing stakeholders to assess progress and make adjustments.

3D .STUDY OF THE SYSTEM

Modules: The modules used in this software are as follows:

- **Register:**
    1. **User Register: Here, users will register to access emergency resources and submit updates.**
    2. **Admin Register: Here, admins will register to manage all system databases and resources.**

- **Verify Account: When a user logs in, an OTP is sent to their email to verify the validity of the provided data.**

- **Login:**
    1. **User Login: Here, users will log in to view available resources, submit updates, and access community information.**
    2. **Admin Login: Here, admins will log in to manage all data and system operations.**

- **Home: This page displays feedback received from users about resource availability and update usefulness.**

- **Resources:**
    1. **User Interface: This page shows available emergency resources that users can access, with a search feature to find specific items.**
    2. **Admin Interface: In this page, admins can view, add, or remove resources.**

- **Updates:**
    1. **User Interface: If users are logged in, they can submit and view crisis updates within the community.**
    2. **Admin Interface: Admins can approve or reject submitted updates for public display.**

- **About: This page will provide details about the SafeAid developers and the project's mission.**

- **Account:**
    1. **User Interface: a) My Profile: Here, users can view their details after logging in. b) Dashboard: Here, users can see their submitted updates and accessed resources.**

    2. **Admin Interface: a) Admin Profile: Here, admins can view their details after logging in. b) Admin Dashboard: Here, admins can manage resources, user roles, and system analytics.**

3E.INPUT AND OUTPUT

The main inputs, outputs, and major functions of the system are detailed as follows:

- **Input:**

    1. Users can log in by entering their credentials on the login page.
    2. Users can submit resource requests or community updates through the respective interfaces.
    3. Admins can input resource details and update approvals into the centralized system.

- **Output:**

    1. Users can view available emergency resources, access resource details, and see approved community updates.
    2. Admins can access a centralized database that includes user details, resource inventories, and update logs, ensuring efficient system management

3F.SOFTWARE REQUIREMENT SPECIFICATIONS

Software Requirements Specification provides an overview of the entire project. It is a description of a software system to be developed, laying out functional and non-functional requirements. The software requirements specification document enlists enough necessary requirements that are required for the project development. To derive the requirements we need to have a clear and thorough understanding of the project to be developed. This is prepared after detailed communication with the project team and the customer.

The developer is responsible for:

- Developing the system, which meets the SRS and solves all the requirements of the system.
- Demonstrating the system and installing the system at the client's location after acceptance testing is successful.
- Submitting the required user manual describing the system interfaces to work on it and also the documents of the system.

Functional Requirements:

A. **User Registration and Authentication:**
   1. Users should be able to create accounts securely.
   2. The system should authenticate users and manage login sessions.

B. **Browse and Search:**
   1. Users should be able to browse and search for courses.

.

C. **Courses Display:**
   1. Each Student should have detailed and up-to-date items with prices.
   2. Users should be able to view courses, helpful educational videos.

Hardware Requirements:

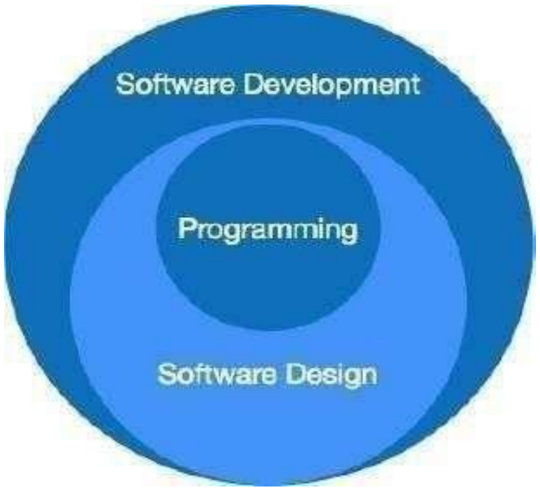
1 . Computer has Intel I3 Processor 2 .
8 GB RAM
3. SSD-ROM Drive

Software Requirements:

1. Windows 11 OS
2. Visual Studio Code
3. Mongo DB Atlas

3G.SOFTWARE ENGINEERING PARADIGM APPLIED

Software paradigms refer to the methods and steps, which are taken while designing the software. There are many methods proposed and are in work today, but we need to see where in software engineering these paradigms stand. These can be combined into various categories, though each of them is containedin one another.



The programming paradigm is a subset of Software design paradigm which is further a subset of the Software development paradigm.

There are two levels of reliability. The first is meeting the right requirements. A careful and thorough systems study is needed to satisfy this aspect of reliability. The second level of systems reliability involves the actual work delivered to the user. At this level, the system's reliability is interwoven with software engineering and development.

There are three approaches to reliability.

1. **Error avoidance:** Prevents errors from occurring in software.

2. **Error detection and correction:** In this approach, errors are recognized whenever they

are encountered, and correcting the error by the effect of the error of the system does not fail.

3. **Error tolerance:** In this approach, errors are recognized whenever they occur, but enables the system to keep running through degraded performance or Applying values that instruct the system to continue process.

4. SYSTEM DESIGN

4A. DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated.

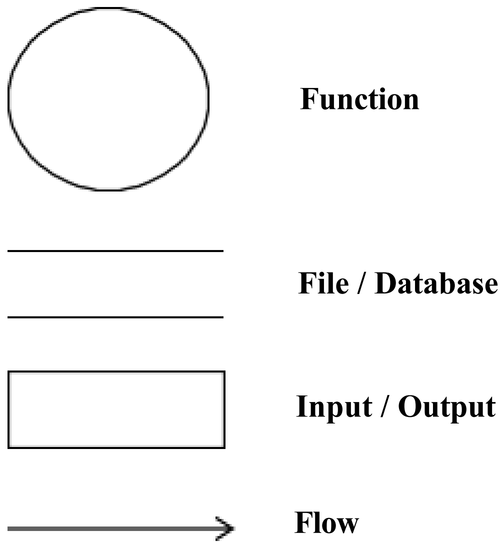
DFD can also be used for the visualization of data processing (structured design). A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of the process or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).

This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modeled. The Level 1 DFD shows how the system is divided into sub- systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present for the system to do its job and shows the flow of data between the various parts of the system.

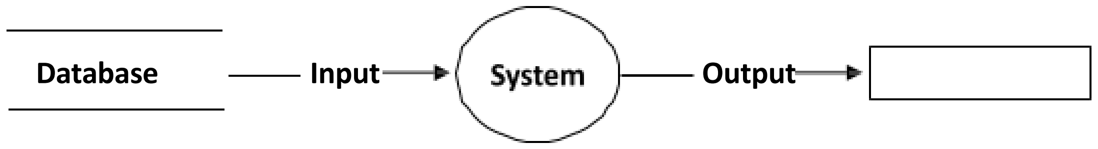
Data flow diagrams are one of the three essential perspectives of the structured- systems analysis and design method SSADM. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users can visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's data-flow diagrams can be drawn up and compared with.

How any system is developed can be determined through a data flow diagram model. In the course of developing a set of leveled data flow diagrams, the analyst/designer is forced to address how the system may be decomposed into component sub-systems and to identify the transaction data in the data model. Data flow diagrams can be used in both the Analysis and Design phase of the SDLC. There are different notations to draw data flow diagrams. Defining different visual representations for processes, data stores, data flow, and external entities.

DFD Notation:



DFD Example:



Steps to Construct Data Flow Diagram:

Four Steps are generally used to construct a DFD.

- Process should be named and referred for easy reference. Each name should be representative of the reference.
- The destination of flow is from top to bottom and from left to right.
- When a process is distributed into lower-level details they are numbered.
- The names of data stores, sources, and destinations are written in capital letters.

Rules for constructing a Data Flow Diagram:

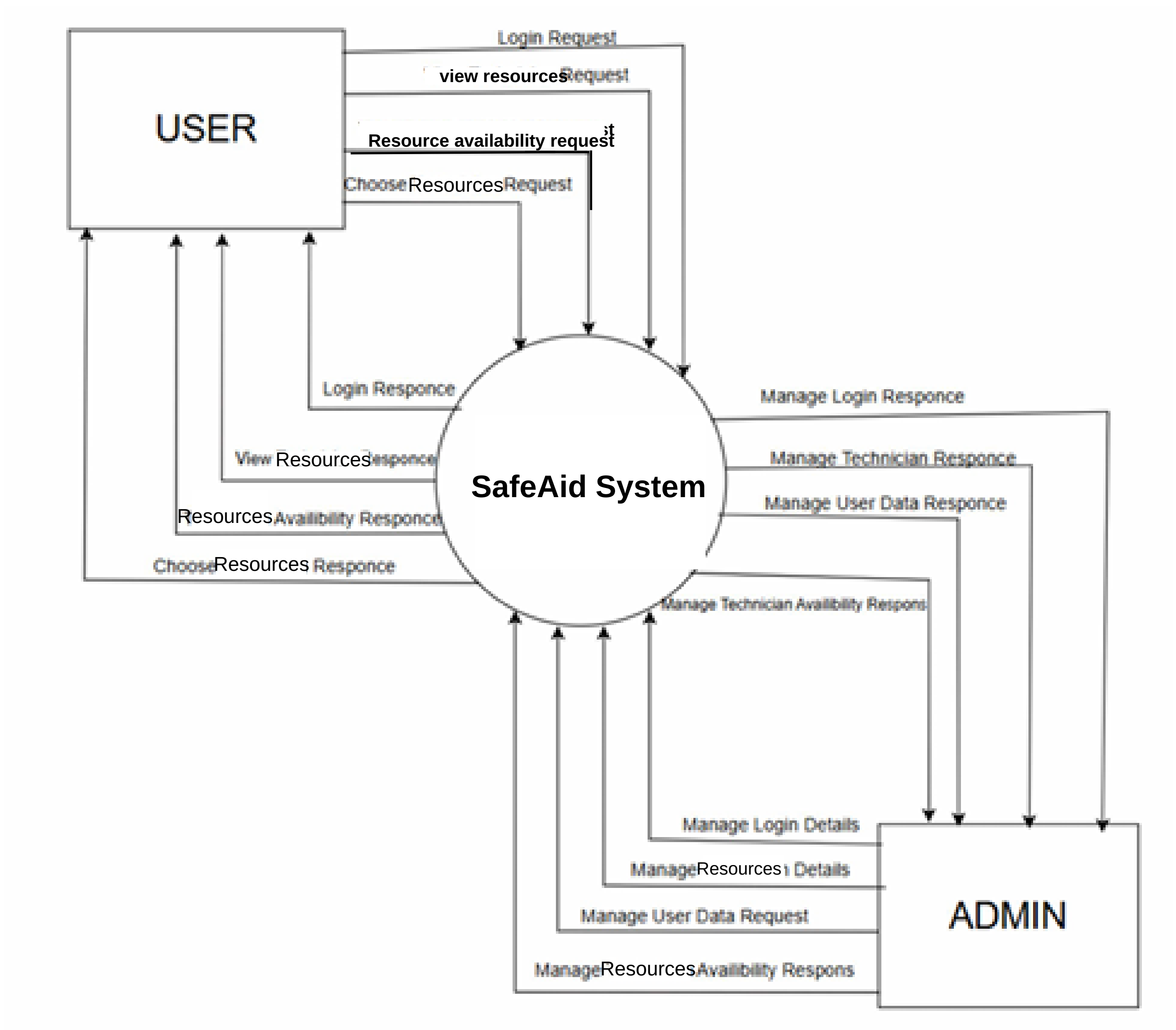
 Arrows should not cross each other.

 Squares, Circles, and Files must bear a name.

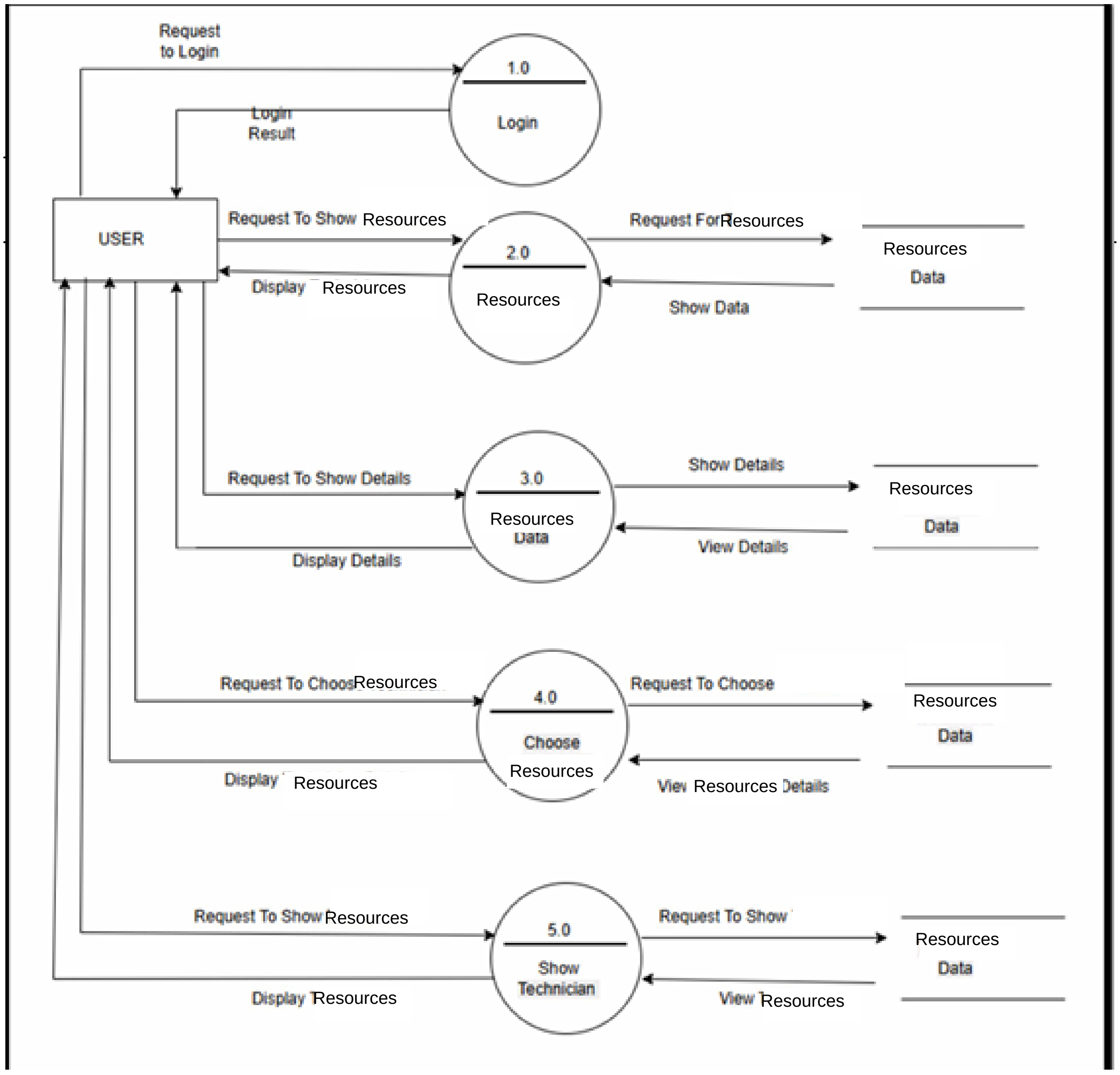
 Decomposed data flow squares and circles can have the same names.

 Draw all data flow around the outside of the diagram.

- **LEVEL 0 DFD OR CONTEXT DIAGRAM:**


USER
Login Request
view resources Request
Resource availability request
Choose Resources Request
Login Responce
View Resources Responce
Resources Availibility Responce
Choose Resources Responce
SafeAid System
Manage Login Responce
Manage Technician Responce
Manage User Data Responce
Manage Technician Availibility Respons
Manage Login Details
Manage Resources Details
Manage User Data Request
Manage Resources Availibity Respons
ADMIN

- **LEVEL 1 DFD:**


Request to Login
1.0
Login
Login Result
USER
Request To Show Resources
2.0
Resources
Display Resources
Request For Resources
Resources Data
Show Data
Request To Show Details
3.0
Resources Data
Display Details
Show Details
Resources Data
View Details
Request To Choose Resources
4.0
Choose Resources
Display Resources
Request To Choose
Resources Data
View Resources Details
Request To Show Resources
5.0
Show Technician
Display Resources
Request To Show
Resources Data
View Resources

4B.SEQUENCE DIAGRAM

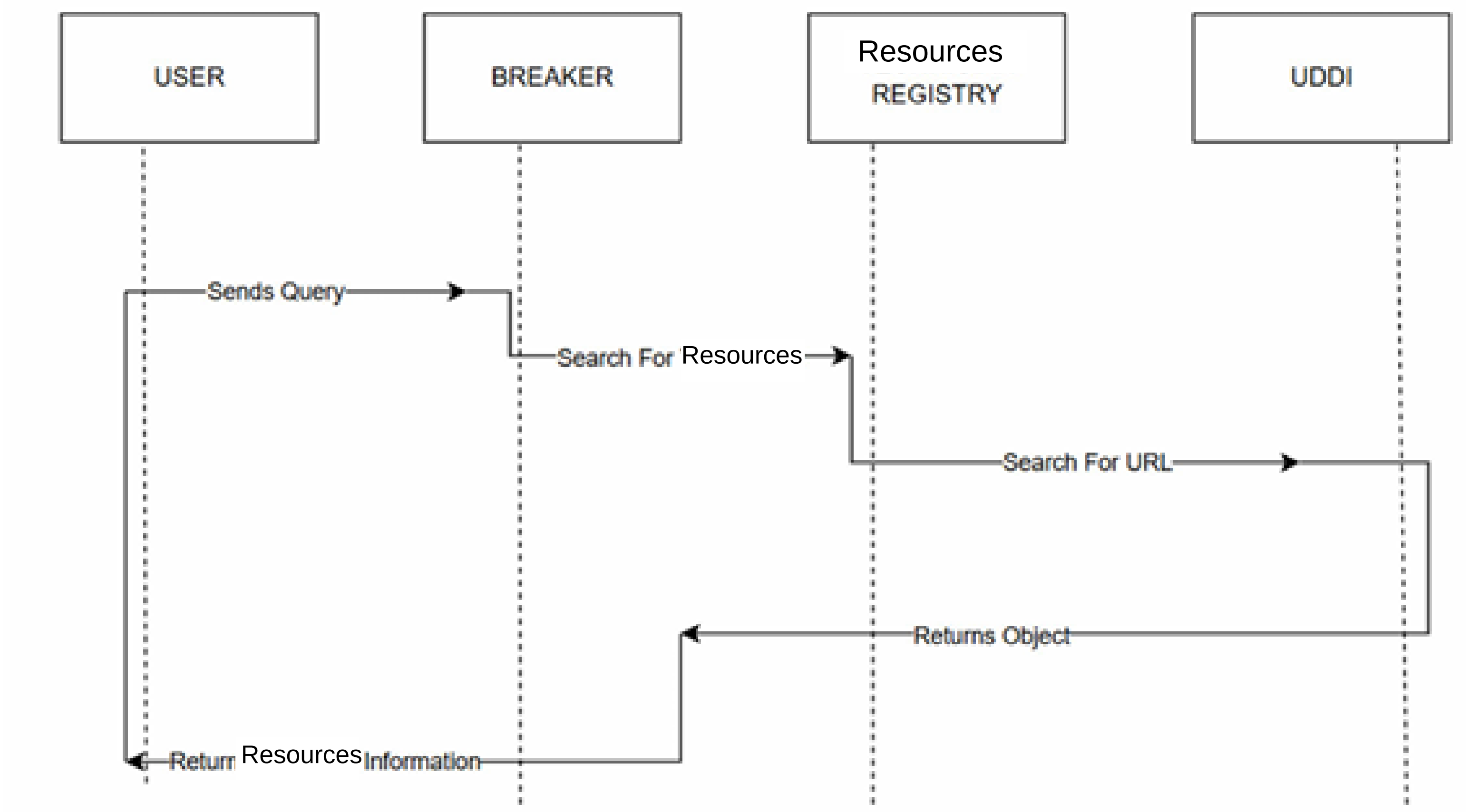
A Sequence diagram is an interaction diagram that shows how processes operate with one another and what is their order. It is a construct of a Message Sequence Chart. A sequence diagram shows object interactions arranged in a time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development.

Sequence diagrams are sometimes called event diagrams or event scenarios.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

A sequence diagram is the most common kind of interaction diagram, which focuses on the message interchange between several life lines .A sequence diagram describes an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines.

The following nodes and edges are typically drawn in a UML sequence diagram: lifeline, execution specification, message, fragment, interaction, state invariant, continuation, and destruction occurrence.



A Use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the use rand the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

So only static behavior is not sufficient to model a system rather dynamic behavior is more important than static behavior. In UML there are five diagrams available to model dynamic nature and a use case diagram is one of them. Now as we have to discuss that the use case diagram is dynamic in nature there should be some internal or external factors for making the interaction.

These internal and external agents are known as actors. So, use case diagrams consist of actors, use cases, and their relationships. The diagram is used to model the system/subsystem of an application. A single-use case diagram captures a particular functionality of a system.

So, to model the entire system numbers of use case diagrams are used. The purpose of a use case diagram is to capture the dynamic aspect of a system. But this definition is too generic to describe the purpose.

Because the other four diagrams (activity, sequence, collaboration, and State chart) are also having the same purpose. So, we will look into some specific purpose that will distinguish it from the other four diagrams.

Use case diagrams are used to gather the requirements of a system including internal and external influences. These requirements are mostly design requirements. So, when a system is analyzed to gather its functionalities use cases are prepared and actors are identified.

Now when the initial task is complete use case diagrams are modeled to present the outside view. So, in brief, the purposes of use case diagrams can be as follows:

- Used to gather requirements of a system.
- Used to get an outside view of a system.
- Identify external and internal factors influencing the system.

<u>How to draw Use Case Diagram?</u>

Use case diagrams are considered for high level requirement analysis of a system. So, when the requirements of a system are analyzed, the functionalities are captured in use cases.

So, we can say that uses cases are nothing but the system functionalities written in an organized manner. Now the second things which are relevant to the use cases are the actors. Actors can be defined as something that interacts with the system.

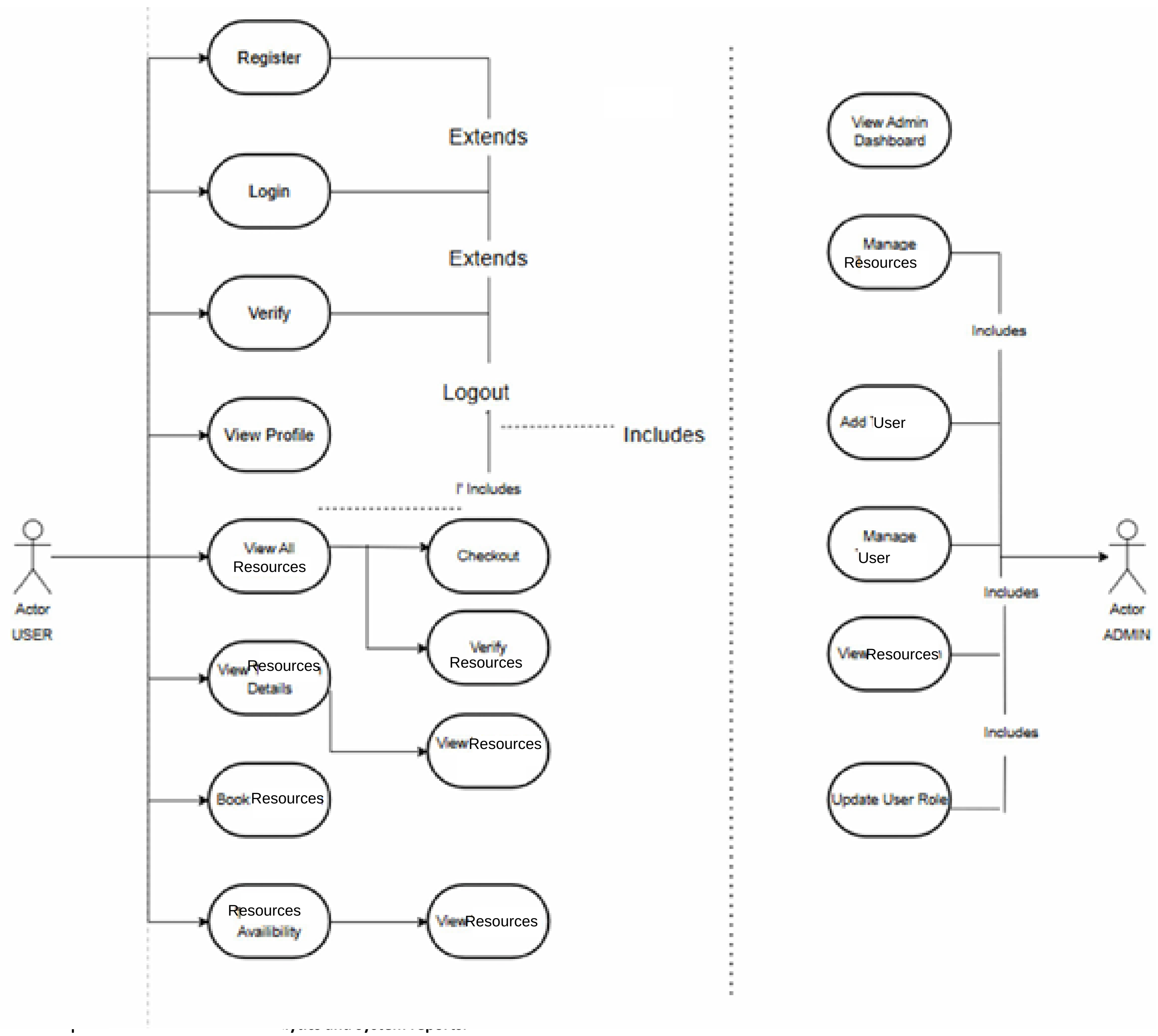
The actors can be human user, some internal applications or may be some external applications. So, in a brief when we are planning to draw use case diagram, we should have the following items identified.

- Functionalities to be represented as a use case
- Actors
- Relationships among the use cases and actors.

Use case diagrams are drawn to capture the functional requirements of a system. So, after identifying the above items we have to follow the following guidelines to draw an efficient use case diagram.

- The name of a use case is very important. So, the name should be chosen in such a way so that it can identify the functionalities performed.
- Give a suitable name for actors.
- Show relationships and dependencies clearly in the diagram.
- Do not try to include all types of relationships. Because the main purpose of the diagram is to identify requirements.
- Use note whenever required to clarify some important point

- **USE CASE DIAGRAM:**

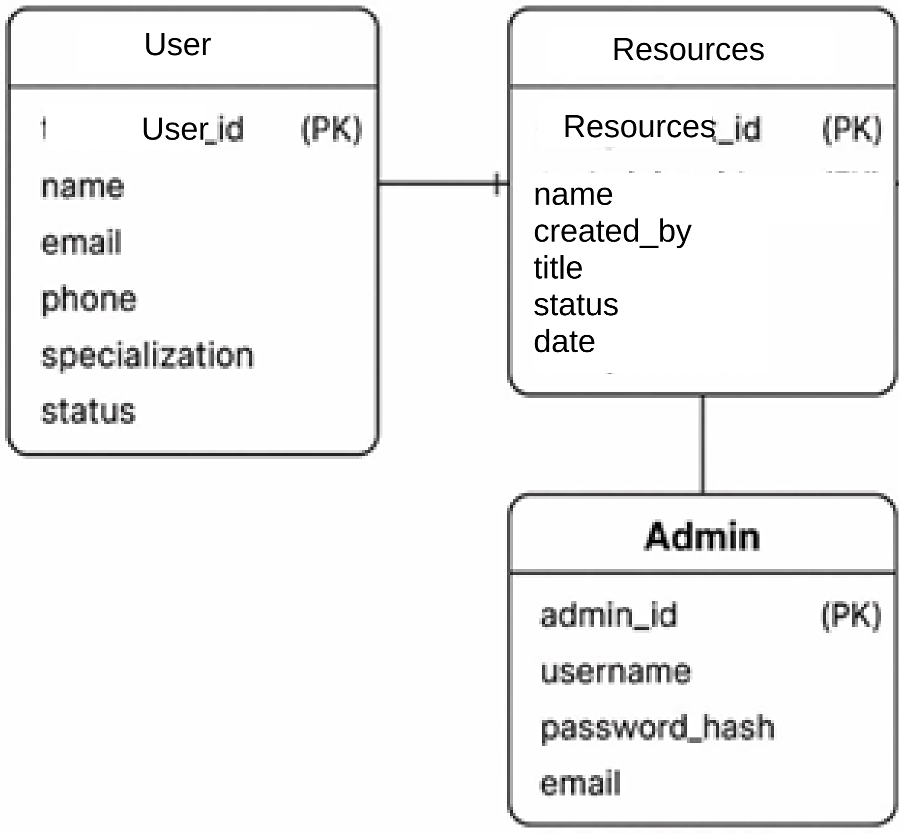


4D.SCHEMA DIAGRAM

The schema is an abstract structure or outline representing the logical view of the database as a whole. Defining categories of data and relationships between those categories, database schema design makes data much easier to retrieve, consume, manipulate, and interpret.

In DB schema design organizes data into separate entities, determines how to create relationships between organized entities, and influences the applications of constraints on data. Designers create database schema to give other database users, such as programmers and analysts, a logical understanding of data.

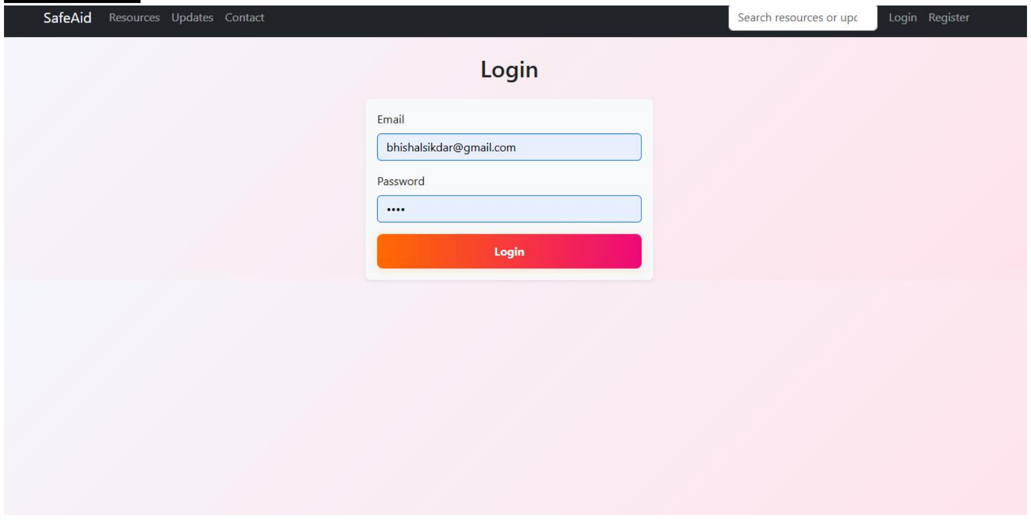
- **SCHEMA DESIGN:**



5. UI SNAPSHOT

❖ **FRONTEND : -**

1) Login Page:

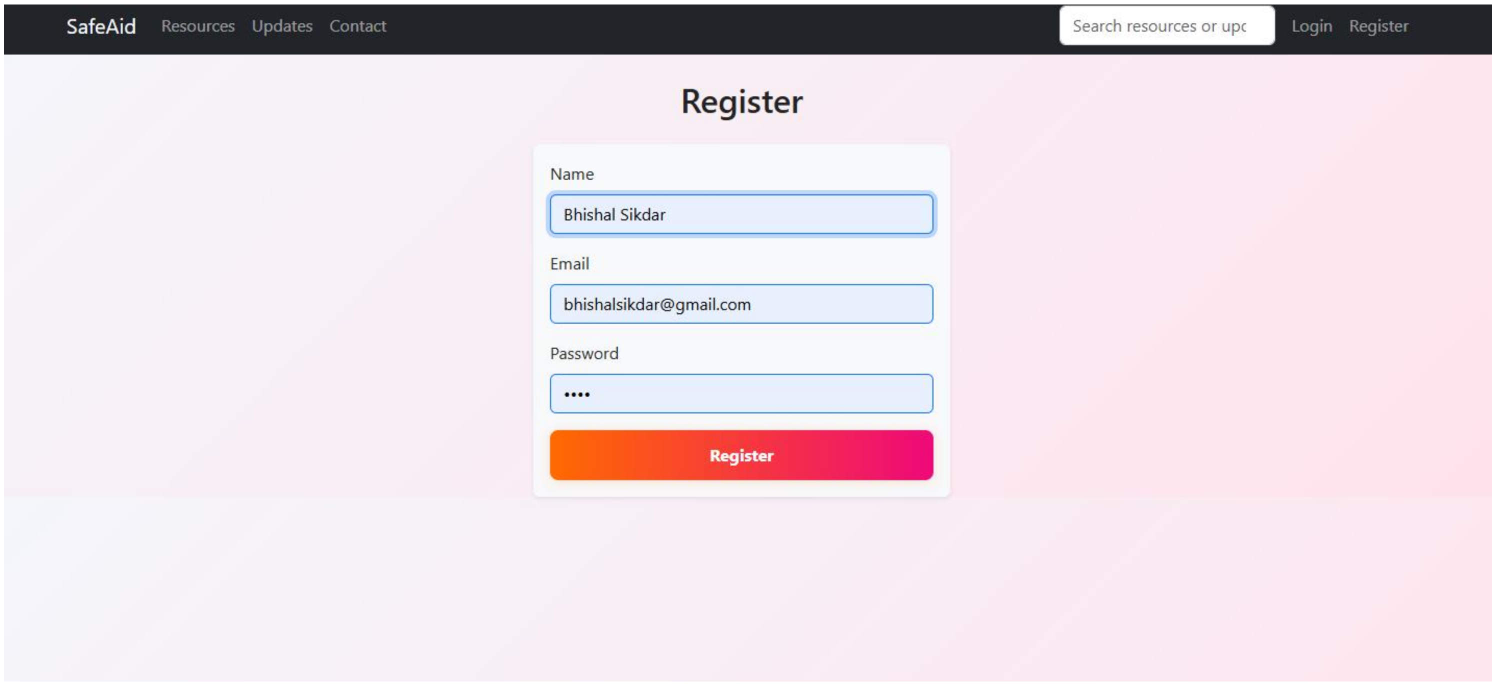


✓ CODE

```
  import React, { useState, useContext } from 'react';
import { Container, Form, Button, Alert } from 'react-bootstrap';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import { AuthContext } from '../AuthContext.jsx';

function Login() {
  const [credentials, setCredentials] = useState({ email: '', password: '' });
  const [message, setMessage] = useState(null);
  const navigate = useNavigate();
  const { login } = useContext(AuthContext);

  const handleInputChange = (e) => {
    setCredentials({ ...credentials, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setMessage('Logging in...');
    try {
      const response = await axios.post('http://localhost:5000/api/auth/login', credentials, {
        headers: { 'Content-Type': 'application/json' },
      });
      login(response.data.token);
      setMessage('Login successful!');
      setTimeout(() => navigate('/'), 1000);
  } catch (err) {
    setMessage('Login failed: ' + (err.response?.data?.message || 'Invalid credentials'));
  }
 };

 return (
  <Container className="mt-4 d-flex justify-content-center">
   <div className="w-100" style={{ maxWidth: '400px' }}>
    <h2 className="text-center mb-4">Login</h2>
    {message && <Alert variant={message.includes('successful') ? 'success' : 'danger'}>{message}</Alert>}
    <Form onSubmit={handleSubmit} className="p-3 bg-light rounded shadow-sm">
     <Form.Group controlId="formEmail" className="mb-3">
      <Form.Label>Email</Form.Label>
      <Form.Control
       type="email"
       name="email"
       value={credentials.email}
       onChange={handleInputChange}
       placeholder="Enter email"
       required
       className="border-primary"
      />
     </Form.Group>
     <Form.Group controlId="formPassword" className="mb-3">
      <Form.Label>Password</Form.Label>
      <Form.Control
       type="password"
       name="password"
       value={credentials.password}
       onChange={handleInputChange}
       placeholder="Enter password"
       required
       className="border-primary"
      />
     </Form.Group>
     <Button variant="primary" type="submit" className="w-100">
      Login
     </Button>
    </Form>
   </div>
  </Container>
 );
}

export default Login;
```

……….

Register Page:



Register

Name

Bhishal Sikdar

Email

bhishalsikdar@gmail.com

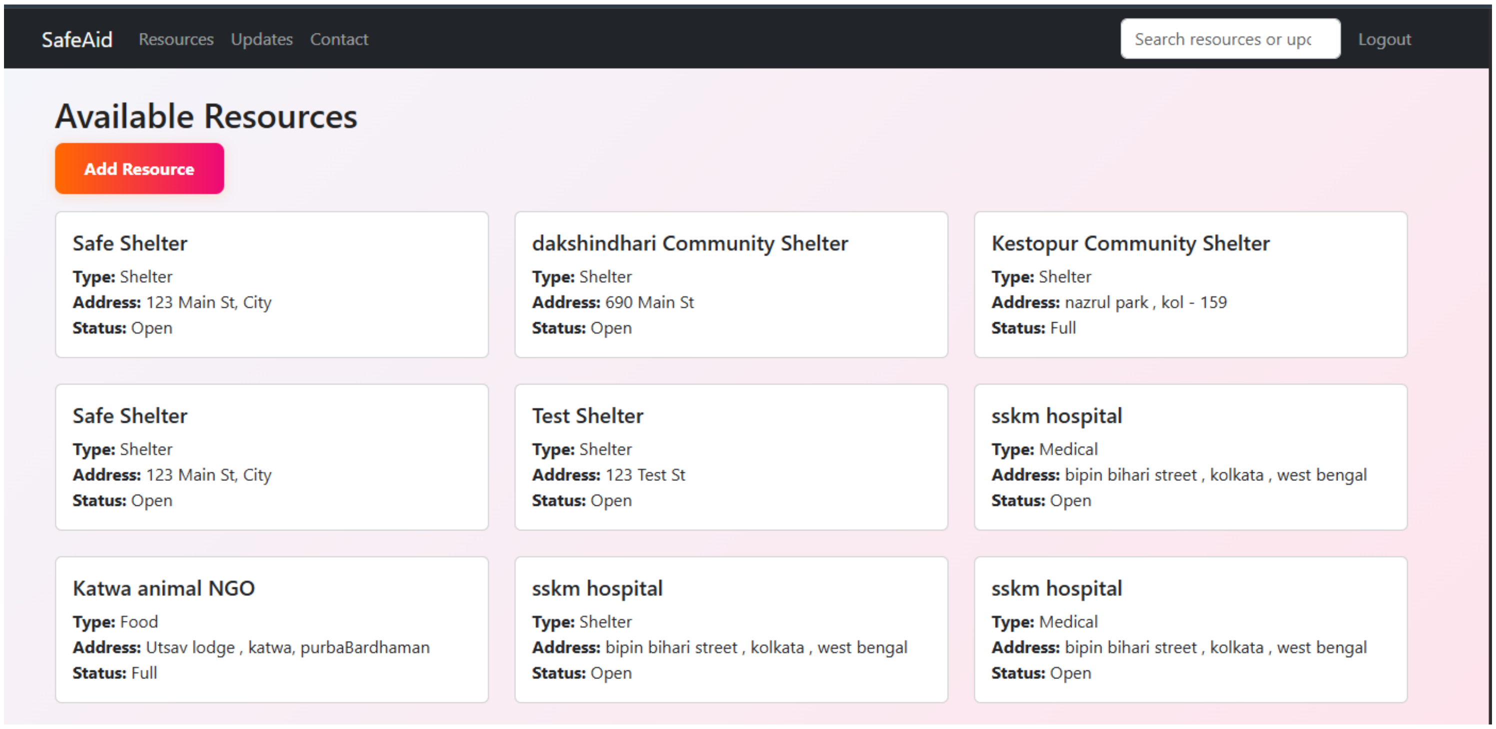
Password

••••

✓ **CODE**

```
import React, { useState, useContext } from 'react';
import { Container, Form, Button, Alert } from 'react-bootstrap';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import { AuthContext } from '../AuthContext.jsx';

function Register() {
  const [userData, setUserData] = useState({ name: '', email: '', password: '' });
  const [message, setMessage] = useState(null);
  const navigate = useNavigate();
  const { login } = useContext(AuthContext);

  const handleInputChange = (e) => {
    setUserData({ ...userData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setMessage('Registering...');
    try {
      const response = await axios.post('http://localhost:5000/api/auth/register', userData, {
        headers: { 'Content-Type': 'application/json' },
      });
      setMessage('Registration successful! Please log in.');
      setTimeout(() => navigate('/login'), 1000);
    } catch (err) {
      setMessage('Registration failed: ' + (err.response?.data?.message || 'Try again'));
    }
  };

  return (
    <Container className="mt-4 d-flex justify-content-center">
      <div className="w-100" style={{ maxWidth: '400px' }}>
        <h2 className="text-center mb-4">Register</h2>
        {message && <Alert variant={message.includes('successful') ? 'success' : 'danger'}>{message}</Alert>}
        <Form onSubmit={handleSubmit} className="p-3 bg-light rounded shadow-sm">
          <Form.Group controlId="formName" className="mb-3">
            <Form.Label>Name</Form.Label>
            <Form.Control
              type="text"
              name="name"
              value={userData.name}
              onChange={handleInputChange}
              placeholder="Enter name"
              required
              className="border-primary"
            />
          </Form.Group>
          <Form.Group controlId="formEmail" className="mb-3">
            <Form.Label>Email</Form.Label>
            <Form.Control
              type="email"
              name="email"
              value={userData.email}
              onChange={handleInputChange}
              placeholder="Enter email"
              required
              className="border-primary"
            />
          </Form.Group>
          <Form.Group controlId="formPassword" className="mb-3">
            <Form.Label>Password</Form.Label>
            <Form.Control
              type="password"
              name="password"
              value={userData.password}
              onChange={handleInputChange}
              placeholder="Enter password"
              required
              className="border-primary"
            />
          </Form.Group>
          <Button variant="primary" type="submit" className="w-100">
            Register
          </Button>
        </Form>
      </div>
    </Container>
  );
}
export default Register;
```

2) **RESOURCES PAGE(for admin and user):**


SafeAid Resources Updates Contact Search resources or upc Logout

Available Resources

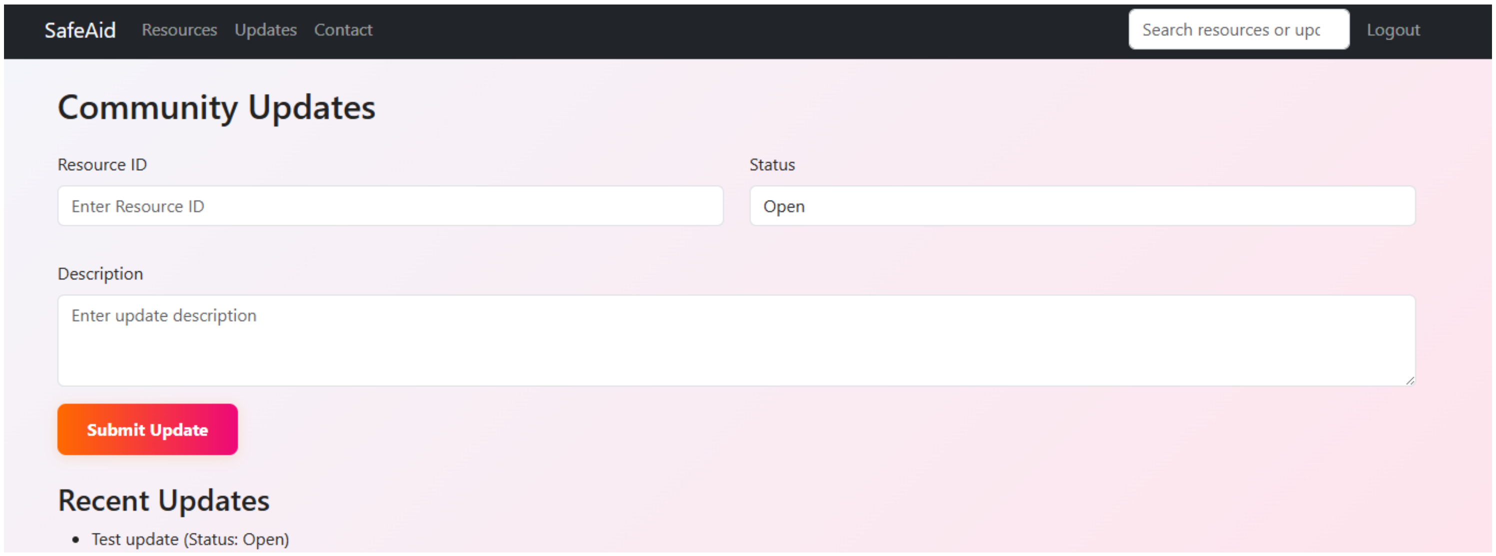
Add Resource

Safe Shelter **Type:** Shelter **Address:** 123 Main St, City **Status:** Open	dakshindhari Community Shelter **Type:** Shelter **Address:** 690 Main St **Status:** Open	Kestopur Community Shelter **Type:** Shelter **Address:** nazrul park , kol - 159 **Status:** Full
Safe Shelter **Type:** Shelter **Address:** 123 Main St, City **Status:** Open	Test Shelter **Type:** Shelter **Address:** 123 Test St **Status:** Open	sskm hospital **Type:** Medical **Address:** bipin bihari street , kolkata , west bengal **Status:** Open
Katwa animal NGO **Type:** Food **Address:** Utsav lodge , katwa, purbaBardhaman **Status:** Full	sskm hospital **Type:** Shelter **Address:** bipin bihari street , kolkata , west bengal **Status:** Open	sskm hospital **Type:** Medical **Address:** bipin bihari street , kolkata , west bengal **Status:** Open

- **Resources-pages-App.jsx**

```
import React, { useState } from 'react';
import { Container, Button } from 'react-bootstrap';
import { Route, Routes, Link, Navigate } from 'react-router-dom';
import ResourceList from './pages/ResourceList';
import Contact from './pages/Contact';
import Updates from './pages/Updates';
import NavbarComponent from './components/NavbarComponent';
import Login from './pages/Login';
import Register from './pages/Register';
import { AuthProvider, AuthContext } from './AuthContext.jsx'; // Added AuthContext to
import

function App() {
  const [searchTerm, setSearchTerm] = useState('');

  const handleSearch = (term) => {
    setSearchTerm(term);
  };

  return (
    <AuthProvider>
      <div>
        <NavbarComponent onSearch={handleSearch} />
        <Container className="mt-4">
          <Routes>

            <Route path="/" element={<Navigate to="/resources" />} />
            <Route path="/resources" element={<ResourceList searchTerm={searchTerm} />} />
            <Route
              path="/updates"
              element={
                <ProtectedRoute>
                  <Updates searchTerm={searchTerm} />
                </ProtectedRoute>
              }
            />
            <Route path="/contact" element={<Contact />} />
            <Route
              path="/login"
              element={
                <PublicRoute>
                  <Login />
                </PublicRoute>
              }
            />
            <Route
              path="/register"
              element={
                <PublicRoute>
                  <Register />
                </PublicRoute>
              }
            />
          </Routes>
        </Container>
      </div>
    </AuthProvider>
  );
}

// Protected Route Component
const ProtectedRoute = ({ children }) => {
  const { isAuthenticated } = React.useContext(AuthContext);
  return isAuthenticated ? children : <Navigate to="/login" />;
};

// Public Route Component
const PublicRoute = ({ children }) => {
  const { isAuthenticated } = React.useContext(AuthContext);
  return !isAuthenticated ? children : <Navigate to="/resources" />;
};

export default App;
```

3) UPDATES PAGE (for user and Admin);



Community Updates

Resource ID

Enter Resource ID

Status

Open

Description

Enter update description

Submit Update

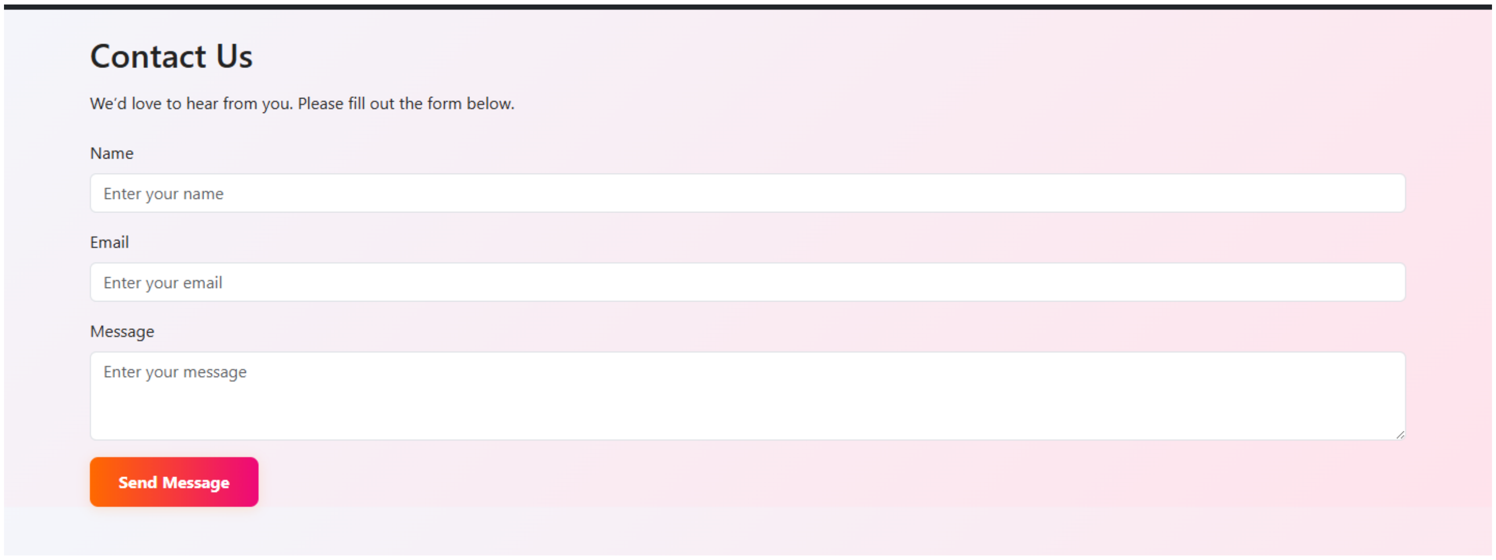
Recent Updates

- Test update (Status: Open)

✓ CODE

```
  import React, { useState, useEffect } from 'react';
import { Container, Button } from 'react-bootstrap';
import UpdateForm from '../components/UpdateForm';
import axios from 'axios';

function Updates({ searchTerm }) {
  const [updates, setUpdates] = useState([]);
  const [showForm, setShowForm] = useState(false);

  useEffect(() => {
    const fetchUpdates = async () => {
      try {
        const response = await axios.get('http://localhost:5000/api/updates');
        setUpdates(response.data);
      } catch (err) {
        console.error('Failed to fetch updates:', err);
      }
    };
    fetchUpdates();
  }, []);

  const handleFormSubmit = () => {
    setShowForm(false);
    // Refetch updates after submission
    const fetchUpdates = async () => {
      try {
        const response = await axios.get('http://localhost:5000/api/updates');
        setUpdates(response.data);
      } catch (err) {
        console.error('Failed to fetch updates:', err);
      }
    };
    fetchUpdates();
  };

  const filteredUpdates = updates.filter(update =>
    update.resourceId.toLowerCase().includes(searchTerm.toLowerCase()) ||
    update.status.toLowerCase().includes(searchTerm.toLowerCase()) ||
    update.description.toLowerCase().includes(searchTerm.toLowerCase())
  );

  return (
    <Container>
      <h2>Community Updates</h2>
      {!showForm && (
        <Button variant="primary" onClick={() => setShowForm(true)} className="mb-
3">
          Add Update
        </Button>
      )}
      {showForm && <UpdateForm onSubmitSuccess={handleFormSubmit} />}
      {filteredUpdates.length > 0 && (
        <div className="mt-4">
          <h3>Recent Updates</h3>
          <ul>
            {filteredUpdates.map((update) => (
              <li key={update._id}>
                {update.description} (Status: {update.status})
              </li>
            ))}
          </ul>
        </div>
      )}
    </Container>
  );
}

export default Updates;
```

5) Contact Page:


Contact Us

We'd love to hear from you. Please fill out the form below.

Name

Enter your name

Email

Enter your email

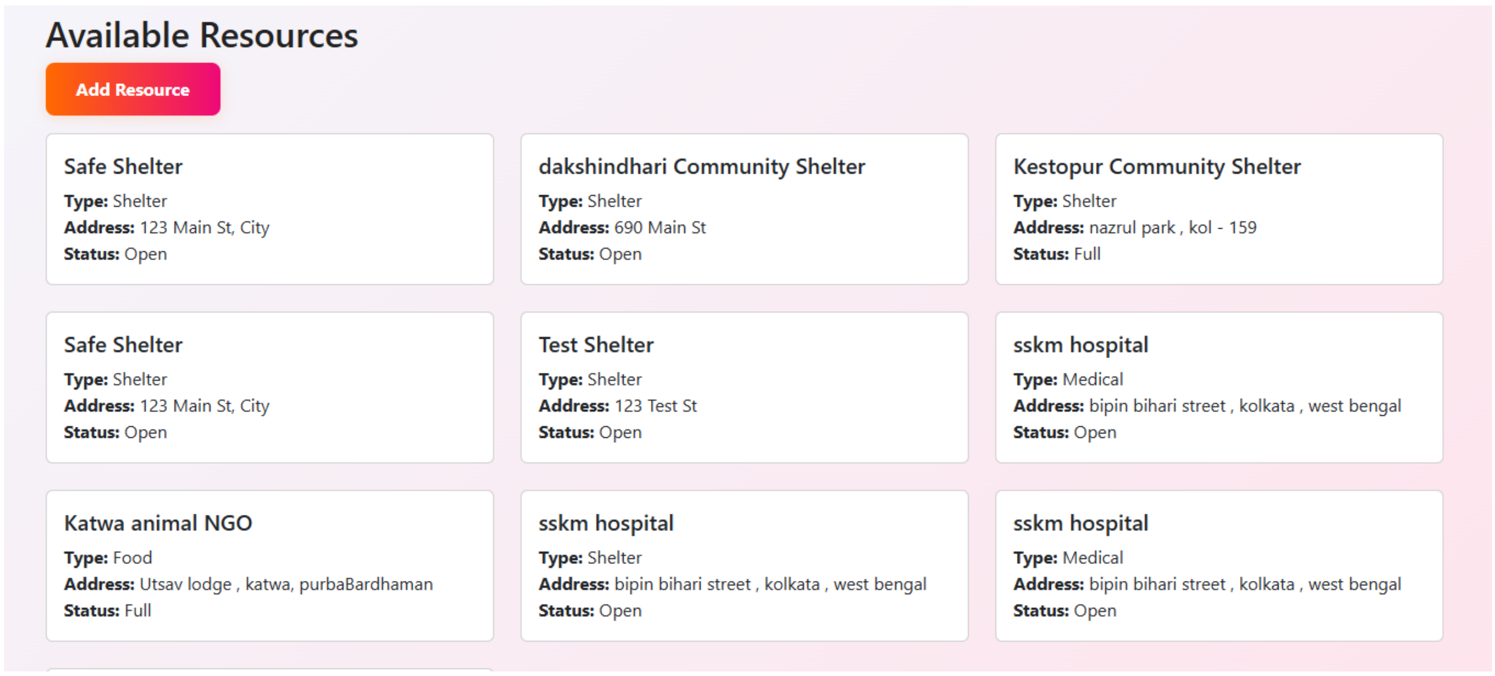
Message

Enter your message

✓ CODE

```
 import React, { useState } from 'react';
import { Container, Form, Button, Alert } from 'react-bootstrap';
import apiService from '../services/apiService';

function Contact() {
    console.log('Contact component loaded');

  const [formData, setFormData] = useState({
    name: '',
    email: '',
    message: '',
  });
  const [message, setMessage] = useState(null);

  const handleInputChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      console.log('Form submitted:', formData);
      setMessage('Thank you for your message! (Placeholder)');
      setFormData({ name: '', email: '', message: '' });
    } catch (err) {
      setMessage('Failed to send message');
    }
  };
  return (
    <Container>
      <h2>Contact Us</h2>
      <p className="mt-3">We’d love to hear from you. Please fill out the form below.</p>
      <Form onSubmit={handleSubmit} className="mt-4">
        <Form.Group controlId="formName" className="mb-3">
          <Form.Label>Name</Form.Label>
          <Form.Control
            type="text"
            name="name"
            value={formData.name}
            onChange={handleInputChange}
            placeholder="Enter your name"
            required
          />
        </Form.Group>
        <Form.Group controlId="formEmail" className="mb-3">
          <Form.Label>Email</Form.Label>
          <Form.Control
            type="email"
            name="email"
            value={formData.email}
            onChange={handleInputChange}
            placeholder="Enter your email"
            required
          />
        </Form.Group>
        <Form.Group controlId="formMessage" className="mb-3">
          <Form.Label>Message</Form.Label>
          <Form.Control
            as="textarea"
            name="message"
            value={formData.message}
            onChange={handleInputChange}
            placeholder="Enter your message"
            rows={3}
            required
          />
        </Form.Group>
        <Button variant="primary" type="submit">
          Send Message
        </Button>
        {message && <Alert variant={message.includes('Thank') ? 'success' : 'danger'}
className="mt-3">{message}</Alert>}
      </Form>
    </Container>
  );
}

export default Contact;
```

6) **<u>RESOURCES PAGE(for admin and user):</u>**


Available Resources

Safe Shelter
Type: Shelter
Address: 123 Main St, City
Status: Open

dakshindhari Community Shelter
Type: Shelter
Address: 690 Main St
Status: Open

Kestopur Community Shelter
Type: Shelter
Address: nazrul park , kol - 159
Status: Full

Safe Shelter
Type: Shelter
Address: 123 Main St, City
Status: Open

Test Shelter
Type: Shelter
Address: 123 Test St
Status: Open

sskm hospital
Type: Medical
Address: bipin bihari street , kolkata , west bengal
Status: Open

Katwa animal NGO
Type: Food
Address: Utsav lodge , katwa, purbaBardhaman
Status: Full

sskm hospital
Type: Shelter
Address: bipin bihari street , kolkata , west bengal
Status: Open

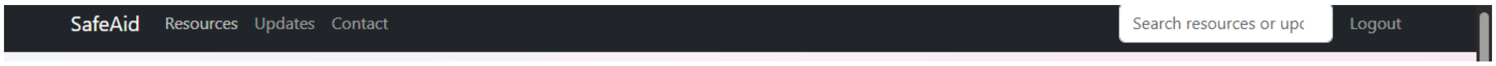
sskm hospital
Type: Medical
Address: bipin bihari street , kolkata , west bengal
Status: Open

✓ **<u>CODE:</u>**

```
   import React, { useEffect, useState } from 'react';
import { Container, Row, Col, Card, Spinner, Button, Form, Alert } from 'react-bootstrap';
import axios from 'axios';
import ResourceCard from '../components/resourceCard';
function ResourceList()
{
  const [resources, setResources] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [showForm, setShowForm] = useState(false);
  const [formData, setFormData] = useState({
    name: '',
    type: 'Shelter',
    address: '',
    status: 'Open',
    createdBy: 'tempUserId',
  });

  useEffect(() => {
    const fetchResources = async () => {
      try {

        const response = await axios.get('http://localhost:5000/api/resources');
        setResources(response.data);
        setLoading(false);
      } catch (err) {
        setError('Failed to fetch resources');
        setLoading(false);
      }
    };
    fetchResources();
  }, []);

  const handleInputChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post('http://localhost:5000/api/resources', formData);
      setResources([...resources, response.data]);
      setFormData({ name: '', type: 'Shelter', address: '', status: 'Open', createdBy:
'tempUserId' });
      setShowForm(false);
      setError(null);
    } catch (err) {
      setError('Failed to add resource');
    }
  };

  if (loading) return <Spinner animation="border" />;
  if (error && !showForm) return <Alert variant="danger">{error}</Alert>;

 return (
  <Container>
    <h2>Available Resources</h2>
    {!showForm && (
      <Button variant="primary" onClick={() => setShowForm(true)} className="mb-3">
        Add Resource
      </Button>
    )}
    {showForm && (
      <Form onSubmit={handleSubmit} className="mb-4">
        <Row>
          <Col xs={12} md={6}>
            <Form.Group controlId="formName">
              <Form.Label>Name</Form.Label>
              <Form.Control
                type="text"
                name="name"
                value={formData.name}
                onChange={handleInputChange}
                required
              />
            </Form.Group>
          </Col>
          <Col xs={12} md={6}>
            <Form.Group controlId="formType">
              <Form.Label>Type</Form.Label>
              <Form.Control
                as="select"
                name="type"
                value={formData.type}
                onChange={handleInputChange}
                required
              >
                <option value="Shelter">Shelter</option>
                <option value="Medical">Medical</option>
                <option value="Food">Food</option>
              </Form.Control>
            </Form.Group>
          </Col>
        </Row>
        <Row className="mt-3">
          <Col xs={12} md={6}>
            <Form.Group controlId="formAddress">
              <Form.Label>Address</Form.Label>
              <Form.Control
                type="text"
                name="address"
                value={formData.address}
                onChange={handleInputChange}
                required
              />
            </Form.Group>
          </Col>
          <Col xs={12} md={6}>
            <Form.Group controlId="formStatus">
              <Form.Label>Status</Form.Label>
              <Form.Control
                as="select"
                name="status"
                value={formData.status}
                onChange={handleInputChange}
                required
              >
                <option value="Open">Open</option>
                <option value="Full">Full</option>
                <option value="Closed">Closed</option>
              </Form.Control>
            </Form.Group>
          </Col>
        </Row>
        <Button variant="primary" type="submit" className="mt-3">
          Submit
        </Button>
        <Button variant="secondary" onClick={() => setShowForm(false)} className="mt-3 ms-
  2">
          Cancel
        </Button>
        {error && <Alert variant="danger" className="mt-3">{error}</Alert>}
      </Form>
    )}
    <Row>
```

```
      {resources.map((resource) => (
        <ResourceCard key={resource._id} resource={resource} />
      ))}
    </Row>
  </Container>
);
}

export default ResourceList;
```

7) NAVBAR PAGE:



✓ CODE:

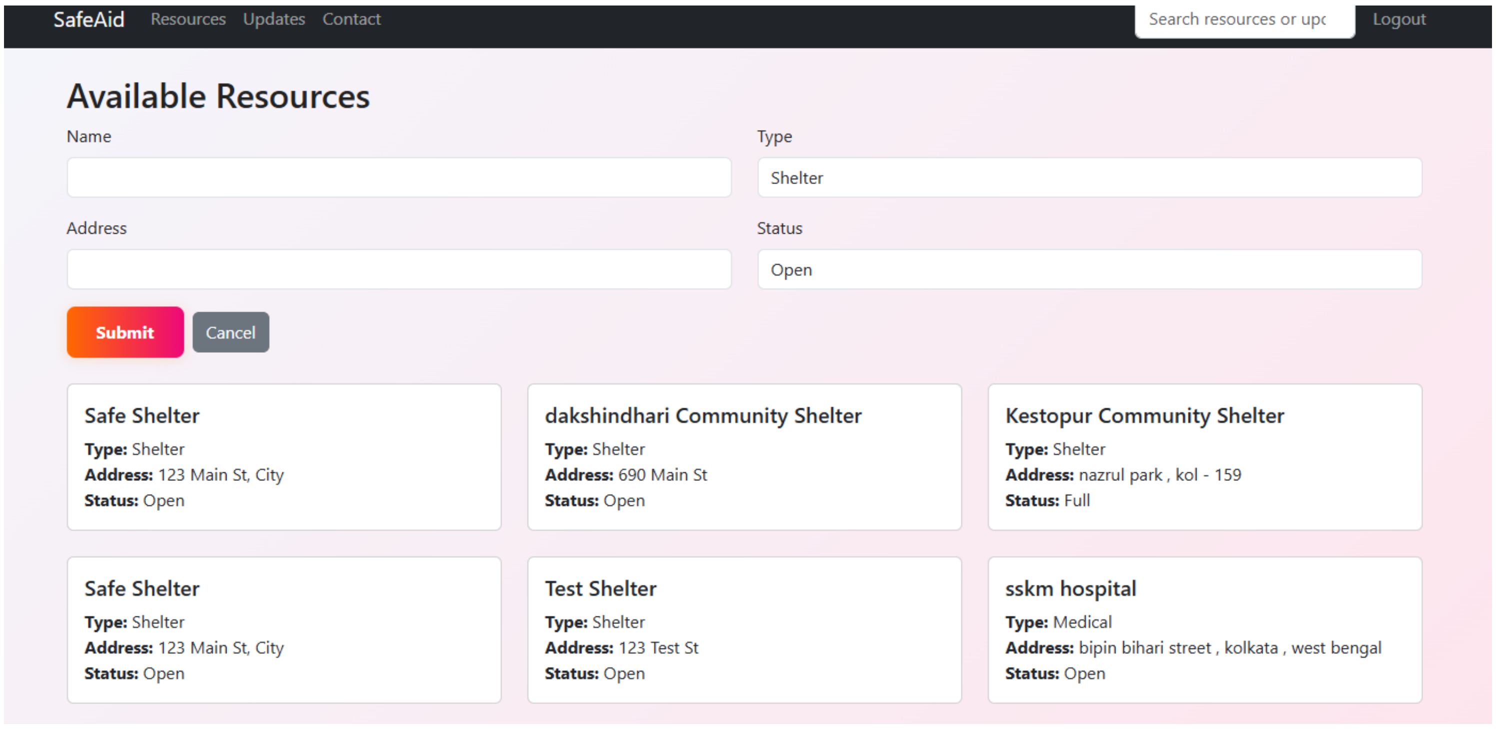
```
import React, { useState, useContext } from 'react';
import { Navbar, Nav, Container, Form, FormControl } from 'react-bootstrap';
import { Link } from 'react-router-dom';
import { AuthContext } from '../AuthContext.jsx';

const NavbarComponent = ({ onSearch }) => {
  const [searchTerm, setSearchTerm] = useState('');
  const { isAuthenticated, logout } = useContext(AuthContext);

  const handleSearch = (e) => {
    setSearchTerm(e.target.value);
    if (onSearch) onSearch(e.target.value);
  };

  const handleLogout = () => {
    logout();
  };

  return (
    <Navbar bg="dark" variant="dark" expand="lg">
      <Container>
        <Navbar.Brand as={Link} to="/">SafeAid</Navbar.Brand>
        <Navbar.Toggle aria-controls="basic-navbar-nav" />
        <Navbar.Collapse id="basic-navbar-nav">
          <Nav className="me-auto">
            <Nav.Link as={Link} to="/resources">Resources</Nav.Link>
            <Nav.Link as={Link} to="/updates">Updates</Nav.Link>
            <Nav.Link as={Link} to="/contact">Contact</Nav.Link>
          </Nav>
          <Form className="d-flex">
            <FormControl
              type="search"
              placeholder="Search resources or updates..."
              className="me-2"
              value={searchTerm}
              onChange={handleSearch}
              aria-label="Search"
            />
          </Form>
          <Nav>
            {isAuthenticated ? (
              <Nav.Link onClick={handleLogout}>Logout</Nav.Link>
            ) : (
              <>
                <Nav.Link as={Link} to="/login">Login</Nav.Link>
                <Nav.Link as={Link} to="/register">Register</Nav.Link>
              </>
            )}
          </Nav>
        </Navbar.Collapse>
      </Container>
    </Navbar>
  );
};

export default NavbarComponent;
```

```
const filterCourses = courses.filter((course) =>
  course.title.toLowerCase().includes(search.toLowerCase())
);

const changeImageHandler = (e) => {
  const file = e.target.files[0];
  const reader = new FileReader();
  reader.readAsDataURL(file);
  reader.onloadend = () => {
    setImagePrev(reader.result);
    setImage(file);
  };
};

const submitHandler = async (e) => {
  e.preventDefault();
  setBtnLoading(true);
  const myForm = new FormData();

  myForm.append("title", title);
  myForm.append("description", description);
  myForm.append("category", category);
  myForm.append("price", price);
  myForm.append("createdBy", createdBy);
  myForm.append("duration", duration);
  myForm.append("file", image);

  try {
    const { data } = await axios.post(
      `${server}/api/course/new`,
      myForm,
      {
        headers: {
          token: localStorage.getItem("token"),
        },
      }
    );

    toast.success(data.message);
    setBtnLoading(false);
    await fetchCourses();
    setTitle("");
    setDescription("");
    setCategory("");
    setPrice("");
    setCreatedBy("");
    setDuration("");
    setImage("");
    setImagePrev("");
    setShow(false);
    setShowSearch(true);
  } catch (error) {
    toast.error(error.response.data.message);
    setBtnLoading(false);
    }
  };

  return (
    <>
      <Layout>
        <div className="admin-courses">
             <div className="left">
               <h2>All Courses</h2>
            <div className="dashboard-content">
              {
                filterCourses && filterCourses.length > 0 ?
                  filterCourses.map((e) => (
                    <CourseCard key={e._id} course={e} />
                  ))
                  : <p>No Courses Yet</p>
              }
            </div>
          </div>

          <div className="right">
            <div className="add-course">
              <button className="add-btn m-3" onClick={() => { setShow(!show);
setShowSearch(!showSearch) }}>
                {show ? "Close" : "Add Course"}
              </button>

                                        { showSearch && (
                                            <input type="text" placeholder="Search
                                            courses
value={search}
                                            "

                                              onChange={(e) =>
                                        }     setSearch(e.target.value)}
                                            />
                                        { )



                                          show && (
                                            <div className="course-form">
                                              <h3>Add Course</h3>
                                              <form onSubmit={submitHandler}>
                                                <label htmlFor="title">Title</label>
                                                <input type="text" value={title}
                                                onChange={e =>
setTitle(e.target.value)} required />


                      <label htmlFor="description">Description</label>
                      <input type="text" value={description} onChange={e =>
setDescription(e.target.value)} required />
```

```
<label htmlFor="price">Price</label>
```

```
<select value={category} onChange={e => setCategory(e.target.value)}>
                              <option value={""}>Select Categories</option>
                              {
                                  categories.map((e) => (
                                    <option value={e} key={e}>{e}</option>
                                  ))
                                }
                            </select>

                            <label htmlFor="duration">Duration</label>
                            <input type="number" value={duration} onChange={e =>
      setDuration(e.target.value)} required />

                            <input type="file" onChange={changeImageHandler}
      required />

                            {
                              imagePrev && <img src={imagePrev} alt='' width={300}
      />
                            }

                            <button type="submit" disabled={btnLoading}
      className='common-btn'>{btnLoading ? "Please Wait..." : "Add"}</button>
                          </form>
                        </div>
                      )
                    }
                  </div>
                </div>
              </div>
            </Layout>
          </>
        )
      }

      export default AdminCourses;
```

9) ADMIN USERS PAGE:



Available Resources

Name	Type
	Shelter
Address	Status
	Open

Safe Shelter
Type: Shelter
Address: 123 Main St, City
Status: Open

dakshindhari Community Shelter
Type: Shelter
Address: 690 Main St
Status: Open

Kestopur Community Shelter
Type: Shelter
Address: nazrul park , kol - 159
Status: Full

Safe Shelter
Type: Shelter
Address: 123 Main St, City
Status: Open

Test Shelter
Type: Shelter
Address: 123 Test St
Status: Open

sskm hospital
Type: Medical
Address: bipin bihari street , kolkata , west bengal
Status: Open

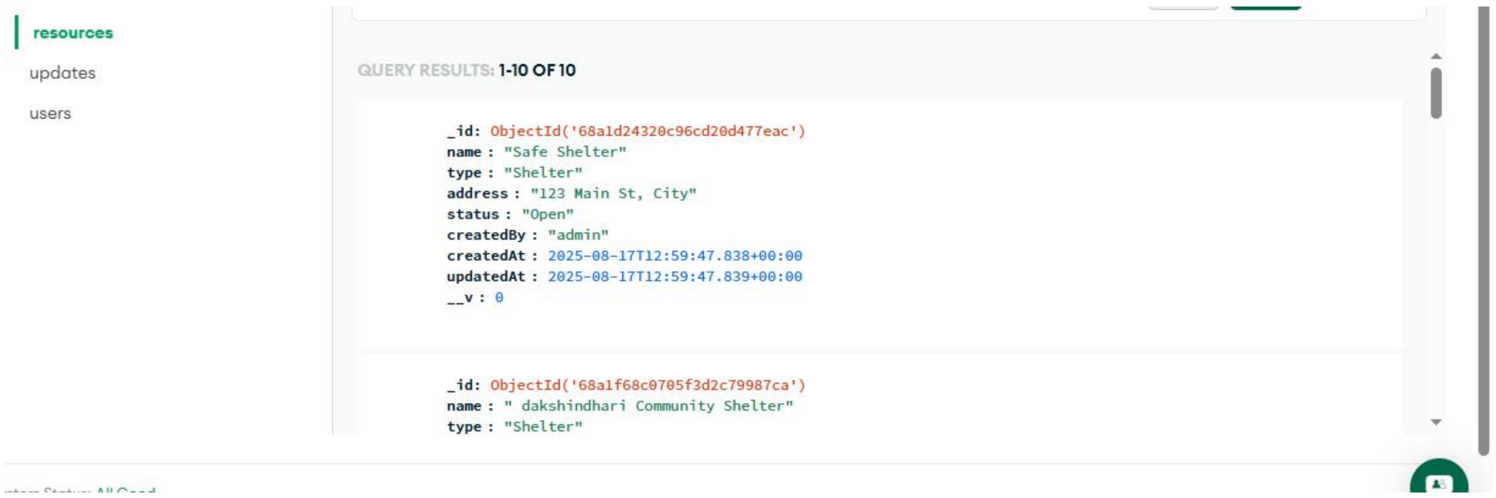
✓ CODE

```
import React, { useState } from 'react';
import { Container, Button } from 'react-bootstrap';
import { Route, Routes, Link, Navigate } from 'react-router-dom';
import ResourceList from './pages/ResourceList';
import Contact from './pages/Contact';
import Updates from './pages/Updates';
import NavbarComponent from './components/NavbarComponent';
import Login from './pages/Login';
import Register from './pages/Register';
import { AuthProvider, AuthContext } from './AuthContext.jsx'; // Added AuthContext to
import

function App() {
  const [searchTerm, setSearchTerm] = useState('');

  const handleSearch = (term) => {
    setSearchTerm(term);
  };

  return (
    <AuthProvider>
      <div>
```

```
        <NavbarComponent onSearch={handleSearch} />
        <Container className="mt-4">
          <Routes>

            <Route path="/" element={<Navigate to="/resources" />} />
            <Route path="/resources" element={<ResourceList searchTerm={searchTerm} />} />
            <Route
              path="/updates"
              element={
                <ProtectedRoute>
                  <Updates searchTerm={searchTerm} />
                </ProtectedRoute>
              }
            />
            <Route path="/contact" element={<Contact />} />
            <Route
              path="/login"
              element={
                <PublicRoute>
                  <Login />
                </PublicRoute>
              }
            />
            <Route
              path="/register"
              element={
                <PublicRoute>
                  <Register />
                </PublicRoute>
              }
            />
          </Routes>
        </Container>
      </div>
    </AuthProvider>
  );
}

// Protected Route Component
const ProtectedRoute = ({ children }) => {
  const { isAuthenticated } = React.useContext(AuthContext);
  return isAuthenticated ? children : <Navigate to="/login" />;
};

// Public Route Component
const PublicRoute = ({ children }) => {
  const { isAuthenticated } = React.useContext(AuthContext);
  return !isAuthenticated ? children : <Navigate to="/resources" />;
};

export default App
```

❖ BACKEND:-

1) USER AND ADMIN DATA:

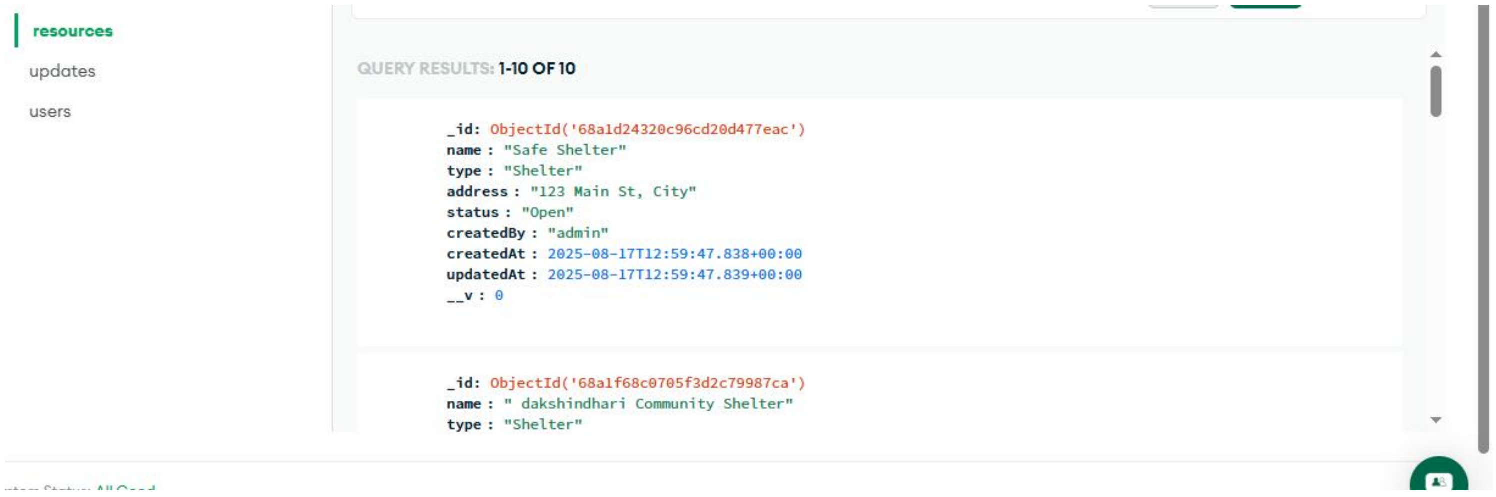


```
_id: ObjectId('68a1d24320c96cd20d477eac')
name : "Safe Shelter"
type : "Shelter"
address : "123 Main St, City"
status : "Open"
createdBy : "admin"
createdAt : 2025-08-17T12:59:47.838+00:00
updatedAt : 2025-08-17T12:59:47.839+00:00
__v : 0

_id: ObjectId('68a1f68c0705f3d2c79987ca')
name : " dakshindhari Community Shelter"
type : "Shelter"
```

- **<u>Database - db.js:</u>**

```
const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log(`MongoDB Connected: ${conn.connection.host}`);
  } catch (error) {
    console.error(`Error: ${error.message}`);
    process.exit(1);
  }
};

module.exports = connectDB;
```

2) RESOURCES DATA:

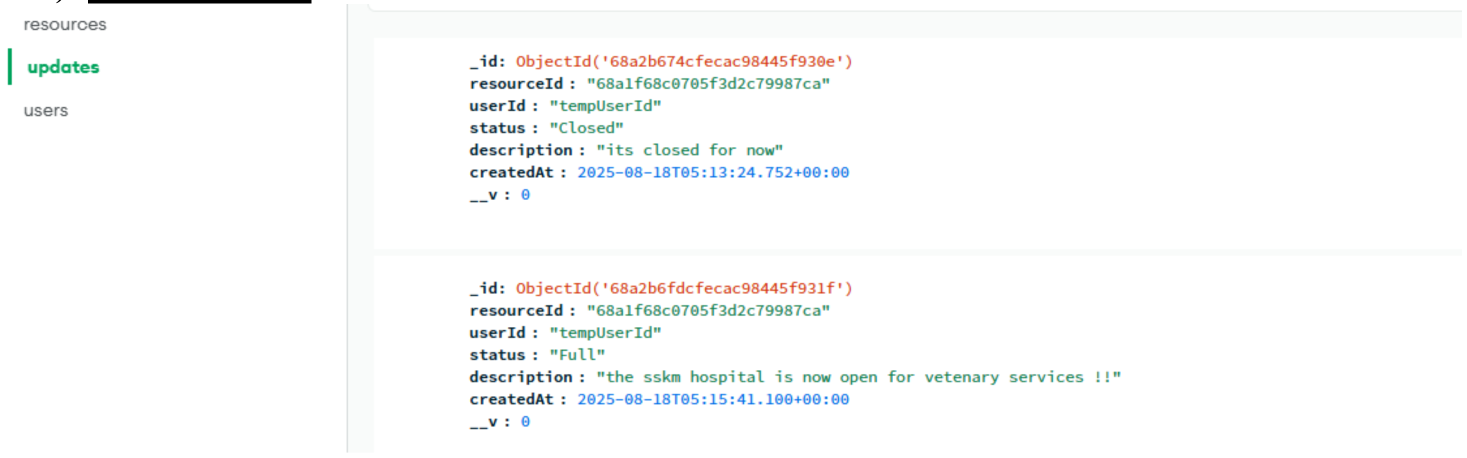


```
_id: ObjectId('68a1d24320c96cd20d477eac')
name : "Safe Shelter"
type : "Shelter"
address : "123 Main St, City"
status : "Open"
createdBy : "admin"
createdAt : 2025-08-17T12:59:47.838+00:00
updatedAt : 2025-08-17T12:59:47.839+00:00
__v : 0

_id: ObjectId('68a1f68c0705f3d2c79987ca')
name : " dakshindhari Community Shelter"
type : "Shelter"
```

- **model – Resources.js :-**

```
const mongoose = require('mongoose');

const resourceSchema = new mongoose.Schema({
  name: { type: String, required: true },
  type: { type: String, enum: ['Shelter', 'Medical', 'Food'], required: true },
  address: { type: String, required: true },
  status: { type: String, enum: ['Open', 'Full', 'Closed'], default: 'Open' },
  createdBy: { type: String, required: true }, // Temporary string placeholder
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now },
});

module.exports = mongoose.model('Resource', resourceSchema);
```

3) UPDATE DATA

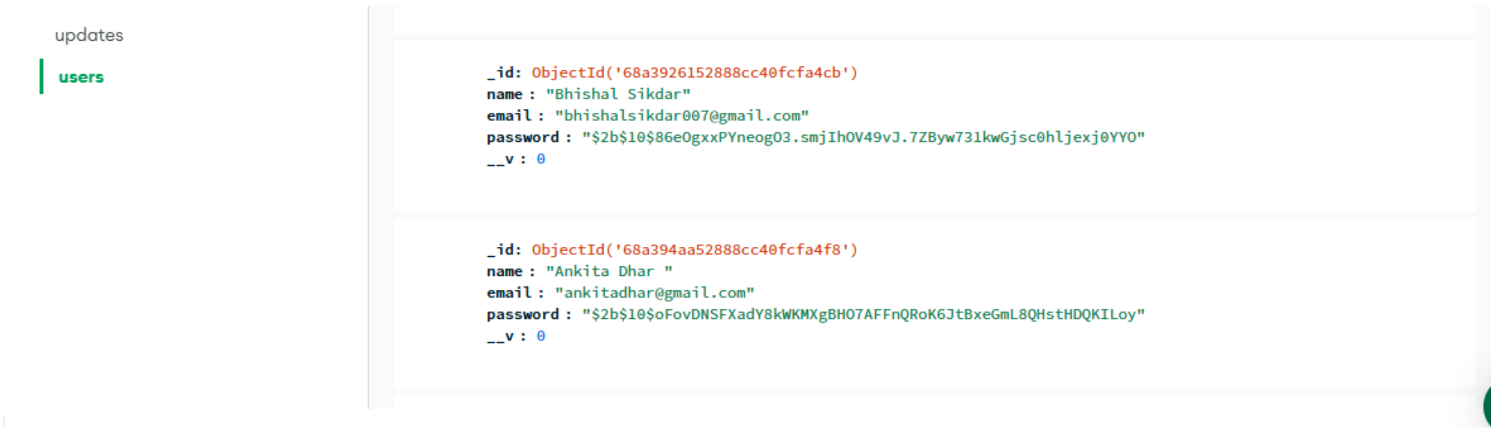


```
_id: ObjectId('68a2b674cfecac98445f930e')
resourceId : "68a1f68c0705f3d2c79987ca"
userId : "tempUserId"
status : "Closed"
description : "its closed for now"
createdAt : 2025-08-18T05:13:24.752+00:00
__v : 0

_id: ObjectId('68a2b6fdcfecac98445f931f')
resourceId : "68a1f68c0705f3d2c79987ca"
userId : "tempUserId"
status : "Full"
description : "the sskm hospital is now open for vetenary services !!"
createdAt : 2025-08-18T05:15:41.100+00:00
__v : 0
```

- **model – Update.js :-**

```
    const mongoose = require('mongoose');

const updateSchema = new mongoose.Schema({
  resourceId: { type: String, required: true },
  userId: { type: String, required: true }, // Placeholder for non-auth setup
  status: { type: String, enum: ['Open', 'Full', 'Closed'], required: true },
  description: { type: String, required: true },
  createdAt: { type: Date, default: Date.now },
});

module.exports = mongoose.model('Update', updateSchema);
```

4) USER DATA



```
_id: ObjectId('68a3926152888cc40fcfa4cb')
name : "Bhishal Sikdar"
email : "bhishalsikdar007@gmail.com"
password : "$2b$10$86eOgxxPYneogO3.smjIhOV49vJ.7ZByw731kwGjsc0hljexj0YYO"
__v : 0

_id: ObjectId('68a394aa52888cc40fcfa4f8')
name : "Ankita Dhar "
email : "ankitadhar@gmail.com"
password : "$2b$10$oFovDNSFXadY8kWKMXgBHO7AFFnQRoK6JtBxeGmL8QHstHDQKILoy"
__v : 0
```

- **model – User.js :-**

```
    const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
});

// Hash password before saving
userSchema.pre('save', async function (next) {
  if (this.isModified('password')) {
    this.password = await bcrypt.hash(this.password, 10);
  }
  next();
});

module.exports = mongoose.model('User', userSchema);
```

6. CONCLUSION

SafeAid serves as a powerful tool for enhancing emergency response by making resource access, coordination, and communication more accessible, efficient, and community-driven. By leveraging technology, it bridges the gap between affected individuals and available aid, enabling flexible and real-time support anytime, anywhere. The platform not only supports a wide range of emergency resources and updates but also incorporates features like user authentication, resource tracking, and admin oversight to enrich community resilience. As crisis management continues to evolve, this app stands as a promising solution to meet the growing needs of communities and administrators alike.

7. FUTURE SCOPE & FURTHER ENHANCEMENTS

❖ **Future scope:-**

The future of SafeAid is promising with numerous opportunities for growth and innovation. Key areas for development include:

- **AI-Powered Resource Matching**: Integrating artificial intelligence to offer personalized resource recommendations, predictive aid distribution, and adaptive response strategies based on crisis patterns and user needs.

- **Gamification**: Adding interactive and game-like features to boost community engagement and encourage timely update submissions.

- **AR/VR Integration**: Enhancing the response experience with augmented and virtual reality for immersive training simulations, especially in disaster preparedness and resource allocation.

- **Offline Accessibility**: Expanding features to allow resource access and update submission without an internet connection, ensuring support in remote or low-connectivity disaster zones.

- **Multilingual Support**: Supporting multiple languages to cater to a diverse, global user base during international crises.

- **Collaborative Response Tools**: Incorporating real-time collaboration features like emergency forums, coordinated relief efforts, and community feedback systems.

❖ <u>**Further enhancement**</u>**:-**

 1. User Experience Enhancements:

- **Personalized Dashboards**: Display resource access history, upcoming alerts, and tailored resource suggestions.
- **Gamification**: Add badges, leaderboards, or achievement systems to increase community participation.
- **Dark Mode & Accessibility Features**: Improve usability for all users, including those with visual impairments.

 2. Content & Response Tools:

- **Interactive Alerts**: Timed notifications, drag-and-drop resource mapping, and immediate feedback on submissions.
- **Update Notes & Transcripts**: Let users take notes alongside updates; include auto-generated summaries.
- **AI Assistants**: Use AI chatbots to answer questions or provide real-time crisis guidance.

 3. Community Features:

- **Live Coordination & Webinars**: Integrate with video conferencing tools for real-time emergency briefings.
- **Mentorship/Relief Groups**: Match users with mentors or form relief teams based on location and skills.

 4. Analytics & Progress Tracking:

- **Detailed Response Reports**: For users, admins, or agencies to monitor resource distribution and update impact.
- **Adaptive Response Paths**: Adjust resource recommendations and alert levels based on crisis progression.

 5. Tech Integrations:

- **Cloud Sync & Offline Mode**: Allow resource access and update submission anytime, even without internet.
- **Multilingual Support**: Expand reach globally to support diverse affected populations.
- **Integration with Emergency Portals**: Sync with government or NGO systems for seamless data sharing.

8. BIBLIOGRAPHY

1) www.w3schools.com

2)www.youtube.com

3)www.pexels.com

4)www.codepen.io

5)www.google.com

6)www.googlefont.com

7)www.react.our

8)www.codingworld.com