CRUD APPLICATION (SRS)(FullStack)

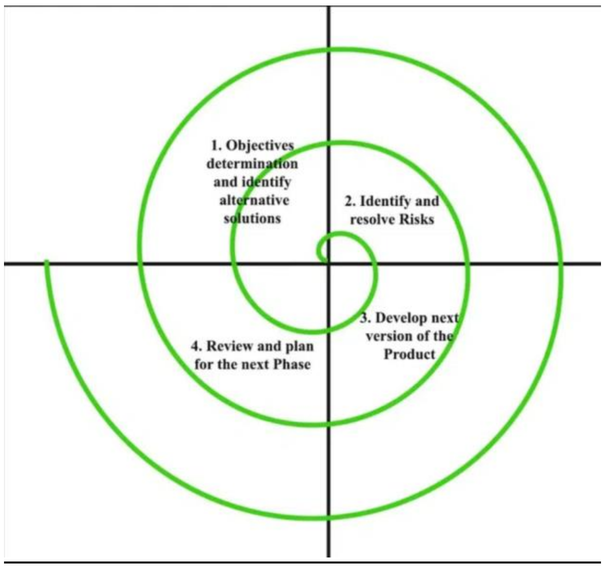
PROJECT SCOPUS:

CRUD APPLICATION-→ Create,Read,Update,Delete (USING SPRINGBOOT,REACT,MYSQL) (TO ADD: ADDITIONAL FEATURES)

SDLC IN USE:-

Since, it is a one person project, working with **spiral model** in mind is a better way.

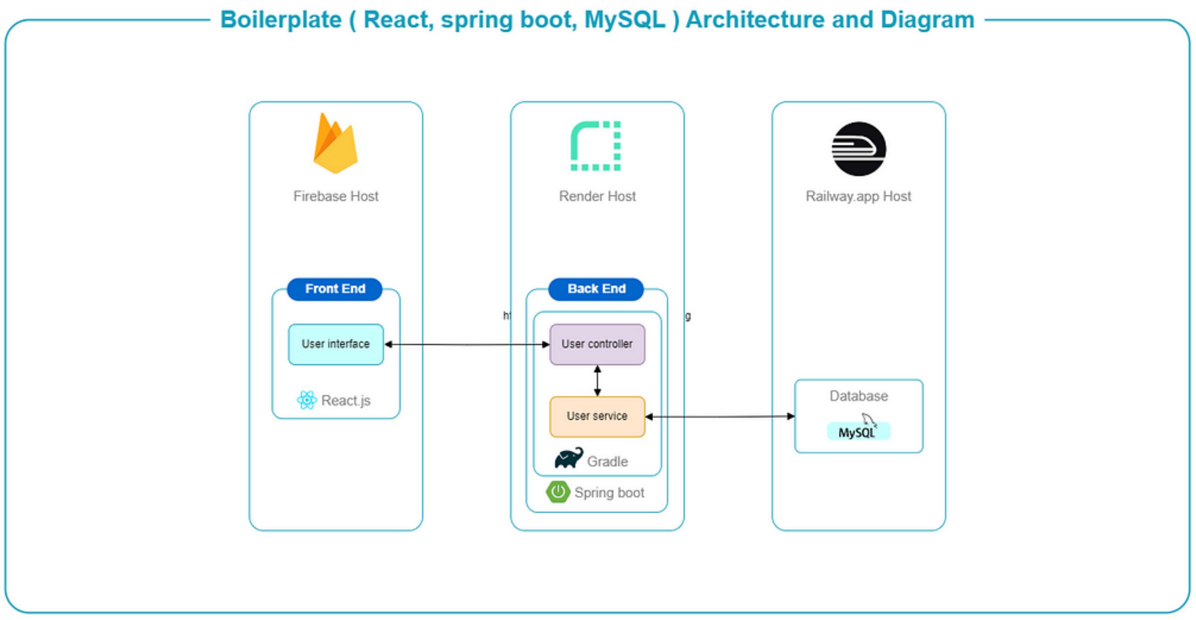
1. **Planning:** The first phase of the Spiral Model is the planning phase, where the scope of the project is determined and a plan is created for the next iteration of the spiral.(TECHNOLOGIES USED, STEP PROCESSES)
2. **Risk Analysis:** In the risk analysis phase, the risks associated with the project are identified and evaluated.(TESTING IN THIS CASE)
3. **Engineering:** In the engineering phase, the software is developed based on the requirements gathered in the previous iteration.
4. **Evaluation:** In the evaluation phase, the software is evaluated to determine if it meets the customer's requirements and if it is of high quality.
5. **Planning:** The next iteration of the spiral begins with a new planning phase, based on the results of the evaluation.


1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

Working On The Local Server

ARCHITECTURE:

- **REACT**
- **SPRINGBOOT**
- **MYSQL**



<u>Other Tools Required:</u>

- ❖ IntelliJ
- ❖ Postman
- ❖ Visual Studio Code
- ❖ Notepad++
- ❖ MySQL(Workbench)
- ❖ Git
- ❖ Spring Suite

SPIRAL MODELLING PLANNING->

(PHASES CREATED AS PLANNING IS EXECUTED)

PHASE 1:

- Creation of Basic Architecture of Backend, using REST API services(SPRINGBOOT MVC).
- Creation of Basic Frontend on React.js
- Creating Database to store data on MySQL
- Connecting React to Springboot and MySQL

- **TO-DO**

  1)HOME PAGE
  2) SUB-PAGES WITH INTERNAL LINKS FROM HOME PAGE
  3)UI CRITERIAN BOXES WITH EXTERNAL LINKS(PROJECTS LINKS)
  4)CREATE BACKEND WITH SPRINGBOOT AND REST-API **(most time taking)**
  5)Backend Storage in RDBMS TABLE in MySql
  6)Connecting each other to form a full stack network.
  7)Check for Security Risks and code factors! (Unit testing in phase-3)

PHASE 2:

1. Make the frontend look good!
   - ADD UI ELEMENTAL LINKS
   - ADD HREFS(Internal References) For The same page and for External Links
   - Create Externally Linked pages, use html and react Js (Atleast 2-4)

2. Add API connectors for creating the chatbot in SpringBoot.
3. Design the chatbot using DialogFlowES OR use Python for creation of a chatbot(Like Dottie{Not so Optimized})
4. Create a data storing requirement using login page or 'connect with me' page.
5. Store the data in table in SQL relational database systems.

PHASE 3:

1)Test Springboot backend for redundancies and optimizing.

2)Add test cases for development testing and prepare for deployment.

3)Cleaning the Code

PHASE 4: (DEPLOY ON GITHUB)

DEPLOYMENT(MEDIUM) (Deploy React+Spring Boot+MySql App for Free! Step-by-Step Guide | by Mariya Abdul Ghafoor | Towards Dev)

Building a Full-Stack Web App with Spring Boot and React

1. Prerequisites

- **Node.js**: Download and install Node.js from the official website.
- **Java Development Kit (JDK)**: Install at least JDK 8.
- **IntelliJ IDEA**: Set up IntelliJ IDEA for Spring Boot app development.
- **Visual Studio Code (VS Code)**: Use VS Code for React app development.

2. Benefits of Using Spring Boot with React

Spring Boot React offer several advantages when building full-stack web applications:

1. **High Performance and Scalability**:
   - Spring Boot provides a lightweight container for deploying and running applications.
   - React excels at rendering complex user interfaces efficiently.
2. **Robust Backend**:
   - Spring Boot is ideal for developing enterprise-level applications.
   - It offers a powerful and scalable backend for building APIs and microservices.
   - Supports various data sources and and easy integration with other projects.
3. **Efficient Frontend Development**:
   - React simplifies frontend development with its component-based architecture.
   - Code reusability leads to faster development and easier maintenance.
4. **Easy Integration**:
   - React can consume RESTful APIs from a Spring Boot backend using libraries like Axios, fetch, and superagent.
5. **Large Community Support**:
   - Both Spring Boot and React have active developer communities.

3. Architecture Overview

We'll create a full-stack application with the following architecture:

1. **Spring Boot Backend**:
   - Develop a backend application using Spring Boot.
   - Configure DB (any database).
   - Create JPA entities and repositories.
   - Implement RESTful APIs for CRUD operations.
2. **React Frontend**:
   - Create a React app using `create-react-app`.
   - Add Bootstrap or any other styling library.
   - Build React components (e.g., list view, form, etc.).
   - Use Axios or other HTTP libraries to consume the Spring Boot APIs.

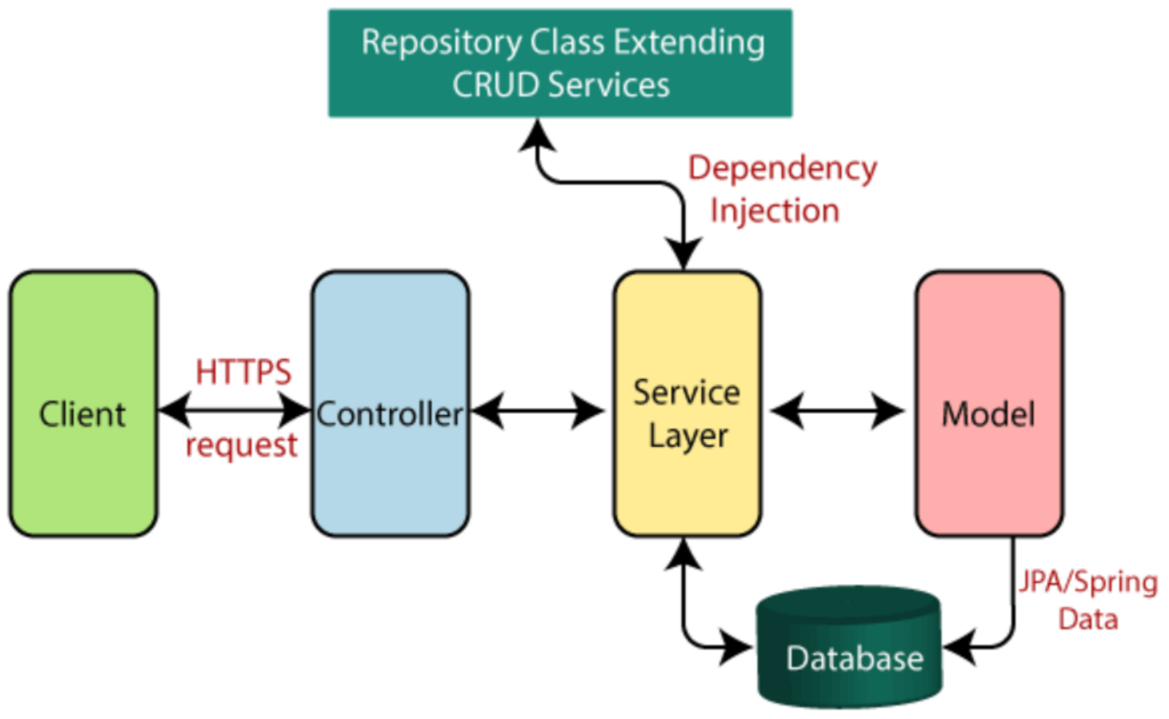
4. Steps to Build the Application

Here's a high-level overview of the steps involved:

1. **Setting Up the Project**:
   - Clone the GitHub repository provided in the tutorial.
   - Install Node.js, Java JDK, IntelliJ IDEA, and VS Code.
2. **Backend Development (Spring Boot)**:
   - Create a Spring Boot project.
   - Configure DB
   - Develop JPA entities and repositories.
   - Implement RESTful APIs for CRUD operations.
3. **Frontend Development (React)**:

- Create a React app using `create-react-app`.
- Add Bootstrap or other styling libraries.
- Build React components (e.g., list view, form).
- Consume Spring Boot APIs using Axios or fetch.

4. **Integration**:
   - Connect the React frontend to the Spring Boot backend.
   - Fetch data from the backend and display it in the UI.
   - Implement features like user registration, login, and post creation.
5. **Testing and Deployment**:
   - Test the application thoroughly.
   - Deploy the application to a hosting service (e.g., AWS, etc.).



DEPLOYMENT PHASE 1 (ARCHITECTURE OF BACKEND):-

PHASE1:

Let's first build the backend CRUD REST APIs using Spring Boot and MySQL database.
ADD MAVEN DEPENDENCIES

Maven dependencies are **external libraries that a Java project needs in order to compile, build, test, and/or run**

Dependencies can be direct or transitive, meaning that they can declare their own dependencies in a pom.xml file

Dependency scope is used to limit the transitivity of a dependency and to determine when a dependency is included in a classpath.

There are 6 scopes:

- compile
  This is the default scope, used if none is specified. Compile dependencies are available in all classpaths of a project. Furthermore, those dependencies are propagated to dependent projects.
- provided
  This is much like `compile`, but indicates you expect the JDK or a container to provide the dependency at runtime. For example, when building a web application for the Java Enterprise Edition, you would set the dependency on the Servlet API and related Java EE APIs to scope `provided` because the web container provides those classes. A dependency with this scope is added to the classpath used for compilation and test, but not the runtime classpath. It is not transitive.
- runtime
  This scope indicates that the dependency is not required for compilation, but is for execution. Maven includes a dependency with this scope in the runtime and test classpaths, but not the compile classpath.
- test
  This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases. This scope is not transitive. Typically this scope is used for test libraries such as JUnit and Mockito. It is also used for non-test libraries such as Apache Commons IO if those libraries are used in unit tests (src/test/java) but not in the model code (src/main/java).
- system
  This scope is similar to `provided` except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.
- import
  This scope is only supported on a dependency of type `pom` in the `<dependencyManagement>` section. It indicates the dependency is to be replaced with the effective list of dependencies in the specified POM's `<dependencyManagement>` section. Since they are replaced, dependencies with a scope of `import` do not actually participate in limiting the transitivity of a dependency.

BACKEND-SPRINGBOOT/MySql/Postman

(CREATE BACKEND SPRINGBOOT DEPENDENCIES IN MAVEN, ADD Spring Web, JPA and MySql)

Create JPA entities for SQL Database table: (SRC- MODEL)

```
package Prasoon.Fullstack_Backend.model;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.Id;

@Entity

public class User {

    @Id

    @GeneratedValue

    private Long id;

    private String username;

    private String name;

    private String email;

public Long getId() {

        return id;

    }

 public void setId(Long id) {

        this.id = id;

    }

 public String getUsername() {

        return username;

    }

  public void setUsername(String username) {

        this.username = username;

    }


    public String getName() {

        return name;

    }


    public void setName(String name) {

        this.name = name; }

   public String getEmail() {

        return email;

    }

public void setEmail(String email) {

        this.email = email; }}
```

Create User Repository (EXTENDS JPA)

```
package Prasoon.Fullstack_Backend.repository;

import  org.springframework.data.jpa.repository.JpaRepository;

import Prasoon.Fullstack_Backend.model.User;

public interface UserRepository extends JpaRepository<User,Long>{

}
```

Connect SpringBoot to MySql (Application.properties)

```
spring.jpa.hibernate.ddl-auto=update

spring.datasource.url=jdbc:mysql://localhost:3306/fullstack

spring.datasource.username=root

spring.datasource.password=mysql@123

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

server.port=8090   //POSTMAN LOCAL HTTP (CREATE C R U D Requests )
```

SQL Workbench

```
Create Database FullStack

Use FullStack

Select * from FullStack
```

ADD REST API'S TO BACKEND FOR C R U D OPERATIONS (UserController.java)

```
package Prasoon.Fullstack_Backend.controller;

import Prasoon.Fullstack_Backend.exception.UserNotFoundException;

import Prasoon.Fullstack_Backend.model.User;

import Prasoon.Fullstack_Backend.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.*;


import java.util.List;

@RestController

@CrossOrigin("http://localhost:3000")

public class UserController {


    @Autowired

    private UserRepository userRepository;


    @PostMapping("/user")

    User newUser(@RequestBody User newUser) {
        return userRepository.save(newUser);
    }

    @GetMapping("/users")
    List<User> getAllUsers() {
        return userRepository.findAll();
    }

    @GetMapping("/user/{id}")
    User getUserById(@PathVariable Long id) {
        return userRepository.findById(id)
                .orElseThrow(() -> new UserNotFoundException(id));
    }

    @PutMapping("/user/{id}")
    User updateUser(@RequestBody User newUser, @PathVariable Long id) {
        return userRepository.findById(id)
                .map(user -> {
                    user.setUsername(newUser.getUsername());
                    user.setName(newUser.getName());
                    user.setEmail(newUser.getEmail());
                    return userRepository.save(user);
                }).orElseThrow(() -> new UserNotFoundException(id));
    }

    @DeleteMapping("/user/{id}")
    String deleteUser(@PathVariable Long id){
        if(!userRepository.existsById(id)){
            throw new UserNotFoundException(id);
        }
        userRepository.deleteById(id);
        return  "User with id "+id+" has been deleted success.";
    }



}
```

Create Exceptions for GET Request (UserNotFoundException & UserNotFoundAdvice)

UserNotFoundException

```
package Prasoon.Fullstack_Backend.exception;


public class UserNotFoundException extends RuntimeException{
    public UserNotFoundException(Long id){
        super("Could not found the user with id "+ id);
    }
}
```

UserNotFoundAdvice

```
package Prasoon.Fullstack_Backend.exception;


import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.ResponseStatus;


import java.util.HashMap;
import java.util.Map;


@ControllerAdvice
public class UserNotFoundAdvice {



    @ResponseBody
    @ExceptionHandler(UserNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public Map<String,String> exceptionHandler(UserNotFoundException exception){


        Map<String,String> errorMap=new HashMap<>();
        errorMap.put("errorMessage",exception.getMessage());


        return errorMap;
```

```
   }

}
```

FullStackBackend.java

```
package Prasoon.Fullstack_Backend;


import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;


@SpringBootApplication

public class FullstackBackendApplication {


	public static void main(String[] args) {

			SpringApplication.run(FullstackBackendApplication.class, args);

	}


}
```

Postman

http://localhost:8090/user   (USED FOR CRUD OPERATIONS FROM FRONTEND TO BACKEND AND VICE VERSA)

POST: http://localhost:8090/user

Body->Raw>JSON

Payload:-

```
{

“Name”:” ”,

“Username”:” ”,

“Email”:” ”

}
```

DEPLOYMENT PHASE 1 (FRONTEND):-

STEP 1:- Build the React JS frontend application and consume CRUD REST APIs that were exposed by the Spring Boot application(*EmployeeController*)

```
npx create-react-app ems-frontend
```

Open React application is created. Next, let's understand important folders and files.

package.json - The package.json file contains all the required dependencies for our React JS project. Most importantly, you can check the current version of React that you are using. It has all the scripts to start, build, and eject our React app.

public folder - The public folder contains index.html. As React is used to build a single-page application, we have this single HTML file to render all our components. Basically, it's an HTML template. It has a div element with id as root and all our components are rendered in this div with index.html as a single page for the complete react app.

src folder- In this folder, we have all the global javascript and CSS files. All the different components that we will be building, sit here.

index.js - This is the top renderer of your React app.

node_modules - All the packages installed by NPM or Yarn will reside inside the node_modules folder.

App.js - The App.js file contains the definition of our App component which actually gets rendered in the browser and this is the root component.

DESIGN LAYOUT FOR THE HOME PAGE:-

Navbar.js(Layout)

```
import React, { useState } from "react";
import { Link } from "react-router-dom";
import "./Navbar.css";


const Navbar = () => {
  const [navbarOpen, setNavbarOpen] = useState(false);


  const handleToggle = () => {
    setNavbarOpen(!navbarOpen);
  };


  return (
    <nav className="navbar navbar-expand-lg navbar-dark bg-primary">
      <div className="container">
        <Link className="navbar-brand" to="/">
        Full Stack Application
      </Link>
      <button
        className="navbar-toggler"
        type="button"
        aria-controls="navbarSupportedContent"
        aria-expanded="false"
        aria-label="Toggle navigation"
        onClick={handleToggle}
      >
        <span className="navbar-toggler-icon"></span>
      </button>
      <div
        className={`collapse navbar-collapse ${navbarOpen ? "show" : ""}`}
        id="navbarSupportedContent"
      >
        <ul className="navbar-nav ms-auto mb-2 mb-lg-0">
          <li className="nav-item">
            <Link
              className="nav-link"
              to="/"
              onClick={() => setNavbarOpen(false)}
            >
              Home
            </Link>
          </li>
          <li className="nav-item">
            <Link
              className="nav-link"
              to="/adduser"
              onClick={() => setNavbarOpen(false)}
            >
              Add User
            </Link>
          </li>
        </ul>
        {/* <Link className="btn btn-outline-light" to="/adduser">
          Add User
        </Link> */}
      </div>
    </div>
  </nav>
  );

};


export default Navbar;
```

Navbar.css(Layout)

```
/* Navbar.css */


.navbar {

    padding: 0.5rem 1rem;

  }


  .navbar-brand {

    font-size: 1.5rem;

    font-weight: bold;

    color: #ffffff;

  }


  .navbar-toggler {

    border: none;

    background: transparent;

  }


  .navbar-toggler-icon {

    background-image: url("data:image/svg+xml,%3csvg viewBox='0 0 30 30' xmlns='http://www.w3.org/2000/svg'%3e%3cpath stroke='rgba(0, 0, 0, 0.5)' stroke-width='2' stroke-linecap='round' stroke-miterlimit='10' d='M4 7h22M4 15h22M4 23h22'/%3e%3c/svg%3e");

  }


  .navbar-nav {

    margin-left: auto;

  }


  .nav-item {

    margin: 0 10px;

  }


  .nav-link {

    color: #ffffff;

    font-size: 1.1rem;

    text-transform: uppercase;

    padding: 0.5rem 1rem;

    transition: color 0.3s ease;
```

```
 }

 .nav-link:hover {
   color: #ffd700;
 }

 .btn-outline-light {
   color: #ffffff;
   border-color: #ffffff;
   transition: color 0.3s ease, background-color 0.3s ease;
 }

 .btn-outline-light:hover {
   color: #282c34;
   background-color: #ffffff;
 }
```

Home.js(Pages)

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import { Link, useParams } from "react-router-dom";
import "./Home.css";

export default function Home() {
  const [users, setUsers] = useState([]);

  const { id } = useParams();

  useEffect(() => {
    loadUsers();
  }, []);

  const loadUsers = async () => {
    const result = await axios.get("http://localhost:8090/users");
    setUsers(result.data);
  };

  const deleteUser = async (id) => {
    await axios.delete(`http://localhost:8090/user/${id}`);
    loadUsers();
  };
  return (
    <div className="container home-container">
      <div className="py-4">
        <h1 className="text-center mb-4">CRUD APPLICATION</h1>
        <table className="table table-hover border shadow">
          <thead className="thead-dark">
            <tr>
              <th scope="col">S.N</th>
              <th scope="col">Name</th>
              <th scope="col">Username</th>
              <th scope="col">Email</th>
              <th scope="col">Action</th>
            </tr>
          </thead>
          <tbody>
            {users.map((user, index) => (
              <tr key={user.id}>
                <th scope="row">{index + 1}</th>
                <td>{user.name}</td>
                <td>{user.username}</td>
                <td>{user.email}</td>
                <td>
                  <Link className="btn btn-info mx-2" to={`/viewuser/${user.id}`}>
                    View
                  </Link>
                  <Link className="btn btn-warning mx-2" to={`/edituser/${user.id}`}>
                    Edit
                  </Link>
                  <button className="btn btn-danger mx-2" onClick={() => deleteUser(user.id)}>
                    Delete
                  </button>
                </td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
  );
}
```

Home.css(Pages)

```
/* Home.css */

.home-container {
    padding: 2rem;
    background-color: #f0f2f5;
    min-height: 100vh;
  }

  h1 {
    color: #343a40;
    font-weight: bold;
  }

  .table {
    background-color: #ffffff;
    border-radius: 10px;
    overflow: hidden;
  }

  .table-hover tbody tr:hover {
    background-color: #f1f1f1;
    cursor: pointer;
  }

  .thead-dark {
    background-color: #343a40;
    color: #ffffff;
  }

  .btn-info {
    background-color: #17a2b8;
    border-color: #17a2b8;
    color: #fff;
  }

  .btn-warning {
    background-color: #ffc107;
    border-color: #ffc107;
    color: #fff;
  }
  .btn-danger {
    background-color: #dc3545;
    border-color: #dc3545;
    color: #fff;
  }

  .btn-info:hover, .btn-warning:hover, .btn-danger:hover {
    opacity: 0.8;
  }
```

ROUTE FRONTEND TO BACKEND

APP. JS

```
import "./App.css";
import "../node_modules/bootstrap/dist/css/bootstrap.min.css";
import Navbar from "./layout/Navbar";
import Home from "./pages/Home";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import AddUser from "./users/AddUser";
import EditUser from "./users/EditUser";
import ViewUser from "./users/ViewUser";

function App() {
  return (
    <div className="App">
      <Router>
        <Navbar />

        <Routes>
          <Route exact path="/" element={<Home />} />
          <Route exact path="/adduser" element={<AddUser />} />
          <Route exact path="/edituser/:id" element={<EditUser />} />
          <Route exact path="/viewuser/:id" element={<ViewUser />} />
        </Routes>
      </Router>
    </div>
  );
}

export default App;
```

APP.CSS

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

PACKAGES REQUIRED (package.json)

```
{
  "name": "fullstack",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "bootstrap": "^3.4.1",
    "react": "^18.3.1",
    "react-bootstrap": "^2.10.4",
    "react-dom": "^18.3.1",
    "react-router-dom": "^6.24.1",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Step-by-Step Explanation of FullStackFrontend Project

1. Frontend (React)

Design Layout for the Home Page

- **Navbar Component (`Navbar.js`)**:
  - This component provides a navigation bar using React Bootstrap styles.
  - It includes links for "Home" and "Add User", which toggle the navbar open and closed on mobile devices.
  - Implemented using functional components and hooks (`useState` for managing state).
  - Styling is defined in `Navbar.css` for colors, toggler styles, and responsiveness.
- **Home Page (`Home.js`)**:
  - Displays a CRUD (Create, Read, Update, Delete) application interface.
  - Fetches user data from a backend server using Axios (`axios.get`).
  - Displays user data in a table format with options to view, edit, and delete users.
  - User deletion triggers an Axios `delete` request to the backend.
  - CSS styling (`Home.css`) defines table appearance, hover effects, and overall page layout.

Routing (`App.js`):

- Defines routes for different pages using `react-router-dom`.
- Imports `Navbar` component and sets up routes for `Home`, `AddUser`, `EditUser`, and `ViewUser` components.
- Each route corresponds to a specific URL path and renders the respective component.

CSS and Dependencies (`App.css` and `package.json`):

- `App.css` provides global styles for the entire React application.
- `package.json` lists dependencies including React, Bootstrap, React Router, Axios, and other necessary libraries.

2. Backend (Spring Boot)

Setup Backend with Spring Boot

- **Entity Class (`User.java`)**:
  - Represents a `User` entity with fields like `id`, `username`, `name`, and `email`.
  - Annotated with `@Entity` for JPA (Java Persistence API) mapping and `@Id`, `@GeneratedValue` for primary key generation.
  - Getter and setter methods are defined for each field.
- **Repository Interface (`UserRepository.java`)**:
  - Extends `JpaRepository<User, Long>` provided by Spring Data JPA.
  - Implements basic CRUD operations (`save`, `findAll`, `findById`, `deleteById`).

    - Allows interaction with the `User` entity stored in the database (`MySQL`).
- **Controller (`UserController.java`)**:
    - Handles HTTP requests (`@RestController`) from the frontend.
    - Uses `@Autowired` to inject `UserRepository` for database operations.
    - Defines endpoints (`@PostMapping`, `@GetMapping`, `@PutMapping`, `@DeleteMapping`) for CRUD operations.
    - Exception handling (`@ControllerAdvice`, `@ExceptionHandler`) for `UserNotFoundException`.
- **Exception Handling (`UserNotFoundException.java`, `UserNotFoundAdvice.java`)**:
    - Custom exception (`UserNotFoundException`) for handling cases where a user with a specific ID is not found.
    - Advice (`UserNotFoundAdvice`) to return appropriate HTTP status codes and error messages for these exceptions.

Application Properties (`application.properties`):

- Configures database connection (`spring.datasource.url`, `spring.datasource.username`, `spring.datasource.password`, etc.) for MySQL.
- Specifies Hibernate properties (`spring.jpa.hibernate.ddl-auto`, `spring.jpa.properties.hibernate.dialect`) for automatic table generation and SQL dialect.

Main Application Class (`FullstackBackendApplication.java`):

- Spring Boot application entry point (`@SpringBootApplication`).
- Uses `SpringApplication.run()` to launch the application.

3. Integration (Frontend to Backend)

- **Axios Requests**:
    - Frontend (React) communicates with the backend (Spring Boot) using Axios HTTP requests (`axios.get`, `axios.post`, `axios.put`, `axios.delete`).
    - Requests are made to specific endpoints (`/users`, `/user/{id}`, `/user`) defined in the `UserController` to perform CRUD operations.
- **REST API Testing (Postman)**:
    - Postman is used to test REST APIs (`POST`, `GET`, `PUT`, `DELETE`) exposed by the backend (`http://localhost:8090/user`).
    - APIs are tested for creating, retrieving, updating, and deleting user data stored in MySQL database.

Summary

The FullStackFrontend project integrates a React frontend with a Spring Boot backend to create a complete CRUD application. Frontend components like `Navbar` and `Home` interact with backend APIs (`UserController`) via Axios for managing user data. Both frontend and backend components are configured using appropriate dependencies (`package.json` for React and Maven for Spring Boot). The project uses MySQL as the database, Spring Data JPA for database operations, and includes comprehensive error handling and exception management.

REFERENCES:-

1) FreecodeCamp

2)Github - GitHub - vivekkakadiya/Organica: Full-stack e-commerce project built using Spring Boot, MySQL, and React.js.

3)Geeks for Geeks – Learn SpringBoot, Connect Springboot and React, Connect Springboot and Mysql, Full Stack Development guide.

4)Generative AI – Gemini.ai, perplexity.ai, nat.dev, copilot, chatgpt

5)Documentation – React,Java,JavaScript,React.js,Database Management System

6) Spring Boot + React JS + MySQL Project CRUD Example (javaguides.net)

7) What is Spring Data JPA? - GeeksforGeeks

8) Axios in React: A Guide for Beginners - GeeksforGeeks

9) Lombok Java - Javatpoint

10) Maven – Introduction to the Dependency Mechanism (apache.org)

11) Accordion | React Bootstrap (react-bootstrap.github.io)

12) A Complete Beginner's Guide to React Router (Including Router Hooks) (freecodecamp.org)

13) React Hooks - GeeksforGeeks

14) CRUD Operations – What is CRUD? (freecodecamp.org)

15) Spring Boot - Dependency Management - GeeksforGeeks

16) Spring Boot Annotations - javatpoint

17) React

18) React Bootstrap | React Bootstrap (react-bootstrap.github.io)

19) Postman API Platform | Sign Up for Free

20) Maven Repository: Search/Browse/Explore (mvnrepository.com)